

DSDM – Practical Testing Guidelines

Table of Contents

1	INTRODUCTION.....	2
1.1	AUDIENCE	2
1.2	CONTRIBUTORS	2
2	DSDM TESTING OVERVIEW.....	3
2.1	TEST STRATEGY	3
2.2	TESTING APPROACH.....	4
2.3	TEST ROLES	6
3	TESTING PREREQUISITES AND PRODUCTS.....	8
3.1	PREREQUISITES.....	8
3.2	TEST OUTPUTS	9
3.3	INPUTS AND OUTPUTS	11
4	TECHNIQUES	16
4.1	RISK BASED TESTING	16
4.2	INSPECTIONS	17
4.3	IMPACT ANALYSIS.....	19
4.4	CONFIDENCE TESTING.....	20
5	METHODS AND TOOLS FOR USER TESTING	23
5.1	BUSINESS PROCESS APPROACH.....	23
5.2	VARIABLES APPROACH	23
5.3	TOOLS FOR USER TESTING.....	24
6	APPENDIX A GLOSSARY	25
7	APPENDIX B BIBLIOGRAPHY	33

1 Introduction

The DSDM Manual, Version 3, contains a section on testing; however practitioners continue to ask for help in this area. The purpose of this White Paper is to provide practical guidance on testing in DSDM projects.

The scope of the paper is as follows:

- To cover testing of software and business process throughout the full DSDM lifecycle from Feasibility Study through to Post-Implementation Review.
- To concentrate on testing for DSDM, not attempt to include general testing process and guidelines, which are already well documented elsewhere.
- To exclude procedures for Risk Analysis and Risk Management, but to refer to other DSDM documents on Risk.

Throughout this white paper, the term 'Test Strategy' has been used to denote a product output from the Business Study stage. This product shows which types of testing will be done during which stages of the life cycle for a particular project/increment. Within DSDM Manual Version 3, this product is called an 'Outline Testing Plan'. The DSDM manual referred to throughout this White Paper is version 3.

1.1 Audience

The white paper is aimed at all members of the IT and business communities involved in the development, testing and implementation of DSDM projects. There is particular guidance for users who have not been previously directly involved in planning and execution of software tests.

1.2 Contributors

This White Paper was put together by a Task Group consisting of:

Brenda Hubbard	Xansa (chair)
David Fox	Clerical Medical Investment Group Ltd.
Greg Peachey	PQR Plus Ltd.
John Stanford	Phoenix Database Technologies

And reviewed by the above, plus:

Graeme Henry	British Airways plc
Jan Heslop	Xansa
Asmund Maehle	Computas (Norway)

2 DSDM Testing Overview

The aim of this section is to give some practical guidance by describing a possible approach to testing on a DSDM project, and giving advice on how this approach could be varied under specific circumstances. The reader can then use this as the basis for determining their strategy for testing a particular development.

The section begins with an overview of the concept and importance of producing a Test Strategy (Outline Testing Plan) for a project or release. It is within this Strategy that the overall shape, purpose and responsibilities for testing will be defined.

2.1 Test Strategy

It is recommended that the approach to testing be considered in both Feasibility and Business Study stages. The following questions need to be addressed:

- When will we do testing and which types of testing will be done?
- What will be the objectives of the testing that we do?
- Who will be responsible for the testing?
- What deliverables will be produced? Which of these will be re-usable, and how will we ensure this?

The Feasibility Study contains a Testing Assessment, which highlights the risk areas, i.e. those which may require extra testing or have special testing needs. These may include such things as high volume transaction areas which need to be tested for performance, or restricted areas of functionality where testing of security is important.

The Business Study includes the production of the Test Strategy for the project/increment, as part of the Outline Prototyping Plan. This maps the types of testing against the project lifecycle, indicating if and where each type of testing will be done. If it is decided not to perform certain types of testing, then the risk of these deliberate omissions will be described in business terms, so that the business is fully aware of the implications.

A contents list for a Test Strategy may include:

- Testing Overview
 - Cross-reference of testing types to DSDM stages
 - Test Data Approach - General
 - Testing Aids
- For each stage of testing:
 - Objectives
 - Environmental Requirements

- Commitments from other teams and third parties
- Strategy for Test Data
- Acceptance Procedures, Entry and Exit Criteria
- Roles and Responsibilities
- Deliverables
- Security and Control (if applicable)
 - Security Considerations
 - Quality Assurance and Standards
 - Configuration Management Standards and Tools
- Outline Schedule For Testing

Note: it is recommended that the exit criteria should contain measurable overall targets for test coverage and results so that it can be determined whether or not objectives have been reached and how close the project is to meeting them.

2.2 Testing Approach

This section shows a possible approach to testing for a DSDM project, including most of the major types of testing applicable to software development. Definitions of the testing types can be found in Appendix A, Glossary.

This approach is outlined here as an example only; the approach to testing for a particular development should be tailored to the requirements and risks of that development and documented in the Test Strategy. This example is intended to give a starting point for that exercise if required.

