

**Iterative Development  
Testing Approaches**  
*Managing Iterative Testing in an Agile Development  
Project*

**Version 1.3**

Created: January 29, 2002

Last Saved on: April 9, 2002

*By Bryan Campbell  
And  
Dr. Glenn Ray*

<b>Iterative Development Testing Approaches</b>	Version: 1.3
<i>Managing Iterative Testing in an Agile Development Project</i>	Last Saved Date: April 9, 2002
C:\Documents and Settings\Bryan.campbell\My Documents\Business\digitalESP\Articles\Iterative Development Testing Approaches v1.4.doc	

## Revision History

<b>Date</b>	<b>Version</b>	<b>Description</b>	<b>Author</b>
January 29, 2002	V1.0	Document Structure Created	Bryan Campbell
February 19, 2002	V1.1	Added content	Bryan Campbell
February 27, 2002	V1.2	Added content and updated structure	Bryan Campbell
March 5, 2002	V1.3	Corrected grammatical errors, updated references	Bryan Campbell
March 29, 2002	V1.4	Updated with additional learnings and synchronized with <u>Blending A Test Strategy Into an Iterative Development Project</u>	Bryan Campbell

<b>Iterative Development Testing Approaches</b>	Version: 1.3
<i>Managing Iterative Testing in an Agile Development Project</i>	Last Saved Date: April 9, 2002
C:\Documents and Settings\Bryan.campbell\My Documents\Business\digitalESP\Articles\Iterative Development Testing Approaches v1.4.doc	

## Table of Contents

<b>1. Abstract</b>	<b>5</b>
<b>2. Purpose</b>	<b>6</b>
<b>3. Overview</b>	<b>6</b>
3.1 Property Reinsurance System Project	6
3.2 Software Development Project Frameworks	7
3.2.1 Rational Unified Process	7
<i>Figure 1 - Rational Unified Process</i>	8
3.2.2 eXtreme Programming (XP)	10
<i>Figure 2 - eXtreme Programming (XP)</i>	11
3.3 Testing Within the Methodologies	12
<i>Figure 3 - RUP Implementation and Test Workflow</i>	12
3.3.1 The FedRe Example	13
<b>4. Defining a Framework</b>	<b>15</b>
4.1 Testing Layers	16
<i>Figure 4 - Testing Phases</i>	18
<b>5. Managing Defects</b>	<b>18</b>
<i>Figure 5 - PRS Iteration Schedule</i>	19
<i>Figure 6 - Defect Testing Cycle</i>	20
<b>6. Learnings</b>	<b>20</b>
6.1 Define a Test Strategy	20
6.2 Test Layers	21
6.3 Test Coordinator	21
6.4 Testing Buffers Between Iterations	21
6.5 Automated Test Tools	22
<b>7. Conclusion</b>	<b>22</b>
<b>8. About the Authors</b>	<b>23</b>
8.1 Bryan Campbell	23
8.2 Dr. Glenn Ray	23

<b>Iterative Development Testing Approaches</b>	Version: 1.3
<i>Managing Iterative Testing in an Agile Development Project</i>	Last Saved Date: April 9, 2002
C:\Documents and Settings\Bryan.campbell\My Documents\Business\digitalESP\Articles\Iterative Development Testing Approaches v1.4.doc	

## 9. About digitalESP Inc.

23

<b>Iterative Development Testing Approaches</b>	Version: 1.3
<i>Managing Iterative Testing in an Agile Development Project</i>	Last Saved Date: April 9, 2002
C:\Documents and Settings\Bryan.campbell\My Documents\Business\digitalESP\Articles\Iterative Development Testing Approaches v1.4.doc	

## 1. Abstract

Over the past five years, there has been increasing interest in agile development approaches to software development (such as eXtreme Programming), however, integrating these into a unified testing approach can be challenging given their rapid delivery environment. Applying a testing framework to an agile development approach provides a greater opportunity to ensure a robust and high quality application. This paper reviews a web-services software development project completed at the end of the year 2000 for a large Fortune 500 company. The project used an object-oriented design and blended the more formal Rational Unified Process (RUP) with the low ceremony approach advocated by eXtreme Programming (XP). The application testing approach applied the unit test framework of XP with a formal testing methodology required by the client for this high profile project. The paper describes the process followed and key learnings discovered throughout the project lifecycle.

<b>Iterative Development Testing Approaches</b>	Version: 1.3
<i>Managing Iterative Testing in an Agile Development Project</i>	Last Saved Date: April 9, 2002
C:\Documents and Settings\Bryan.campbell\My Documents\Business\digitalESP\Articles\Iterative Development Testing Approaches v1.4.doc	

## 2. Purpose

The Property Reinsurance System (PRS) described in this document refers to a ‘replacement and re-engineering’ project deployed for a Fortune 500 company over a 12-month period from October 2000 to October 2001. The purpose of this document is to determine some best practices for applying a comprehensive testing strategy to an iterative development project. The project blended approaches of two leading, open methodologies, the Rational Unified Process (RUP) and eXtreme Programming (XP). One of the challenges discovered throughout the project was in maintaining a disciplined testing schedule throughout a series of back-to-back iterations. As a large-scale web development effort dealing with a revenue generating application, testing to ensure a robust and reliable application was paramount. This document describes key learnings and proposed solutions to integrating a rigorous testing cycle into an iterative development project.

## 3. Overview

### 3.1 Property Reinsurance System Project

In October 2000, the customer, FedRe Inc., initiated the development of a new software application known as the Property Reinsurance System (PRS). The application was intended to be an object oriented, web based application, which could provide real-time property re-insurance quotes to FedRe customers. As part of the development process, FedRe identified, as one of its requirements, that a software development methodology known as the Rational Unified Process (RUP) be used throughout the lifecycle of the project. As the consulting company responsible for delivering the project within a fixed time and price contract the Rational Unified Process was also complemented by a more recent methodology known as eXtreme Programming (XP).

