

# Managing Object-Oriented Projects: DSDM and XP

## Chapter 21B

## In this Lecture you will Learn:

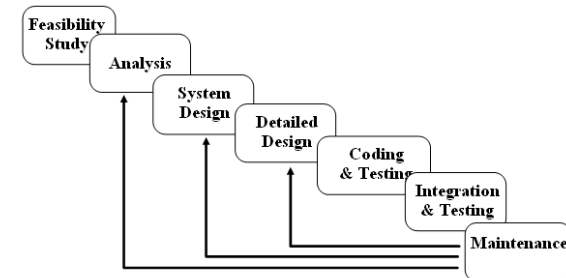
- How to manage iterative projects
- The main features of Dynamic Systems Development Method (DSDM) and eXtreme Programming (XP)

## Dynamic Systems Development Method

- DSDM is a management and control framework for Rapid Application Development (RAD)
- RAD aims to build a working system rapidly
- Prototyping aims to build elements of a system (a partial system) quickly
- Similar development environments can be used for both
- A prototype may incrementally become a working system

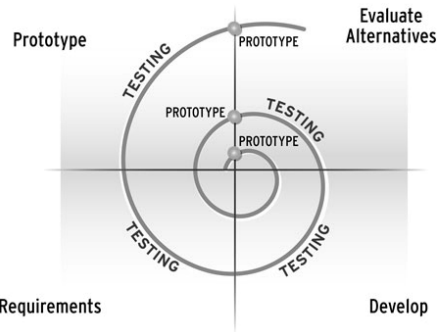
## Why not Waterfall?

- Traditional waterfall approach has various deficiencies, particularly the time taken to deliver a working system and the inflexibility of the approach to requirements change.



## Why not Iterative?

- Iterative approaches to development can also be problematic.
- It is difficult to cease the iterations when they become unproductive.



## Origins of DSDM

- RAD popular in early 1990s
  - ◆ No commonly accepted approach
- DSDM Consortium set up in 1994
  - ◆ DSDM defines the structure and controls for a RAD project but does not specify the methodology
  - ◆ Has been used with SSADM and OO

## DSDM Project Management

- DSDM takes a fundamentally different perspective on project control
- Conventionally requirements are viewed as fixed and then matched by resources
- DSDM fixes resources for the project, fixes the time available and then sets out to deliver only what can be achieved within these constraints

## 9 Underlying Principles for DSDM

1. Active user involvement is imperative. In DSDM they are part of the team and known as 'Ambassador' users
2. DSDM teams are empowered to make decisions including refining or changing requirements without the direct involvement of higher management.
3. The focus is on frequent product delivery. A team delivers product(s) in within a time box (2 – 6 weeks) and adopts the most appropriate approach

## 9 Underlying Principles for DSDM

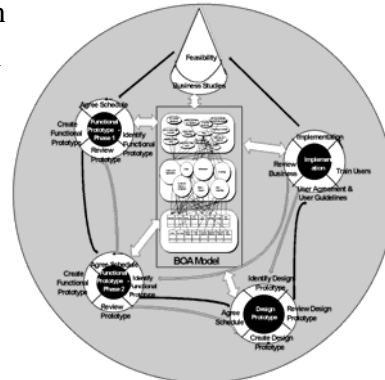
4. Fitness for purpose is the key criterion. DSDM is geared to delivering essential functionality at the specified time
5. Iterative and incremental development is necessary to converge on an accurate business solution. Incremental development allows user feedback to inform later development
6. All changes during development are reversible. This allows inappropriate iterations to be undone