2.2.1 Example approach to DSDM testing.

Table 1 is based upon Table 21.1 from the DSDM Manual, and has been expanded to include a wider range of testing types.

Note: Regression Testing should be planned for throughout the lifecycle wherever changes may impact on non-changed software, e.g. if multiple vertical prototypes are being built which interface/share data, then we need to consider Regression Test of all prototypes when new/amended ones are built. For guidance in selection of Regression Tests, see Section 4.3, Impact Analysis.

Feasibility Study	Business Study	Functional Model Iteration	Design and Build Iteration	Implementation
<input type="checkbox"/> Test the Feasibility Prototype (optional)	<input type="checkbox"/> Produce Test Strategy	<input type="checkbox"/> Unit Test <input type="checkbox"/> Link Test <input type="checkbox"/> Business Acceptance Test – against functional	<input type="checkbox"/> Unit Test <input type="checkbox"/> Link Test <input type="checkbox"/> System Test against functional and non-functional	<input type="checkbox"/> Operational Acceptance Test against technical infrastructure <input type="checkbox"/> Portability

		functional requirements	functional requirements	Test
		<input type="checkbox"/> Usability Test <input type="checkbox"/> Regression Test of unchanged parts of prototype <input type="checkbox"/> Confidence Test	<input type="checkbox"/> Integration Test, both internal and external <input type="checkbox"/> Performance Test <input type="checkbox"/> Volume Test <input type="checkbox"/> Multi-user Test <input type="checkbox"/> Break Test <input type="checkbox"/> Regression Test of unchanged parts of prototype <input type="checkbox"/> Business Acceptance Test against business processes <input type="checkbox"/> Confidence Test	<input type="checkbox"/> Installation Test <input type="checkbox"/> Recovery Test <input type="checkbox"/> Regression Test of unchanged parts of application <input type="checkbox"/> Confidence Test

Table 1: Testing activity in DSDM phases

2.2.2 Possible variations

Testing may vary from this pattern for many reasons, hence the need for a Test Strategy demonstrating 'fit for purpose' testing.

Some alternative scenarios may be:

- *Large, complex programme of which this development is one part, some other parts may be developed using traditional methods.*
In this scenario, the above approach may be followed by a separate testing stage, including Integration Testing of the whole programme and Business Acceptance Testing across end-to-end business processes.
- *Environment for Design and Build Iteration is unsuitable for high volume test data.*
In this scenario, Volume and Performance Testing may be done during the Implementation stage.
- *No tools are available to simulate multiple simultaneous access, so multi-user test is not done; however resources are made available to monitor the application post-implementation and provide fast and effective support when required.*

- *Performance of application is critical*, so some Performance Testing and/or modelling is started in the Functional Model iteration.

There could be many of these scenarios. Each project is different, hence the need to develop and maintain a Test Strategy.

2.3 Test Roles

This section describes the following roles that are specific to DSDM projects. General testing roles such as Test Analyst and Tester are not included here.

2.3.1 Testing Manager

The DSDM Manual refers to the role of Testing Manager who will be responsible for ensuring the Test Strategy is produced, ensuring the environments are set up correctly and generally co-ordinating the testing activity. This is a specialist role and, as such, one that is involved in the project on an *ad hoc* basis.

2.3.2 Test Co-ordinator

Given the particular nature of testing in DSDM (i.e. that it is integrated throughout the lifecycle), and that there is frequent rapid change in what is being produced, DSDM teams should consider appointing one of the team as a Test Co-ordinator. The Test Co-ordinator would be a core team member who would be able to perform other roles (such as development) when not required to act as a Test Co-ordinator.

The responsibilities of the Test Co-ordinator would vary depending on the size and nature of the project but would include either doing or co-ordinating the following:

- producing the Test Strategy
- designing, specifying and monitoring the production of the test environment
- assisting developers and ambassador/adviser users in producing Test Plans
- co-ordinating the user resources for testing
- reporting on test progress
- resolving/escalating any issues or problems encountered when testing
- controlling environments (e.g. back-ups of databases)
- maintaining a log of which levels of testing have been performed on each element of code
- configuration management of test assets (in liaison with the Configuration manager/librarian)
- providing support on testing best practice (in conjunction with the Testing Manager)

- maintaining the impact analysis matrix (see section 4.3).

The focus of DSDM testing should always be in line with the DSDM Testing Principles and the over-riding priority of the co-ordinator is to ensure that these are understood and adhered to by the rest of the team, in order to maximise the testing benefit in the time available.

3 Testing Prerequisites and Products

3.1 Prerequisites

3.1.1 Timebox referencing

The Test Strategy defines the key objectives of testing within each stage and should be the point of reference for all aspects of testing throughout each increment. Each timebox holds a discrete set of Test Requirements and it is suggested that a checklist of tasks be extracted from the Test Strategy and reviewed prior to each timebox.