**Comment:** We might need to develop a similar naming convention for AmRe itself, FedRe might not be terribly appropriate but it was the best I could come up with ☺

PRS was intended to replace an existing client-server application that had been developed in the early 1980s for a specific customer and overtime had grown into a product used by a large number of FedRe’s customers. The project had several stated objectives and constraints:

1. Develop a new version of PRS that was an n-tiered, web services based application. Web services refer to exposing core application services through an XML interface using an HTTP transport layer.
2. Develop the system using a Microsoft toolset (Visual Basic 6.0, ADO 2.5 and ASP 3.0) that fitted the stated architectural direction of FedRe and its parent organization.
3. Apply an iterative development process to the application based on the Rational Unified Process (RUP) and eXtreme Programming (XP).
4. Develop new functionality, not present in the current client-server version of PRS,

<b>Iterative Development Testing Approaches</b>	Version: 1.3
<i>Managing Iterative Testing in an Agile Development Project</i>	Last Saved Date: April 9, 2002
C:\Documents and Settings\Bryan.campbell\My Documents\Business\digitalESP\Articles\Iterative Development Testing Approaches v1.4.doc	

and examine re-engineering opportunities in the process of developing the system.

Through the Rational Unified Process the project was delivered in a series of iterations over four phases (Inception, Elaboration, Construction and Transition). The project was a \$3 Million U.S. initiative with a twelve month development period and involved a team of 11 full time members (one project manager, two system architects and eight developers). Approximately 130 customers used the system generating more than \$40 Million in revenue per year.

## 3.2 Software Development Project Frameworks

Within the software development industry there are a variety of different project methodologies for developing software systems. An increasing trend in the industry has been the introduction of an iterative approach to software development. Iterative software development focuses on creating functional components of software in short and consistent periods of time. These iterations focus on ensuring high levels of visibility into the software development and provide greater flexibility for changes to user requirements. There are two primary methodologies representing two very different schools of thought that have achieved prominence in the management of software development projects:

### 3.2.1 Rational Unified Process

The Rational Unified Process<sup>i</sup> (RUP) undeniably ranks among the leading software development processes available today. RUP evolved from the pioneering work of the ‘Three Amigos’ (Grady Booch, James Rumbaugh and Ivar Jacobson) who aligned their individual efforts to develop an object oriented, visual modeling process into a comprehensive approach with a common language, Universal Modeling Language (UML) notation. Currently supported by Rational Software Inc. the methodology is distributed on a CD in HTML format and is tightly coupled with some of the leading object oriented development tools such as Rational Rose and Requisite Pro. RUP contains a wealth of information, including process roles, activities, recommended artifacts and document templates.

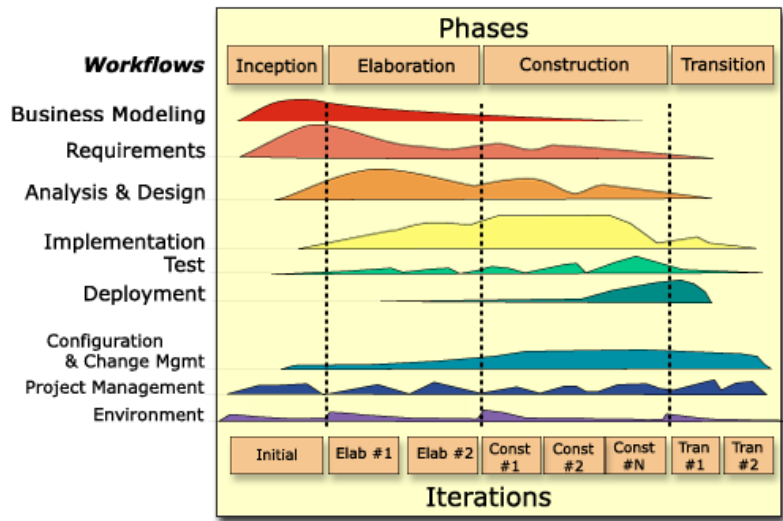
RUP is described as a use-case driven, architecture-centric, iterative and incremental methodology. Functional requirements are captured in use cases that drive the development process. Specified use cases drive the analysis & design and test workflows. Use cases also drive the architecture, which in turn, influences the selection of use cases. This interplay between use cases and architecture evolves in the context of an iterative and incremental process. A project is divided into a series of iterations where the most architecturally significant and/or technically complex use cases are tackled initially. During early project iterations, some previous work may be discarded but later iterations add a greater proportion of incremental functionality.

<b>Iterative Development Testing Approaches</b>	Version: 1.3
<i>Managing Iterative Testing in an Agile Development Project</i>	Last Saved Date: April 9, 2002
C:\Documents and Settings\Bryan.campbell\My Documents\Business\digitalESP\Articles\Iterative Development Testing Approaches v1.4.doc	

The motivation for a process like RUP is based on the argument that iterative processes reduce project risk. Requirements that change over time are more easily incorporated into an iterative process, and management gains greater visibility into the current status of the project. Project cost is reevaluated at the end of each iteration. By focusing on essential requirements first, a functional system can be delivered sooner with deferral of lower priority functionality to subsequent releases.

As seen in Figure 1, RUP is composed of four phases and nine core workflows. The phases are Inception, Elaboration, Construction and Transition. The workflows are Business modeling, Requirements, Analysis and Design, Implementation, Test, Deployment, Configuration & change management, Project management, and Environment. Phases are subdivided into iterations during which the workflows are performed. Each iteration essentially applies the core workflows in what could be constituted a mini-waterfall process. So RUP can be viewed as a succession of short waterfall processes during which a system is built incrementally. Scott Ambler aptly describes this type of process as serial in the large-scale and iterative in the small-scale<sup>ii</sup>.

**Figure 1 - Rational Unified Process**



During the Inception phase the project scope is defined and the business case is justified. High-level use cases are developed for the system’s primary functional requirements. Project management tasks include conducting an initial risk assessment, and estimating resources, schedule and overall cost.

<b>Iterative Development Testing Approaches</b>	Version: 1.3
<i>Managing Iterative Testing in an Agile Development Project</i>	Last Saved Date: April 9, 2002
C:\Documents and Settings\Bryan.campbell\My Documents\Business\digitalESP\Articles\Iterative Development Testing Approaches v1.4.doc	

During Elaboration, the problem domain is subjected to detailed analysis and the architectural foundation is established. A supplementary specification is developed for non-functional requirements.

In the Construction phase, the detailed design is realized and implemented iteratively. The objective is to produce a functional system by the end of each iteration. Supporting documentation is also developed iteratively.

The Transition phase results in the delivery of the system to the user community. This commonly occurs after a beta release has been tested by end users and debugged.

RUP's strengths lie in its foundation as an iterative process that is requirements driven and architecture-centric. A working system is delivered at the end of each iteration, which act as important control gates on the progress of the project. The process is extensively documented; all model artifacts are based on UML, and supported by a suite of automated tools. It is a robust process framework that can be tailored to project needs.