## 9 Underlying Principles for DSDM

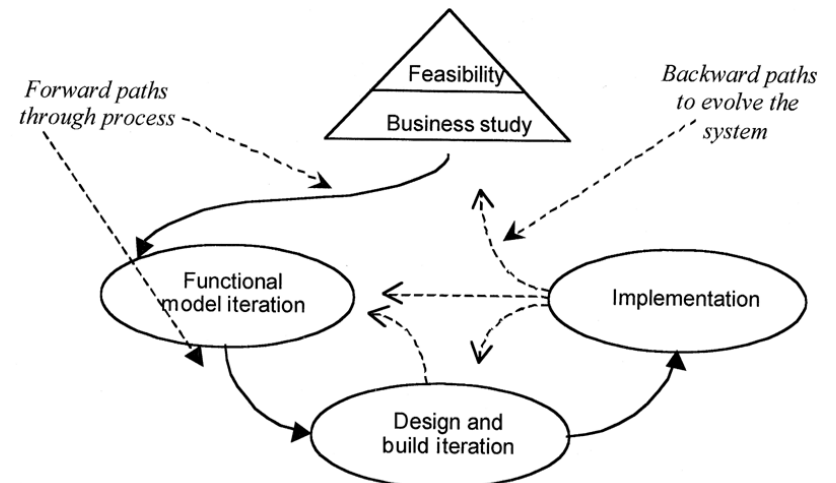
7. Requirements are initially agreed at a high level. These provide objectives for prototyping and can be investigated by the teams. Normally the scope of the high level requirements is not changed significantly
8. Testing is integrated throughout the life cycle – this is essential with an incremental approach. Developers test for technical compliance, user team members for functional compliance
9. A collaborative and co-operative approach between all stakeholders is essential. Includes resource managers and quality assurance teams

## DSDM Life Cycle

- The DSDM life cycle has these phases
  - ◆ Feasibility Study
  - ◆ Business Study
  - ◆ Functional Model Iteration
  - ◆ Design and Build Iteration
  - ◆ Implementation



## Simplified DSDM Life Cycle



## DSDM Life Cycle - Feasibility Study

- Determines whether the project is suitable for a DSDM approach
- Typically lasts only weeks
- Should also address the following questions
  - ◆ Is the computerized information system technically possible?
  - ◆ Will the benefit of the system be outweighed by its costs?
  - ◆ Will the information system operate acceptably within the organization?

## DSDM Life Cycle - Business Study

- Identifies the overall scope of the project and results in agreed high level functional and non-functional requirements
- Maintainability objectives are set at this stage and these determine the quality control activities for the remainder of the project
  - ◆ Maintainable from initial operation
  - ◆ Not necessarily maintainable when first installed but this can be addressed later
  - ◆ Short life-span system that will not be subject to maintenance

## DSDM Life Cycle - Functional Model Iteration

- Concerned with the development of prototypes to elicit detailed requirements
- Prototypes that can ultimately be delivered as operational systems are built – for operational use and non-functional requirements
- Comprises high level analysis models and documentation together with prototypes that are concerned with detailed functionality and usability

Activities

- ◆ The functional prototype is identified
- ◆ A schedule is agreed
- ◆ The functional prototype is created
- ◆ The functional prototype is reviewed

## DSDM Life Cycle - Design and Build Iteration

- Concerned with developing the prototypes to the point where they can be used operationally
- Distinction between the functional model iteration and the design and build iteration is not clear-cut and both phases can run concurrently
- Similar activities to functional model iteration

## DSDM Life Cycle - Implementation

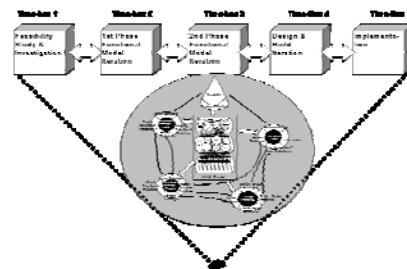
- The installation of the latest increment including user training
- Review the extent to which the requirements have been met
- Return to the design and build iteration to complete non-functional requirements
- Return to the functional model iteration to complete functional requirements
- May return to the business study for newly identified functional area

## DSDM Life Cycle - Implementation

- Implementation comprises the following iterative activities
  - ◆ Producing user guidelines and gaining user approval
  - ◆ Training users
  - ◆ Implementing the system
  - ◆ Reviewing the business requirements

## Timeboxing

- Fixing the resource allocation for a project or a part of a project
- Limits the time available for the refinement of requirements, design, construction and implementation as appropriate
- Each time box has a set of prioritised objectives