3.1.2 Personnel

The development team should include a Test Co-ordinator who will have overall responsibility for the facilitation and co-ordination of testing activity on the project. For more detail of the responsibilities of a Test Co-ordinator, see Section 2.3.2. Having the right person in this role can be key to the success of the project.

The Testing Manager (see Section 2.3.1) has overall responsibility for testing across multiple projects which may be a mixture of DSDM and traditional.

Also key to the successful completion of a DSDM project is the understanding and acceptance of the testing responsibilities on each team member.

3.1.3 Standards

DSDM defines a rapid approach for developing products and the team must be empowered to make changes speedily and effectively. An effective Test Strategy is one that can be defined at the outset and refined as the project progresses. Therefore, DSDM must allow for documentation of relevant Test Plans within each timebox and a speedy method of refining and agreeing changes to these plans.

An appropriate set of standards that encourages re-use and allows team members to become effective quickly will have a positive impact on productivity. It would be useful for a set of agreed or standard test documents to be produced prior to each timebox so that the effort required to produce Test Plans and undertake the testing work can be made as efficient as possible.

3.1.4 Supporting procedures

DSDM allows for a team to be empowered to make and implement changes without recourse to lengthy debate from persons outside the project team. However, all change must be handled in a controlled manner.

For testing, as for the rest of the lifecycle, the following procedures are essential:

- Change Control
- Incident Management

- Risk Management
- Configuration Management
- Testing Procedures - to help ensure repeatable tests.

3.1.5 Deliverables needed for testing

The products in Table 2 should be taken into account when planning testing at any stage of the lifecycle. A more detailed product breakdown can be found in Table 3.

Deliverable	Main Purpose in Testing
Prioritised Requirements List	Forms the main basis for testing in DSDM, as it is against this list that we can validate the application.
Style Guide/Interface Standards	Where these exist, they form the basis for the look and feel part of Usability Testing. Other measures for success of Usability Testing may come from productivity measures and/or ease of learning/ time to become proficient.
System Architecture Definition	This is needed in order to design, build and manage the test environments.
Implementation Strategy	Useful input to Test Strategy, even if not detailed at this stage
Non-Functional Requirements List	Needed as basis for testing; input to Test Strategy and to Test Plans where relevant.
Risk Analysis Report	See Section on Risk Based Testing

Table 2: Inputs to test planning

3.2 Test Outputs

Standard testing deliverables take on a particular significance in the context of DSDM.

This section lists deliverables from the testing activities within a DSDM project. The types of test listed are not prescriptive. The stages of testing as defined within the Test Strategy will determine the number and type of plans to be produced.

3.2.1 Testing Assessment

The Testing Assessment is produced during Feasibility stage. For more details refer to Appendix B of the DSDM Manual.

3.2.2 Test Strategy

In broad terms the Test Strategy covers the framework for testing. It is useful to have key documents identified and details of how each is applied although each project may lay emphasis on different aspects of the system. For example, it may be essential for the usability of the system to be at a higher priority than the performance of the system – in which case specific time is set aside to work on and confirm the usability of the system. These points will have been uncovered by the early risk assessment.

The Test Strategy will allow all members to understand and agree the testing process and for the Project Manager to plan the testing effectively.

3.2.3 Test Plans

Test Plans contain the scope of conditions to be tested and describe how the testing will be done, via test cases, environment and data. They are essential as a basis for test execution and monitoring progress of tests, coverage and gaps. (Test Plan often called Test Specification).

Note: for DSDM it is still necessary to plan testing, and a Test Plan is necessary to ensure repeatability of tests; however, they are not generally as detailed as traditional Test Plans.

Some of the Test Plans that may be produced on a DSDM project follow.

Link Test Plan

Link Testing should be performed during each iteration within the timebox allowing the team to complete areas of functionality before moving on to the next iteration. The Link Test Plan should reflect this as its primary objective. Link Testing is preceded by Unit Testing to ensure accordance with the functional objectives and the Link Testing will allow an 'independent' check on the deliverable. A member of the team other than the developer himself should plan and perform Link Testing.

Business Test Plan

Key to DSDM is the need to meet the overall business objectives of the project and the business processes should have been documented during the business study. As a result of the business study the Business Users can translate these processes into a Business Test Plan to provide the foundation for acceptance.

Within the context of the Business Test Plan there will be Usability Testing taking place during the Functional Model Iteration and Business Process Testing taking place during Design and Build.

System and Integration Test Plans

As the project progresses each increment will be completed by successfully undertaking System and Integration Testing prior to implementation in the client environment. System Testing may involve combining the previous Link Test Plans from the current increment to ensure a coherent product is produced and will also ensure that regression testing on other relevant areas of the application is performed. Whilst testing should relate only to the functionality developed within the current increment the Test Strategy may dictate that some testing on other system areas may be necessary.

Regression Test Plan

Testing should be constrained by the needs of the timebox but there may be circumstances where testing needs to ensure that no impact has been incurred on other areas of the system.