Nevertheless, there are deficiencies in RUP. Ambler notes that it is essentially a single-project development process. It fails to consider enterprise-level, multi-project issues. As a result, application architectures may fragment across the enterprise and opportunities for software reuse may be lost. It does not address the post-Transition phase concerns of operational support and maintenance, which over a system lifetime probably consume more total resources than initial development. Since it is based on UML, non-UML model artifacts are not covered and the process documentation can consume expensive resources. In Ambler's opinion, the marketing of Rational's commercial tools dominates the process. Other weak areas include metrics, testing and the crucial area of human resource management.

Hesse's criticisms of RUP are more fundamental<sup>iii</sup>. He argues that phases, which are a holdover from the classic waterfall process, are not an appropriate structuring concept for Object Oriented projects. Instead, process activities and iterations should be linked to architecturally significant milestones. Workflow is an overloaded term and its relationship to phases is unnecessarily complex. RUP lacks high-level support for structuring complex processes such as hierarchy, recursion and orthogonality. RUP should be decomposable into sub-processes. There is inadequate support for project management and roles/interactions of project groups, especially in the area of user feedback.

In addition to these criticisms, RUP also lacks detailed advice on the Implementation workflow. Since coding is arguably the most important project activity, a software development team needs comprehensive guidance on how to most efficiently manage the actual coding of the system.

<b>Iterative Development Testing Approaches</b>	Version: 1.3
<i>Managing Iterative Testing in an Agile Development Project</i>	Last Saved Date: April 9, 2002
C:\Documents and Settings\Bryan.campbell\My Documents\Business\digitalESP\Articles\Iterative Development Testing Approaches v1.4.doc	

### 3.2.2 eXtreme Programming (XP)

Kent Beck<sup>iv</sup> started XP about six years ago to develop a lightweight approach to the ‘high ceremony’ process of the RUP methodology. This approach has been emulated by others including Alistair Cockburn’s Crystal family of methodologies<sup>v</sup>, Jim Highsmith’s Adaptive Software Development<sup>vi</sup>, SCRUM<sup>vii</sup>, Scott Ambler’s Agile Modeling<sup>viii</sup>. More recently, the leading proponents of lightweight methodologies have formed the Agile Alliance, an umbrella organization that promotes the principles of lightweight processes. A growing number of websites, magazine articles, books and conferences devoted to agile methods attest to their current popularity among programmers.

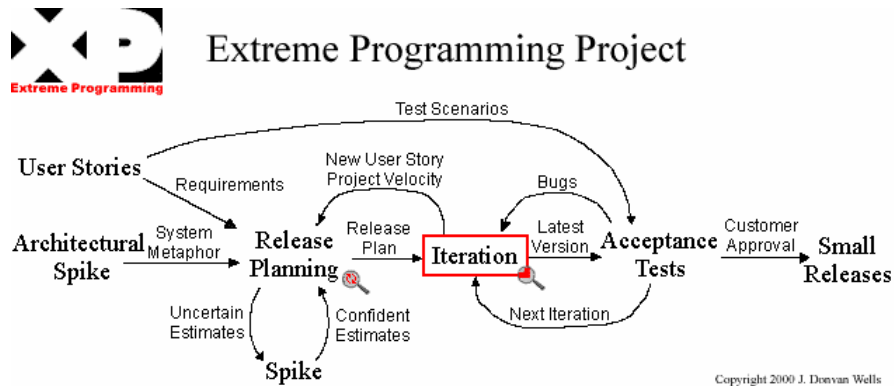
XP promises a high level of customer satisfaction based on four process principles: communication, simplicity, feedback and courage. To gather requirements, it supports communication within a development team that includes client staff, and it encourages the use of the simplest design and implementation that meets those requirements. Feedback is obtained through extensive unit testing during each iteration. In fact, unit tests should be written prior to actual coding. With this foundation of teamwork, simplicity, and automated testing, XP claims that developers have the courage to respond to unexpected requirements changes.

It’s probably more accurate to classify XP as a collection of practical programming techniques than a complete software development process. Figure 2 depicts the process flow for an XP project. Requirements are captured in user stories, which are essentially high-level use case descriptions without detailed, step-by-step functional details. The XP website provides the following description of user stories:

*User stories serve the same purpose as use cases but are not the same. They are used to create time estimates for the release planning meeting. They are also used instead of a large requirements document. User Stories are written by the customers as things that the system needs to do for them. They are similar to usage scenarios, except that they are not limited to describing a user interface. They are in the format of about three sentences of text written by the customer in the customer’s terminology without techno-syntax.*

<b>Iterative Development Testing Approaches</b>	Version: 1.3
<i>Managing Iterative Testing in an Agile Development Project</i>	Last Saved Date: April 9, 2002
C:\Documents and Settings\Bryan.campbell\My Documents\Business\digitalESP\Articles\Iterative Development Testing Approaches v1.4.doc	

Figure 2 - eXtreme Programming (XP)



As Figure 2 shows, each iteration is preceded by a Release Planning activity that describes the work to be done based on the following inputs: requirements (User Stories), high-level architectural analysis (Architectural Spike) and throw-away programs that explore particularly difficult areas (Spikes). The iteration proceeds after confident estimates are obtained and the code is debugged until it passes all acceptance tests. The result of an iteration is the release of a functional sub-system. Eventually, the complete system is developed through this iterative process.

XP is an iterative process designed for small (2-12 person) teams. Developers, who work in pairs, write unit tests prior to writing the actual code and they review each other's code. The code should be the simplest possible implementation of the requirements and it should be refactored<sup>1</sup> repeatedly.

In reaction to heavyweight processes that have an elaborate management structure and produce extensive documentation artifacts, XP deliberately minimizes these process activities. Project management is viewed as inefficient bureaucracy, and code is regarded to be the only documentation that is inherently reliable. These policies are justified with claims that modern software development, especially internet applications, can not be developed with heavyweight processes when requirements are unpredictable and delivery timeframes are short.

XP's strength is its unflinching focus on getting the code written so that each iteration yields a functional product. It is a hands-on approach driven by developers with direct client involvement to foster communication, much of which is informal. Product quality is achieved through pair programming and

<sup>1</sup> Refactoring involves improving the internal structure of software without changing its observable behaviour (e.g. making it more efficient)

<b>Iterative Development Testing Approaches</b>	Version: 1.3
<i>Managing Iterative Testing in an Agile Development Project</i>	Last Saved Date: April 9, 2002
C:\Documents and Settings\Bryan.campbell\My Documents\Business\digitalESP\Articles\Iterative Development Testing Approaches v1.4.doc	

automated unit testing.