## Timeboxing

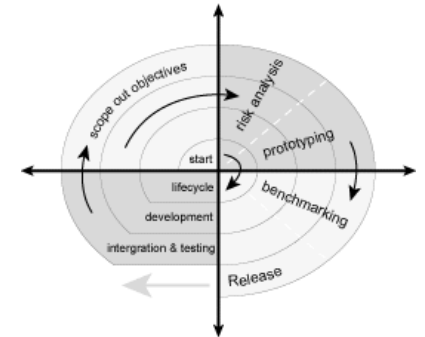
- Each time box produces one or more deliverables
- Within a time box the team have three major concerns
  - ◆ Carry out any investigation needed to determine the direction
  - ◆ Develop and refine the specified deliverables
  - ◆ Consolidate their work prior to the final deadline

## MoSCoW Rules

- Heuristic to help prioritise time box requirements – (Must, Should, Could, Want)
  - ◆ **Must** have requirements are crucial
  - ◆ **Should** have requirements are important
  - ◆ **Could** have requirements are less important
  - ◆ **Want** to have but will not have this time around requirements can reasonably be left for development in a later increment

## Extreme Programming (XP)

- XP is a novel combination of elements of best practice in systems development
- First publicized by Kent Beck (Beck, 2000)
- Known for its use of pair programming but has other important aspects



## Underlying Principles of XP

- Communication
- Simplicity
- Feedback
- Courage



## Underlying Principles: Communication

- XP highlights the importance of good communication among developers and between developers and users

## Underlying Principles: Simplicity

- XP focuses on the simplest solution for the immediate known requirements

## Underlying Principles: Feedback

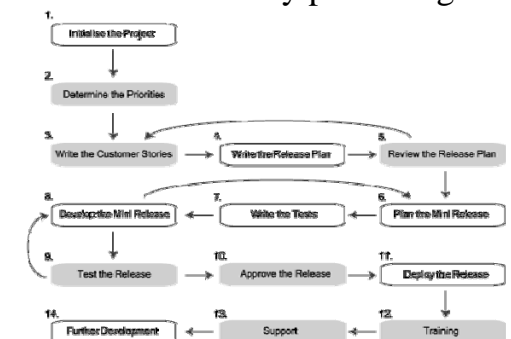
- Feedback in XP is geared to giving the developers frequent and timely feedback from users and also in terms of test results.
- Work estimates are based on the work actually completed in the previous iteration

## Underlying Principles: Courage

- Urges the developer to throw away code that is not quite correct and start again.
- Essentially the developer has to leave unproductive lines of development despite personal investment in the ideas

## Extreme Programming

- XP also emphasises
  - ◆ Embracing change is important and key to systems development
  - ◆ Development staff are motivated by producing quality work



## Requirements Capture in XP

- Based on user stories that describe the requirements
  - ◆ Written by the user
  - ◆ Basis of project planning and the development of test harnesses
- Very similar to use cases though some proponents of XP suggest that there are key differences in granularity
  - ◆ Typical user story is about three sentences with no technology indicated
  - ◆ Developers get detailed descriptions from the customer when they start developing

## XP Activities

- The planning game involves quickly defining the scope of the next release from user priorities and technical estimates. The plan is updated regularly as the iteration progresses
- The information system should be delivered in small releases that incrementally build up functionality through rapid iteration
- A unifying metaphor or high level shared story focuses the development

## XP Activities (cont')

- The system should be based on a simple design
- Programmers prepare unit tests in advance of software construction and customers define acceptance tests
- The programme code should be restructured to remove duplication, simplify the code and improve flexibility – this is known as refactoring, and is discussed in Fowler (1999) in detail

## XP Activities (cont')

- Pair programming means that code is written by two programmers using one workstation
- The code is owned collectively and anyone can change any code
- The system is integrated and built frequently each day. This gives the opportunity for regular testing and feedback

## XP Activities (cont')

- Normally staff should work no more than forty hours a week
- A user should be a full-time member of the team
- All programmers should write code according to agreed standards that emphasise good communication through the code

## Using XP

- Best suited to projects with a relatively small number of programmers – say no more than ten
- Critical to maintain clear communicative code and to have rapid feedback
  - ◆ If these are not possible then XP would be problematic
- XP not sympathetic to using UML for analysis & design