3.2.4 Test Script

The Test Script tells the tester how to run the test. Although needed for repeatability, DSDM Test Scripts should be kept brief and are often a simple checklist of actions and expected results.

3.2.5 Test Report

The Test Report summarises what actually happened during the test. In DSDM it is particularly relevant as part of the confirmation that a software deliverable is suitable for business use.

3.3 Inputs and Outputs

Table 3 summarises the testing-related inputs and outputs by DSDM phase. Note: the inputs and outputs listed in the table are typical of a DSDM project, but are not mandatory.

Stage	Input	Action	Output	Owner
Feasibility Study	Suitability Filter	Risk assessment and identification of key risk areas	Risk documentation concerning risk based Test Strategy	Project Manager
	Definition of Business Problem	Identify Acceptance Criteria and means of testing to demonstrate successful outcome	Testing Assessment	Project Manager
Business Study	Testing Assessment	Determine / confirm Test Strategy, using Prioritised Requirements List	Test Strategy	Visionary / Ambassador User
	Business processes	Determine Business Test Plan for Usability and Business Process Testing	Business Test Plan	Ambassador User
Functional Model Iteration	Test Strategy	Define Unit Testing criteria	Unit Test Plan	Developer
	Test Strategy Business Test Plan Style Guide/Interface Standards	Define Link Testing criteria Plan in detail Usability and Regression Testing	Link Test Plan Usability Test Plan (may be part of Business Test Plan) Regression Test Plan	Test Co-ordinator / Team Leader
	Unit Test Plan	Each iteration should contain suitable Unit Testing by the developer	Test Report	Developer

Stage	Input	Action	Output	Owner
	Link Test Plan	Each iteration should contain suitable Link Testing by team member other than the developer	Test Report	Test Co-ordinator / Team Leader
	First prototype Prioritised Requirements List	Business acceptance of : Look and feel (usability); Functional requirements;	Acceptance of functional and usability aspects	Test Co-ordinator / Ambassador User
	Regression Test Plan	Confirm that unchanged aspects remain unaffected	Acceptance of Regression Test findings	Test Co-ordinator / Team Leader
	Testing standards Prioritised Requirements List	Determine criteria for Volume Testing; Performance Testing; Multi-user Testing; Break Testing, System and Integration Testing	Plans for Volume Testing, Performance Testing, Multi-user Testing; Break Testing, System and Integration Testing	Project Manager Test Co-ordinator
Design and Build Iteration	Unit Test Plan	Each iteration should contain suitable Unit Testing by the developer	Test Report	Developer
	Link Test Plan	Each iteration should contain suitable Link Testing by team member other than the developer	Test Report	Test Co-ordinator / Team Leader

Stage	Input	Action	Output	Owner
	Plans for Volume Testing, Performance Testing, Multi-user Testing; Break Testing, System and Integration Testing	Execute and report on tests as defined in Test Strategy	Test Report(s)	Test Co-ordinator / Ambassador User
Implementation	Operational Acceptance criteria, Test Strategy, Non-functional requirements	Plan Operational Acceptance Testing, Portability, Installation, Recovery, Regression and Confidence Tests, as appropriate according to Testing Strategy.	Plans for OAT, Portability, Installation, Recovery, Regression and Confidence Tests	Ambassador User / Visionary
	Plans for OAT, Portability, Installation, Recovery, Regression and Confidence Tests	Execute and report on tests as defined in Test Strategy	Test Report(s)	Test Co-ordinator, Project Manager

Table 3: Test Stages Inputs and Outputs

4 Techniques

4.1 Risk Based Testing

With the majority of IT projects, testing can be both resource hungry and expensive. To overcome this, efficient use of time available can be made through Risk Based Testing (RBT).

The premise behind RBT is to base testing on risk as most IT projects are subjected to constraints of time, resource and money. To maintain the overall quality of the application the test coverage must ensure that the critical business processes are covered.

In essence, RBT is covered by the following steps:

- **Identify the risk:** Collect all information about the parts of the system most likely to be exposed to errors. This stage will require the input from all members of the project team to help identify risk areas. This can be achieved via a facilitated workshop.
- **Impact:** Identify the effects of potential errors and the probability of such errors occurring.
- **Plan / strategy:** Testing is carried out on areas of the system identified as a risk, i.e. the effort is focused on the risk areas.
- **Reduction of risk:** Measures should be taken to reduce the probability of errors occurring.

Note: Unit Testing is completed as normal, whereas an RBT approach can be applied to all subsequent stages of testing.

4.1.1 Identify the risk

Time is of the essence in DSDM projects and whilst testing and quality should not be compromised it is essential that best use is made of the time available; therefore, some form of Risk Based Testing may be appropriate. Risk can be assessed from the moment that the Suitability Filter has been checked. Specific aspects of the filter may highlight key areas of high risk to the project or may indicate certain filter requirements that have not been met – in themselves, these are now risk points.