XP does have limitations. Like RUP, it is not an enterprise process. In fact, its focus is really in RUP's construction phase where code is being planned and implemented. The claim that pair programming is more productive is debatable. It does not provide clear guidance on what models or documentation artifacts should be developed. With minimal management and documentation activities, the ability of XP to scale to enterprise level projects is questionable. XP also has the potential to make the support and maintenance of a system difficult with its lack of emphasis on formal documentation.

### 3.3 Testing Within the Methodologies

Both RUP and XP include a heavy focus on testing throughout their methodology framework. Within RUP a test workflow exists that provides an activity diagram that illustrates how to develop and support a test management strategy for all elements within the project. XP focuses on testing more at the unit level and is based on the assumption that no code can be released until it has passed its basic unit tests.

Within RUP, the testing workflow mirrors the development workflow (Figure 3) with increasing amounts of testing activity as the project moves through the Elaboration and Construction phases. Interestingly, the amount of testing effort is seen to decrease in the Transition phase even though the RUP process supports regression testing<sup>2</sup> throughout its lifecycle. RUP provides an excellent overview of the testing process including key concepts such as coverage measures (both code and requirements) and quality measures (such as defect tracking and trend analysis).

Figure 3 - RUP Implementation and Test Workflow



Both XP and RUP emphasize the need for automating testing particularly for regression testing of the application (testing all test cases with the release of a new iteration). RUP uses the concept of a test case to validate functional testing of the system which is developed from the use cases documenting system functionality. XP bases its testing structure on acceptance tests (formerly known as functional tests) which capture 'black box' test results (validating external outputs not internal component testing of code). XP advocates creating acceptance tests from user stories that are similar to RUP's use cases but are shorter (typically "three sentences of text written by the customer"). However, XP does not provide any guidelines on

<sup>2</sup> Regression testing is the retesting of a software system that has been modified to ensure that any bugs have been fixed and that no other previously-working functions have failed as a result of the reparations and that newly added features have not created problems with previous versions of the software.

<b>Iterative Development Testing Approaches</b>	Version: 1.3
<i>Managing Iterative Testing in an Agile Development Project</i>	Last Saved Date: April 9, 2002
C:\Documents and Settings\Bryan.campbell\My Documents\Business\digitalESP\Articles\Iterative Development Testing Approaches v1.4.doc	

managing the testing sequence beyond the code level and its primary emphasis on managing and correcting defects is with the developers. Developers are tasked with the responsibility of scheduling time each iteration to correct defects, which are based on acceptance test scores. XP lacks the holistic approach to testing that the RUP methodology provides and does not go into the detail on the organizational alignment required for the project.

The one most significant deficiency of both RUP and XP regard their defect correction from a previous iteration within a current iteration. As the remainder of this paper will illustrate, managing defects within an iterative process become one of the most complex elements of a project. The challenge lies in ensuring that defects identified outside of the tests executed by developers (during User Acceptance testing for example) can be adequately factored into the development cycle and corrected.

### **3.3.1 The FedRe Example**

There are some particularly challenging aspects of iterative processes that should be noted. A commonly touted advantage is that iterative processes avoid the unpredictably long testing period at project end that are seen in waterfall development methodologies. Admittedly, it is less expensive to correct a defect early in the project than later. Nevertheless, testing remains a major resource commitment that continually threatens the project schedule. In the PRS project even with automated unit testing, it was very difficult for the developers to finish the iteration build in time for adequate integration testing particularly with regression testing.

In the PRS project, developers were scheduled to deliver a build early in the final week of the iteration. The analyst/architect then performed integration testing. Bugs were logged in a Lotus Notes database and the developers fixed bugs generating a new build at least daily. This cycle was repeated until Thursday afternoon at which time the system was frozen for the Friday demo to the client. Unfortunately due to the ambitious workload, the initial build was often delivered later in the week. This prevented the analyst/architect from conducting thorough testing, and the developers did not have enough time to fix all the bugs. Although the primary functionality of the system was available at the end of each iteration (as represented through the basic flow of each use case) a number of known defects were present in the build as it was delivered to the client's QA department.

As a result, it was not possible to achieve the level of stability required and some known bugs were carried over to the next iteration. This ongoing deferment of bug fixes to later iterations became a major impediment and resulted in a longer than planned stabilization period after iteration nine. To some degree this necessitated an iteration ten, although this was also a reflection of a significant architectural change associated with an off-site

<b>Iterative Development Testing Approaches</b>	Version: 1.3
<i>Managing Iterative Testing in an Agile Development Project</i>	Last Saved Date: April 9, 2002
C:\Documents and Settings\Bryan.campbell\My Documents\Business\digitalESP\Articles\Iterative Development Testing Approaches v1.4.doc	

hosing service. We feel this problem can be partially addressed by automated integration testing. To successfully use automated testing, it is crucial that the GUI be stabilized as early as possible. Unfortunately, this was not possible because the usability expert joined the project very late resulting in extensive revisions to the GUI.

Another mitigating technique is to schedule a buffer period for extra testing periodically throughout the project, a practice employed by Microsoft<sup>ix</sup>. The idea is to designate certain iterations as major release points which are allotted an extra period of time to stabilize the application with extra testing. We believe that inserting a buffer period after every third iteration, for example, is merited given the scheduling challenges. For a one-month iteration, we recommend a buffer period of at least one week.

Within FedRe, the client had already developed a formal testing methodology that had been applied to a series of completed projects, primarily waterfall based development efforts. Prior to deployment, applications required ‘certification’ from the Quality Assurance Department, which acted as an independent and objective testing group for all applications. Due to the sensitive nature of the PRS application, testing was seen as a critical component of the overall project plan.

The challenge then became how to integrate the testing captured by the application development team in its iterative development cycle and appropriately blend it with an independent testing group. To further compound matters, the independent testing group had a strong desire to be more actively involved in the full project lifecycle, specifically the Inception, Elaboration, Construction and Transition phases. This meant applying their testing structure to each completed iteration. However, the testing group was more familiar with a waterfall development approach that provided greater application functionality and much more stable code base when they applied their testing.<sup>3</sup>

In addition, agile development techniques advocate developing a unit test framework that provide good code level testing and help manage. However, this testing approach is not exhaustive, particularly for functional and system level testing. While the Rational Unified Process provides a more comprehensive overview for testing, it is deliberately high level and requires more ‘real-world’ grounding. The remainder of the document describes how to integrate testing across both methodologies and how to align it with an independent testing group.