Note that while risk is assessed at the earliest possible moment in the project life-cycle it will continue to be assessed and managed throughout the project.

Risk assessment, as the basis for a Test Strategy and Test plans, is performed by all members of the project team. This will ensure that a balanced view of the impact of each area of functionality on the business can be obtained.

The project is broken down into relevant functional areas and the risk to the development team and the business is assessed against certain arbitrary factors; e.g. complexity of processing, usability, knowledge and experience of the developer, consequence of failure, etc. Once this grading has been undertaken it

will then be possible to identify those areas of the system likely to be at high risk and to ensure that testing resources are concentrated at these points.

4.1.2 Assess impact

For each function in the application, the impact of failure is assessed. Example: failure of customer facing software has higher impact than non-production of an internal report (unless, perhaps, the report is produced for the Board).

This impact, together with the probability of risk, can be used as input to Test Planning.

4.1.3 Plan/Strategy for RBT

Based on the outcome of the risk identification and impact assessment, plans are then developed with test cases prioritised according to risk and impact. In this way, should the timebox expire before testing is complete, the higher risks will have been addressed first.

4.1.4 Reduction of risk

In addition to prioritisation of tests, other steps will be taken to manage and reduce risk. These may include introduction of inspection activities on high-risk deliverables/tasks.

Note: for more information on Risk Management, refer to the DSDM White Paper on Risk Management.

4.2 Inspections

It is widely accepted that static testing, in the form of inspections, is a very efficient method of identifying defects, with the additional benefit of improving team learning and speeding up knowledge transfer. This paper will not elaborate on those benefits, nor will it go into any details on the various types of techniques available (e.g. reviews, walkthroughs, formal (Fagan) inspections). To ensure common understanding the term “informal review” will be used to mean a relatively spontaneous desk-check of a product by a peer. A “formal review” will be used to mean a review which has been planned several days in advance, involves all relevant stakeholders (both in and outside of the team), has clearly assigned roles (e.g. moderator, scribe), and for which each participant prepares in advance. Obviously there can be types of review in between the two extremes.

Formal reviews tend to identify more defects but are more costly in terms of time and effort. Therefore the DSDM team will need to find the right balance for the product being inspected depending on the associated risk.

4.2.1 Difficulties and benefits of using inspections in DSDM

There are several difficulties associated with using inspections in DSDM that would not be encountered in a waterfall lifecycle.

Iterative development

Because of the iterative nature of the development and the fact that products are continuously evolving it is difficult to know when a product is ready for inspection. Inspect too early and the exercise may be wasted as the product may be subject to further dramatic change. Inspect too late and the defects may have already cascaded down to subsequent products and the benefits will have been lost. Therefore a product should be considered ready for inspection when it is “finished” for the time being. Note that finished does not necessarily mean complete as more work on that product may be planned, but it does mean that it is in a stable state.

One of the DSDM principles is that changes are reversible. Clearly, therefore, there will be points in the development of each product where it is re-baselined. These should be the points where inspections should be considered.

Time constraints

Because of the tight timescales typical of DSDM there will often be a tendency to treat inspections as unaffordable luxuries rather than an essential tool.

However, the following features of DSDM make inspections easier to arrange.

- **Collocation.** Because the right people are (ideally) located close to each other, and probably already have a dedicated meeting area it should be much easier to plan and organise inspections than in a typical waterfall environment.
- **Timeboxing.** Despite the fact that time constraints may “squeeze out” inspections they can also have the opposite effect. If a timebox is due to finish on a certain day, and as time, not functionality, is sacrosanct in DSDM it should be relatively easy to identify when inspections should occur, and arrange them in advance. The scenario should never arise in DSDM where an inspection has to be postponed because “the product is not quite ready yet”.

4.2.2 Principles of inspections

As every project is different it is not possible to be prescriptive about using inspections but the following gives some guidelines. The guidelines focus on three of the six testing principles in the DSDM manual and show how each applies equally well to static as to dynamic testing.

Benefit-directed

It may be of value to identify which requirements on the Prioritised Requirements List (PRL) should be tested dynamically, statically or both. For functionality that is highly visible to the users then static testing may be an unnecessary overhead, as any defects will become apparent very quickly. However, there may be some requirements that are exceptionally difficult to test dynamically (because they require a certain set of complex conditions to be true, or because the data needs to be in a certain state). If the functionality does not provide high business

benefit, and if time is short, then static testing may be an ideal solution for validating the code.

Independent

Often in DSDM developers are working on further items on the PRL while users are busy testing what has already been coded. Another option is for some of the developers to be statically testing each other's code whilst the users are doing their testing. A short review period will be necessary at the end of each testing session to consolidate the defects and plan for any correction and re-testing that may be required. It is recommended that each testing session is kept short (probably about half a day) to keep the process manageable and to avoid too much duplication.

Error-centric

It is very easy with static testing for the focus to be on less important details such as style, presentation, and technique. The focus must always be on identifying and correcting errors. A good moderator will ensure that this focus is maintained but the danger remains for informal inspections where there may not be formal allocation of roles.

There is a risk that defects will remain undetected in some products for longer than others, during which time the effect may spread to subsequent products. For example a defect in the System Architecture Definition (SAD) may only become apparent during implementation. The additional danger is that there may be small localised manifestations of the defect in subsequent products without the root cause ever being spotted. Attention should therefore be focused on these types of products (examples of other products are Feasibility Report, Business Area Definition, Implementation Strategy and User Documentation). Consideration should therefore be given to using more formal inspections on these products.

4.3 Impact Analysis

Current software testing approaches take account of the fact that, despite modern modular approaches to development, functional areas of a system (or application) remain highly inter-dependent. That is, even where an apparently isolated change is made to a system, there is a high probability that undesirable knock-on effects will occur in supposedly unrelated areas. This gives rise to a risk area particularly where incremental development methods are being applied. The risks are further pronounced in a rapid application development environment where neither the time nor the resources are available to re-test a system comprehensively after every system modification or enhancement. DSDM developments are thus doubly vulnerable.

As DSDM testing resources tend to be limited, it is vital that they are expended in the most effective manner, by being very selective regarding those elements of a test suite that are re-run in order to maintain ongoing confidence in system

integrity. It is therefore recommended that an impact analysis matrix be maintained as illustrated below:

Feature s	A	B	C	D	E	F	G
A							
B							
C							
D							
E							
F							
G							

Table 4 - Impact Analysis Matrix

In Table 4, a full set of system features is listed both in column and row headings. A feature may, for example, be an item in the Prioritised Requirements List. A black cell indicates that whenever the feature listed on the corresponding row is modified, it is mandatory that the feature in the corresponding column is re-tested. A grey cell indicates that whenever the feature listed on the corresponding row is modified, it is recommended that the feature in the corresponding column is re-tested. Any further re-testing is optional.

The above practice may be viewed as an extension to the principle of Risk Based Testing.

4.4 Confidence Testing

As described in Section 4.3, the value of limited testing may be maximised by identifying the interdependence between system features. Testing may be further targeted by categorising the depth of test cases. An example categorisation might be:

- *Development tests* run by the developers, armed with a knowledge of the internal workings of a system;
- *Regression Tests* designed to ensure that parts of the system which have not been directly changed, continue to operate successfully after the change has been made.
- *Confidence Tests*: cases that intuitively exercise each system feature in the most fundamental manner, i.e. Breadth Tests.

Different combinations of these categories of testing may be applied depending upon the nature of the changes being made, e.g.:

Confidence									
				Regression					
				Development					

Figure 1: Test Cases (hatched) to be Re-applied Following Local Modifications

Confidence test across whole system, detailed regression and development testing in the areas of modification.

Confidence									
Regression									
Development									

Figure 2: Test Cases to be Re-applied Following a Minor Software Release

Confidence test the whole system, e.g. does it still run?

Confidence
Regression
Development

Figure 3: Test Cases to be Re-applied Following a Major Software Release

Confidence and regression test the whole system, e.g. does it still run and function/perform correctly?

Given relaxed timescales, all test cases would be automated and re-run after the development of each increment and after each system-wide modification. Where schedules are tight, however, it is recommended that, as a minimum, the Confidence Test cases are automated (e.g. using record/playback tools), maintained and reapplied after each Design and Build Iteration. The project will then release its final deliverable without unexpected, extensive, last minute re-working, but with justified confidence in system integrity.

5 Methods and Tools for User Testing

The purpose of this section is not to re-iterate any of the content of the DSDM Manual, to explain existing testing theory or to list existing tools on the market. The aim is, rather, to indicate how current theory may be adapted and existing tools may be applied within the context of DSDM.

In DSDM projects, the proportion of testing conducted by non-technical users is greater than in waterfall projects. Classically, testing theory has rarely been grasped (and even less frequently applied) by software developers working to challenging deadlines. Non-technical users are less likely to be familiar with the theory but under even greater pressure to minimise testing effort in RAD projects. The natural trend is therefore a return to ad-hoc testing practices. Experience has shown, however, that unless testing is systematic, large areas of software tend to remain totally unexercised and there is no way to plan/gauge the comprehensiveness of testing or to identify areas of uncertainty.

Classical testing methods and tools, therefore, should not simply be discarded, as they do have a very sound basis. They need, however, to be adapted so that they are readily assimilated by users and workable within real-world deadlines and cost constraints. The essential problem is that classical testing methods focus on somewhat esoteric technical types of testing, such as Black Box — Equivalence Partitioning and White Box — Linear Code Sequence and Jumps (see Glossary for further information). When being applied in DSDM projects, testing theory needs to be re-focused. Two approaches are outlined here.