---

<sup>3</sup> The waterfall approach, however, has also proved to be too inflexible for changing business requirements which often occur over longer project lifecycles

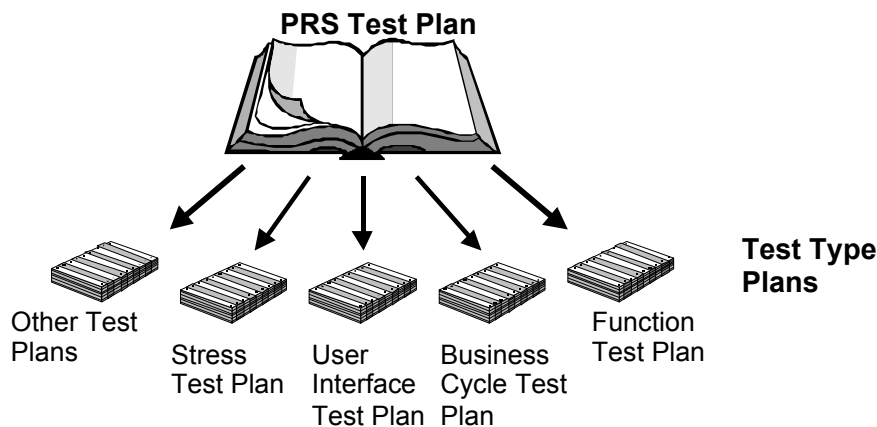
<b>Iterative Development Testing Approaches</b>	Version: 1.3
<i>Managing Iterative Testing in an Agile Development Project</i>	Last Saved Date: April 9, 2002
C:\Documents and Settings\Bryan.campbell\My Documents\Business\digitalESP\Articles\Iterative Development Testing Approaches v1.4.doc	

## 4. Defining a Framework

To successfully apply a testing framework within an iterative development project it is important to explicitly document what you want to test and determine who is going to do the testing. To help align the testing of the PRS project and make it more explicit to the project stakeholders (which included their Internal Audit department), a comprehensive Test Plan was developed that encapsulated all testing within the system. While this artifact was similar to the product identified in the Rational Unified Process it was focused more on showing testing as a separate workflow within the application development and it collapsed many of the test types into more manageable groupings. The purpose of application testing was:

- To verify the interaction between objects.
- To verify the proper integration of all components of the software.
- To verify that all requirements have been correctly implemented.
- To identify and ensure defects are addressed prior to the deployment of the software

The Test Plan acted as a Master Guide to testing within the PRS project and it in turn supported the details of Test Type Plans developed to capture all of the testing requirements of the application. Conceptually, the Test Plan can be seen in the following way:



The test type plans contained the actual details on the tests developed for the system. This documentation was used for regression testing of application functionality and for automation of testing using tools. While the Unit tests advocated in XP provide good white box (structural) testing at the code level these still need to be complemented by black box (functional tests) and appropriate component level testing.

<b>Iterative Development Testing Approaches</b>	Version: 1.3
<i>Managing Iterative Testing in an Agile Development Project</i>	Last Saved Date: April 9, 2002
C:\Documents and Settings\Bryan.campbell\My Documents\Business\digitalESP\Articles\Iterative Development Testing Approaches v1.4.doc	

Within the PRS Project specific testing areas that were be addressed included **Functional Tests** (driven by use cases), **User Interface Tests** (driven by User Interface Design Standards), **Performance Tests** (load, stress and performance related), **Data Integrity Tests** and **Security Tests**. Developers are still required to create unit tests and these tests must be successfully passed before being integrated. While developing a test plan of this nature might feel like a lot of work, the payoff is knowing that your testing the application from a variety of perspectives.

As Steven Splaine states in his Web Testing Handbook<sup>x</sup>

*In our experience, the easiest way to plan (or design) the testing of a Web site/application is to develop a series of Test Plans. These plans help ensure that the Web site/application is comprehensively tested and can be reliably re-tested. Each test plan is focused on a different phase of the testing process (e.g., unit testing, system testing, or post-implementation testing), while the individual test cases within each test plan are grouped into categories that focus on a single aspect of testing.*

## 4.1 Testing Layers

The testing lifecycle is an integral part of the software lifecycle; they should start at the same time. The design and development process for tests is as complex and arduous as the application being built. If not started early enough, the tests will either be deficient, or cause a long testing and bug-fixing period to be appended to the development schedule, which defeats the goals of iterative development.

Furthermore, the test planning and design activities can expose faults or flaws in the application design. The earlier these are resolved the lower the impact on the overall schedule. Problems found during evaluation can be solved within this iteration, or postponed to the next iteration. To address these challenges it is important to test in *layers*.

The test plan for the PRS project consisted of four primary testing layers: **Coding, Integration, Quality Assurance and User Acceptance Testing**. Each of these layers progressively built upon the other to ensure as many defects as possible had been discovered and corrected. Since test details for the PRS project were all use case driven they reflect traceability from use cases to the design model to the test cases of the system. The details of the testing are as follows:

**Development Level Testing:** The development of the application used a Unit Test Framework to ensure that at least 50% of the bugs present in the application were captured and corrected prior to moving the code to the next level of testing. The Unit Test Framework was developed by implementers throughout the development effort of each iteration. Unit tests are automated tests that verify the internal structure of methods and validate the expected results that these methods are to provide. Designing unit tests allows the code in the application to be frequently tested and makes it easier to provide regression testing throughout the life of the project.

<b>Iterative Development Testing Approaches</b>	Version: 1.3
<i>Managing Iterative Testing in an Agile Development Project</i>	Last Saved Date: April 9, 2002
C:\Documents and Settings\Bryan.campbell\My Documents\Business\digitalESP\Articles\Iterative Development Testing Approaches v1.4.doc	

As advocated by Karlson, Andersson and Leion, the developers had their own private environment where the builds and testing could occur prior to submitting to the integration environment. As the authors recommend the PRS project followed “a code review, compile, load and some form of regression test before submitting the code. The full regression test of old functionality is completed centrally”<sup>xi</sup>.

Unit tests focus on small testable elements of the software that independently might not provide appropriate testing coverage but used in conjunction with one another provide decision-to-decision path testing for the entire application.

All unit tests had to be passed before code was released.

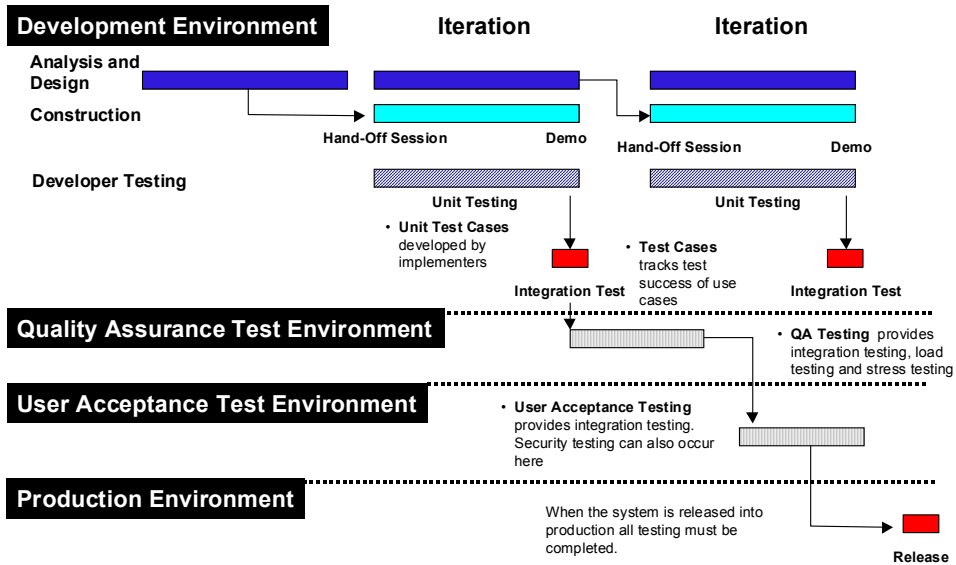
**Integration Testing:** Integration testing involves the creation of a test matrix and related test cases to test the functionality of groupings of classes and methods. The integration testing was conducted by the Architect, System Analyst and Code Reviewer of the project and followed a use case flow with a standard set of inputs and expected outputs. This level of testing was completed before the demo held at the completion of each iteration. At this point it was estimated that approximately 80% of the bugs present in the system were corrected but 100% of the documented functionality is available for testing.

**Quality Assurance Testing:** After integration testing, the PRS application was migrated to the independent testing area in the customer’s Quality Assurance department. This level of testing was conducted in a separate physical environment from the development environment. At this level of testing, integration between use cases was tested and non-functional testing such as load testing, stress testing and security testing was conducted. Defects discovered from this testing were provided to the development team for inclusion in the upcoming iteration. At the completion of QA testing 90% of the system bugs should be corrected.

**User Acceptance Testing:** Once the system has satisfactorily passed the previous three levels of testing it was moved into a separate physical environment for User Acceptance Testing. This type of testing was typically performed prior to releasing the system into production and it involves system users developing and executing their own test cases to validate the functionality of the system. Oftentimes, system users require assistance in developing these test cases and in following through on their execution. This assistance was provided to staff through the development of templates and in direct mentoring and support.

This form of testing is similar to that promoted by the Department of Defense<sup>xii</sup> which describes three levels of tests; low level, software integration and hardware/software integration.

**Figure 4 - Testing Phases**



As Figure 4 shows the different phases of testing (Development, Quality Assurance and User Acceptance) all added incremental amounts of time to the testing process. This was particularly influenced by the waterfall nature of the testing that the client required. QA would not begin testing until development testing was completed and User Acceptance testing was not allowed to begin until QA testing was completed.

**5. Managing Defects**

Probably one of the biggest challenges posed by an iterative development process is the task of managing defects. Defects are a natural result of software development and the key to effective software development is ensuring that these defects are captured and corrected before moving an application into production. However, testing needs to occur at much more than just the developer and code level. Users need to apply their own testing to an application and using an independent testing assessment also offers greater testing coverage to an application. Unfortunately, the challenge of managing defects with a series of back-to-back iterations can be particularly challenging.

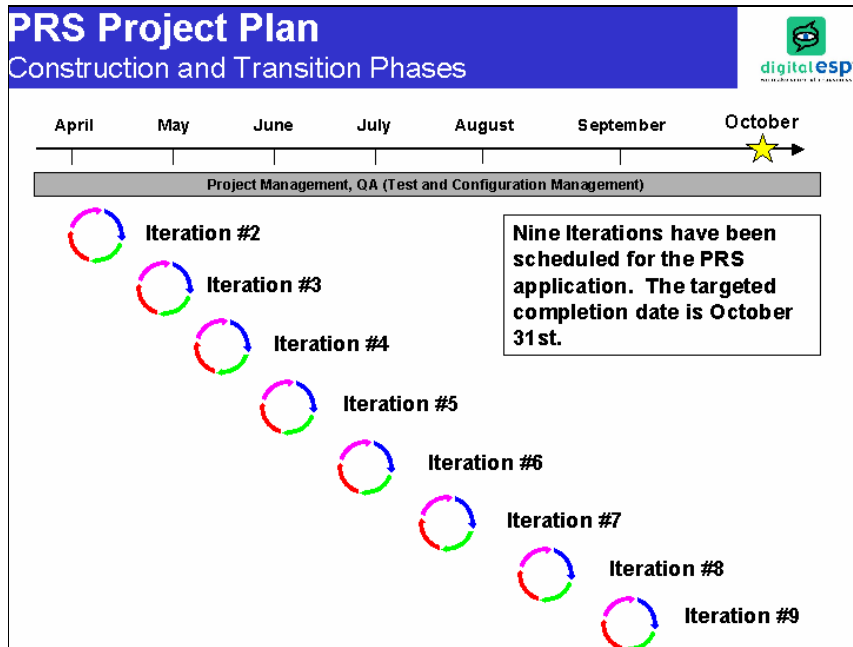
Neither RUP nor eXtreme Programming offers much insight into managing defects throughout a project with a large number of scheduled iterations. The challenge that rapidly becomes apparent is how to schedule defect correction within a defined iteration.

The best way to illustrate this is through an example. The PRS project approached the project as a series of nine iterations (Figure 5). An iteration routemap (a tool that

<b>Iterative Development Testing Approaches</b>	Version: 1.3
<i>Managing Iterative Testing in an Agile Development Project</i>	Last Saved Date: April 9, 2002
C:\Documents and Settings\Bryan.campbell\My Documents\Business\digitalESP\Articles\Iterative Development Testing Approaches v1.4.doc	

prioritizes uses cases based on complexity and risk and maps these to successive iterations) was used to identify functionality scheduled for each iteration. Each iteration was four weeks long and progressively built on functionality provided in the iteration previous.