5.1 Business Process Approach

Test cases for Business Acceptance Testing can be based upon analysis of business processes. This is done using workshops involving both users and technical staff. The business processes are modelled down to activity level, and roles, departments and applications are then mapped against activities. Test Plans and Scripts are then created for each process.

An additional benefit of this approach is that the business process scenarios modelled as input to testing can also be used for designing training material.

If this or a similar approach is to be followed, it should be defined and adopted during the Business Study stage.

5.2 Variables Approach

Features on a Prioritised Requirements List may be analysed in terms of associated run-time variables, for instance valid and invalid values of a password, the menu item selected, the balance in a financial account or the number of attempts at performing a given operation. Users may then be presented with guidance as to which types of test case should be applied to which types of variable. The comprehensiveness of a test suite can then be readily assessed in terms of the coverage of variables and required test cases,

and gaps in coverage may be used to identify residual risk areas or incrementally to improve the quality of primitive, but validated prototypes.

5.3 Tools For User Testing

Unfortunately, there is as yet a shortage of tools to assist users without technical support in the design, running or measurement of tests. For technical users, there are tools that, for example, “instrument code” (i.e. identify unexercised program logic and measure coverage). Even tools that record and playback scripts at the user interface often require scripts so-generated to be “massaged” so that they refer to controls and objects, rather than to screen areas — otherwise those scripts are subject to being rendered obsolete by trivial screen layout changes. Such updates to scripts often require knowledge of a programming type language or similar technical aptitude.

Some capture/replay tool vendors are now taking steps to build ‘user-friendly’ front-ends to their testing tools, so these should become more relevant to user testing.

Data manipulation utilities are often straightforward and easy to use with minimal training, e.g. SQL.

Test management tools that contain lists of requirements, functions, test conditions and cases, and monitor these against actual test execution can be used effectively by users in some cases.

6 Appendix A Glossary

Term	Description	Comments
Black Box Testing	A strategy which is concerned, not with the internal software structure, but with finding circumstances in which the software does not behave according to its specification or reasonable expectation	<p>It is applicable to both low and high order testing. It supplements the White Box approach at unit level, and replaces it at higher levels.</p> <p>It is capable of detecting incorrect or missing code (i.e. potential scenarios overlooked by developers).</p> <p>There are three standards, whose applicability depends upon the nature of software under test: Equivalence Partitioning, Boundary Value Analysis, Cause/Effect Analysis</p>
Boundary Value Analysis	This standard requires that one must write sufficient test cases to explore those situations directly on, above and beneath the edge of input equivalence classes.	This is applicable where it is possible to identify error-prone borderline situations.
Break Testing	To test that the system handles in a controlled manner, breakdown and disconnections in the technical infrastructure, e.g. POS till becomes disconnected from server.	

Term	Description	Comments
Business Acceptance Testing	<p>Test(s) to ensure that the system meets the acceptance criteria of the business, and fits in with the business processes it is designed to support.</p> <p>To check that the system performs as expected from a user perspective.</p> <p>To check that the changes permit efficient and effective use of the applications. To check that wording changes make sense to the users.</p> <p>To enable users to test their own procedures and to demonstrate that they are able and ready to use the new software.</p>	Also known as User Acceptance Test (UAT) and Customer Acceptance Testing (CAT).
Cause/Effect Analysis	This standard requires that one must write sufficient test cases to explore the entire set of output conditions caused by a combination of input conditions, or equivalence classes of input conditions.	This approach is necessary where required behaviour depends upon a number of related factors or a sequence of events.
Confidence Testing	<p>To test with Breadth Testing that the system is intact and can operate in the current environment.</p> <p>The aim is to give confidence to proceed in more detail and with greater levels of effort.</p>	Example: when doing Acceptance Testing of software developed by a third party, it is recommended that Confidence Testing of the delivery from the supplier is carried out before full Acceptance Testing begins.

Term	Description	Comments
Decision Coverage	<p>The percentage of decision outcomes exercised by a test suite.</p> <p>For one hundred percent coverage, one must write sufficient test cases, such that each branch takes on all possible outcomes at least once, and invoke each point of entry to the program or subroutine at least once.</p>	This provides a sound level of confidence in the correctness of simple software — where 'simple' means that the internal decisions are largely independent of each other.
Equivalence Partitioning	<p>Involves dividing situations that the software might encounter into groups (equivalence classes) where similar behaviour is required.</p> <p>(Equivalence class = a portion of the component's input or output domains for which the component's behaviour is assumed to be the same from the component's specification)</p>	This standard requires that one must write a minimal set of test cases that explore all valid and invalid equivalence classes of each input condition in the specification.
Functional Testing	Test(s) to check that the system fulfils its functional requirements.	Test all main functions within the system, and check interfaces into and out of system, (usually done by using utilities to generate input and by sight checking output data).
Increment	A complete instance of the DSDM lifecycle that delivers a release of fit-for-purpose software to the business.	A typical DSDM Project will have several increments, each building upon the previous increment. It is not an iteration within the lifecycle