Figure 5 - PRS Iteration Schedule



However, testing completed at the end of an iteration consisted only of unit testing and integration testing (test cases derived from use cases). This testing did not permit adequate user testing which often identifies not only functional defects but also functional omissions (requirements not reflected in use cases). Once the users and, also the independent testing group, were able to test the application any new defects logged could only get addressed in the current iteration (see Figure 4 for an illustration). Unfortunately, even with the flexibility for requirements change in an iterative schedule, most Severity One or Two defects discovered in the last two weeks of an iteration had to be deferred to the following iteration.

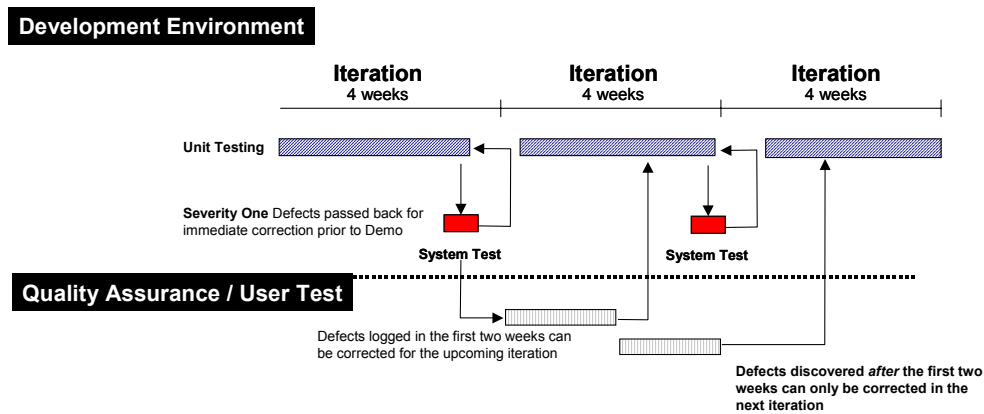
As a result, tracking the iteration in which a defect was being corrected required significant coordination. The process also required good communication as users and the independent testing team often had challenges in waiting upwards of two months to see their defects get corrected. This was in addition to the traditional challenge of communicating to users that not all functionality for the system would be available in

<b>Iterative Development Testing Approaches</b>	Version: 1.3
<i>Managing Iterative Testing in an Agile Development Project</i>	Last Saved Date: April 9, 2002
C:\Documents and Settings\Bryan.campbell\My Documents\Business\digitalESP\Articles\Iterative Development Testing Approaches v1.4.doc	

each iteration and that this did not necessarily constitute a defect<sup>4</sup>.

The challenge became managing the feedback loop from testers into the iterative schedule of the application developers. Often Severity One defects (defects that prevented the system from functioning) required an Architect to review and assess any design changes that might be required. This further impacted the ability of architects to complete analysis design on the iteration they were working on.

Figure 6 - Defect Testing Cycle



## 6. Learnings

### 6.1 Define a Test Strategy

Creating a Test Strategy to define what should be tested and who should perform the testing is critical to developing a thorough testing approach for an application. The RUP test plan artifact is a good starting point for developing an overall strategy. The Unit tests advocated in XP provide good white box (structural) testing at the code level, however, these still need to be complemented by black box (functional tests) and appropriate component level testing. Within the PRS project the test plans identified testing for several key areas common in most application development projects: Functional Tests (driven by use cases), User Interface Tests (based on User Interface Design standards), Performance Tests (load, stress and performance related), Data Integrity Tests and Security Tests. Developers are still required to create unit tests and these tests must be successfully passed before being integrated., however these additional test ensure the application is thoroughly tested before deployment.

<sup>4</sup> When defects of this nature were identified by users, the testing coordinator would classify these as 'pending' to illustrate that the functionality was forthcoming

<b>Iterative Development Testing Approaches</b>	Version: 1.3
<i>Managing Iterative Testing in an Agile Development Project</i>	Last Saved Date: April 9, 2002
C:\Documents and Settings\Bryan.campbell\My Documents\Business\digitalESP\Articles\Iterative Development Testing Approaches v1.4.doc	

## 6.2 Test Layers

Both RUP and XP place a great deal of emphasis on the developers to test the application. This makes sense from the perspective that they know the code and they should be responsible for fixing their own mistakes. However, it is important to have different groups with different motivations testing the application. On larger projects, a Quality Assurance group separate from the development group should test the application, particularly the ‘system components’ (data integrity, security, performance tests). A User Acceptance group should test the application primarily from a functional and user interface perspective. Having developers focus on code level testing, QA focusing on system testing and users focusing on whether the functionality meets their needs should help identify most defects before the system moves into production.

## 6.3 Test Coordinator

A critical role within an iterative development project is a Test Coordinator. This role, which might be full or part-time, needs to keep on top of what has been tested and what defects are outstanding. This role is usually a good one to advocate testing automation and the person in the role should be both a strong developer and well organized. The testing coordinator also needs to help the project manager understand the real state of the application (architects and lead developers are often too close to the application to offer accurate assessments). The testing coordinator should be responsible for reviewing and assessing all of the defects in the defect tracker used each day. The coordinator should work closely with the architect to determine if a defect really exists or if it reflects functionality that will be built in an upcoming iteration (since only partial functionality is available at the end of each iteration since the application is incrementally developed over the lifecycle of the project). The coordinator needs to also work closely with those who sourced a defect to ensure they re-test it prior to closing it. It’s not uncommon to have an individual enter a number of defects but not close them after they’ve been corrected, the coordinator needs to ensure that defects are well managed otherwise the defect log can become overwhelming. Finally, the testing coordinator needs to actively ensure that tests are regressed frequently throughout the lifecycle of the project. Many projects start short changing full regression testing in order to save time near the end of a project. While there might be some ‘balancing’ required between full regression testing and meeting project deadlines, the test coordinator should ensure that tests are at least alternated between iterations, that a full regression test is applied every three iterations and that test cases are prioritized so more complex functionality is tested frequently.

## 6.4 Testing Buffers Between Iterations

One of the primary differences between an iterative development approach and a traditional waterfall development approach is that there is less risk of a disconnect between requirements and functionality. This is because the period between gathering requirements and provisioning functionality is shorter in an iterative

<b>Iterative Development Testing Approaches</b>	Version: 1.3
<i>Managing Iterative Testing in an Agile Development Project</i>	Last Saved Date: April 9, 2002
C:\Documents and Settings\Bryan.campbell\My Documents\Business\digitalESP\Articles\Iterative Development Testing Approaches v1.4.doc	

development project. However, one of the advantages of the waterfall approach was that it provides for ‘level sets’ which allows functional components of software to be thoroughly tested before moving to the next stage of development. From the experience of the PRS project it is suggested that a blend of approaches might reflect a software development “Best Practice”. The proposed solution is to provide a defect correction buffer at the completion of every two or three iterations. This allows development to continue rapidly while also periodically ensuring that it does not get too far ahead of itself leading to greater rework in the future.

## 6.5 Automated Test Tools

Iterative development places a heavy emphasis on automation. One significant advantage of an iterative development project is that each iteration provides an opportunity to regress through all tests. This provides a tremendous opportunity to thoroughly test high risk and high complexity functionality (which should be loaded early into the iterations schedule). However, testing can rapidly become onerous if test cases are not automated. The challenge is that most GUI based testing tools are very fragile in supporting iterative development projects due to the frequent updates to the user interface. One choice is to accept the extra testing effort required during initial iterations and to manually document and execute tests though test cases which can then be automated once the application is released into production and the code base becomes stable. Alternatively, the unit tests developed can include more functional testing although this can run the risk of making these lightweight tools difficult to manage. The approach adopted for the PRS project was to only develop automated testing tools after the application had a release moved into production.

## 7. Conclusion

The PRS project provided a real world example of the challenges of integrated a testing framework into an iterative development process. Following the Rational Unified Process (RUP) and components of the eXtreme Programming (XP) methodology, the PRS project developed an iterative testing framework to support the iteration schedule developed for the project. Through this process a series of learnings were documented which included the assignment of a Test Coordinator to validate test execution and coordinate defect resolution throughout iterations, a testing buffer between iterations to allow defect correction efforts to re-synchronize with the development effort and an aggressive focus on automated testing tools to fully support regression testing. With the relative infancy of the iterative development process it is understandable that a testing approach has not been adequately developed to support this software development approach. However, the experiences gleaned from the PRS project described in this paper might provide future project managers an opportunity to better support their own iterative development projects.

<b>Iterative Development Testing Approaches</b>	Version: 1.3
<i>Managing Iterative Testing in an Agile Development Project</i>	Last Saved Date: April 9, 2002
C:\Documents and Settings\Bryan.campbell\My Documents\Business\digitalESP\Articles\Iterative Development Testing Approaches v1.4.doc	

## 8. About the Authors

### 8.1 Bryan Campbell

Bryan Campbell is an Engagement Manager with digitalESP Inc., a software development company specializing on object oriented developing using the Unified Process. Mr. Campbell has spent more than 15 years in the information technology industry across a wide range of disciplines ranging from operations to enterprise architecture to software development. Currently completing his MBA specializing in Information Technology Management, Mr. Campbell has managed a series of large scale IT projects including software development projects using iterative development techniques. Mr. Campbell has worked in a variety of industries including Telecommunications, Utilities and Insurance. Mr. Campbell has presented at several conferences and has authored several papers on iterative development.

### 8.2 Dr. Glenn Ray

Glenn Ray is an OO analyst/architect for digitalESP, Inc. Previous to joining digitalESP, he ran his own marketing and consulting company for 10 years. He is a former adjunct professor of Investment Planning at Florida State University and adjunct professor of computer science at Florida A&M University. He has a B.S. from Florida State, a M.S.E.S from Florida A&M and a Ph.D. from MIT. He is also a graduate of the College for Financial Planning.

## 9. About digitalESP Inc.

DigitalESP Inc. is a leading web services consulting company based in Raleigh, North Carolina. The company's primary focus is on custom software development for mission critical applications. DigitalESP follows uses the Unified Process and agile development methodologies to deliver object oriented software development solutions that are on-time and on-budget. For more information on digitalESP or to contact the writers of this article please phone 1-800-789-6595.

- 
- <sup>i</sup> Jacobson, Ivar, Booch, Grady and Rumbaugh, James, The Unified Software Development Process, Reading, MA: Addison-Wesley, 1999.
- <sup>ii</sup> Ambler, Scott, "Enterprise Unified Process: Enhancing the Unified Process to Meet the Real-World Needs of Your Organization", [www.ronin-intl.com/publications/unifiedProcess.PDF](http://www.ronin-intl.com/publications/unifiedProcess.PDF), 2001.
- <sup>iii</sup> Wolfgang Hesse, "Dinosaur Meets Archaeopteryx? Seven Theses on Rational's Unified Process (RUP)", [www.mathematik.uni-marburg.de/~hesse/papers/Hes\\_01b.pdf](http://www.mathematik.uni-marburg.de/~hesse/papers/Hes_01b.pdf), 2001.
- <sup>iv</sup> Beck, Kent, Extreme Programming Explained-Embrace Change, Addison-Wesley, 2000.
- <sup>v</sup> Cockburn, Alistair, Agile Software Development: Software through People, Addison Wesley Longman Inc, December 2001.
- <sup>vi</sup> Highsmith, James A III, Adaptive Software Development: A Collaborative Approach to Managing Complex Systems, Dorset House Publishing, December 1999.
- <sup>vii</sup> Schwaber, Ken and Beedle, Mike, Agile Software Development with Scrum, Prentice Hall PTR, October 2001.
- <sup>viii</sup> Scott Ambler, Agile modeling website, [www.agilemodeling.com](http://www.agilemodeling.com), 2001.
- <sup>ix</sup> Cusumano, Michael and Selby, Richard, Microsoft Secrets, New York, NY: Free Press, 1995

<b>Iterative Development Testing Approaches</b>	Version: 1.3
<i>Managing Iterative Testing in an Agile Development Project</i>	Last Saved Date: April 9, 2002
C:\Documents and Settings\Bryan.campbell\My Documents\Business\digitalESP\Articles\Iterative Development Testing Approaches v1.4.doc	

---

x Splaine, Steven et al. The Web Testing Handbook, New York: New York, McMillan Publishing, 2000

xi Karlsson, Even-Andre, Andersson, Lars-Goran and Leion, Per, Daily Build and Feature Development in Large Distributed Projects, Limerick, Ireland. ICSE Publishing, 2000

xii RTCA/DO178B, Software Considerations in Airborne Systems and Equipment Certification, TRCA, Inc. 1992