Term	Description	Comments
Installation Testing	<p>To check the quality of the database and installation scripts and to ensure that Installation will work.</p> <p>To check timings needed for installation, for migration of data for instance.</p>	
Integration Testing	<p>Checks that the system interfaces correctly with other internal systems e.g. the interfaces between a stock system and a purchase ledger.</p> <p>Ensures that the system interfaces correctly, at both control and data levels, with external systems, e.g. BACS.</p> <p>Ensures that the system functions correctly, at both control and data levels, within the overall business/technical environment in which it will operate, e.g. a system for a telecommunications company will be tested in the telecomms network.</p>	
LCSAJ	Linear code sequence and jump.	<p>Usually identified by source code line numbers –</p> <p>Start line of linear sequence of executable statements,</p> <p>End line of linear sequence,</p> <p>Target line to which control is passed at the end of the linear sequence.</p>

Term	Description	Comments
LCSAJ Coverage	<p>The percentage of LCSAJs of a component which are exercised by a test case suite.</p> <p>For one hundred percent coverage, one must write enough test cases for each LCSAJ to be executed at least once.</p>	<p>This level is suitable where there is interdependence between internal decisions. It is a time-consuming exercise, though, and can normally only be applied to safety-critical software or to small, key elements of systems or applications.</p>
Link Testing	<p>Test(s) on those functions that can be performed by two or more modules together and not by one alone.</p>	<p>Ensure that modules which have been individually Unit Tested can be linked together, that data and control can be passed correctly between them and that they continue to function effectively.</p> <p>Also known as Low Level Integration Testing or Integration Testing In The Small</p>
Multi-user Testing	<p>To ensure that the application continues to operate successfully and meets the performance requirements when operating with high levels of multiple access.</p> <p>To assess the impact of the system on the overall performance of other systems running on the same hardware.</p> <p>Includes multi-user concurrent access to the system, to prove locking, system processes and data integrity.</p>	

Term	Description	Comments
Operational Acceptance Test	<p>Ensures that the system meets the acceptance criteria of the IT Operations Department, DBAs and maintenance functions.</p> <p>Checks that the system performs as expected from an operational perspective.</p> <p>Enables operations to test their own procedures and to demonstrate that they are able and ready to use the new software.</p>	
Performance Testing	<p>To ensure that time critical or high volume parts of the system meet the performance requirements of related service level agreements when operating in an environment similar to production.</p>	
Portability Testing	<p>To ensure that the results obtained after a change in machine environment are the same as those obtained under the old environment, e.g. transfer from development to production environment.</p>	<p>In client server environments there may be many combinations of hardware and software across client workstations, network connections and servers. Portability Testing would ensure that the application works across all combinations.</p> <p>Also known as Configuration Testing.</p>

Term	Description	Comments
Recovery Testing	<p>To evaluate the system features for processing interruptions and for returning to specific points in the system. Checkpoints, back-ups, restores and restarts are tested here.</p> <p>Special types of Recovery Tests are: <i>Immediate Full Recovery Test</i>, to test backout of system immediately after transfer to live environment. <i>Delayed Full Recovery Test</i>, To test backout of system after users have been allowed access to the system.</p>	
Regression Testing	<p>Test(s) to ensure that parts of the application, which have not been directly changed, continue to operate successfully after system/ database changes have been made.</p>	<p>Acts as a check on areas potentially not affected by the system.</p> <p>Based upon before and after tests to prove no unexpected change.</p>
Statement Coverage	<p>The percentage of executable statements in a component that has been exercised in a suite of test cases.</p> <p>For one hundred percent coverage, one must write enough test cases for each executable statement to be executed at least once.</p>	<p>This can only provide the most basic level of confidence in the integrity of a piece of software, because the choice of test scenarios is sparse and incidental. It is often possible to devise one or two 'artificial' test cases that force a program through virtually all statements.</p>

Term	Description	Comments
Unit Testing	Test(s) on an individual software component to confirm that it meets its specification in isolation from other system components, and to exercise each path through the code.	Also known as Component Testing.
Usability Testing	To test the usability aspects of the system.	May be done in a formal usability lab. Should be done at various points throughout development, not left until User Acceptance Testing. Sometimes called Ergonomics Testing.
Volume Testing	Test(s) to ensure that the system continues to operate successfully and meets the performance requirements when operating with the large volumes of data expected in the live environment.	
White Box Testing	A technique where test data is derived from an examination of a program's structure	This approach is best suited to low-order testing and is capable of detecting incorrect or redundant portions of software. A better name would be "Glass Box Testing". There are three standards of White Box Testing: Statement Coverage, Decision Coverage , LCSAJ Coverage.

7 Appendix B Bibliography

- a) Risk Based Testing Avenir (UK) Ltd www.avenir.co.uk
- b) Gilb T., and Graham D., Software Inspection, ISBN 0-201-63181-4, Addison-Wesley