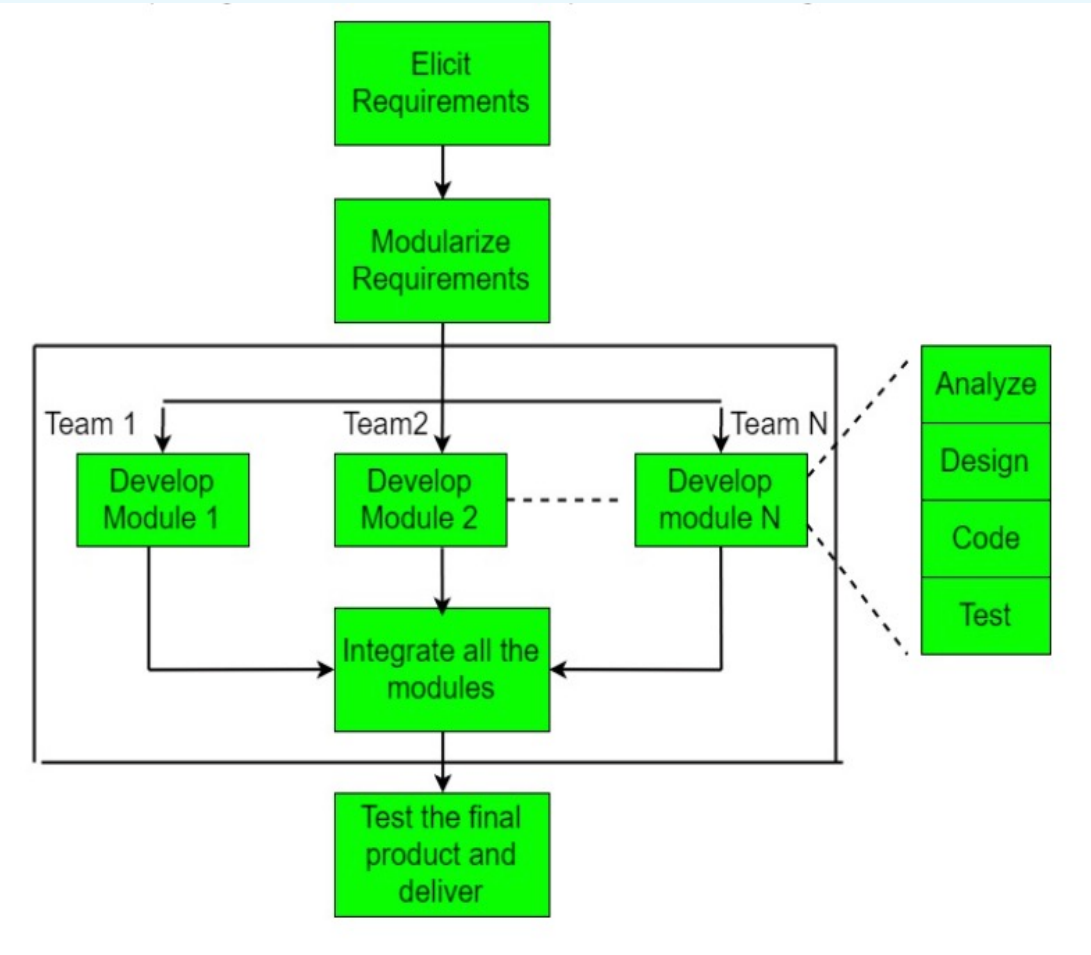
# COMP1787 Requirements Management

Lecture 4 : Rapid  Application Development - Techniques

Dr. Aditi Rawal

UNIVERSITY OF GREENWICH

# Overview of RAD




- Short development cycles/ timebox (30 - 90days)
- Continuous user participation and feedback
- Project can be broken down into smaller modules
- Use of powerful development tools and techniques

Source: https://www.geeksforgeeks.org/software-engineering-rapid-application-development-model-rad/

UNIVERSITY OF GREENWICH

# Overview of the RAD techniques

Several techniques are used in RAD environments.

It is the use of these techniques, together with the underlying philosophy of RAD, that enable Rapid Application Development to successfully take place.

The techniques we will be examining are:

**Prototyping**

**JAD**

**Timeboxing**

UNIVERSITY OF GREENWICH

# Technique 1 – Timeboxes

# Timeboxes – definitions

A method for progressively developing and delivering operational system functionality within predefined timeframes.

- Timeboxes can be used at varying time scales.
- Something useful to be delivered at the end.
- Strict time to be kept.

UNIVERSITY OF GREENWICH

# Timebox in detail

It is a fixed timescale for a particular RAD activity e.g. *develop mortgage quotation functionality for an online mortgage service*

The deadline is immovable

Developers and users both agree that some functionality will be delivered by the deadline

this means that through prototyping/incremental development *something* that is *useable* and *of importance* is delivered by the deadline

UNIVERSITY OF GREENWICH

# RAD and timeboxing continued

- 90 days often considered maximum time for a timebox.
- Something workable must be delivered within the timebox.
- RAD partitions the development and delivers something quickly and often.
- **Being late is not an option**

# Timeboxes view point – Mainly product based

**Product-based view**

- This is where the project is driven by completion of a particular delivery (product) e.g. “ in two weeks' time I will have delivered the order entry input screens and processing”.

**Activity-based view**

- This is where the project is driven by activities that must be completed. E.g. “in two weeks', I will have finished modelling the user requirements”.

UNIVERSITY OF GREENWICH

# RAD and timeboxing

- High level requirements are determined
- The development process divides the system into components or timeboxes based on requirement priorities.
- System is prioritised into a number of timeboxes
- Most important requirements are those with the largest potential benefit.
    - These are developed in first timebox

# Development – Time and Resources

Traditional Development **VS** Rapid Application Development


UNIVERSITY OF GREENWICH

# Technique 2 – MoSCoW Prioritisation

- To help  understand and manage priorities
- This prioritization method was developed by Dai Clegg in 1994 for use in rapid application development (RAD).

UNIVERSITY OF GREENWICH

# More on MoSCoW rules

All these requirements are essential for the full system.

These rules serve as the foundation for decision-making regarding the tasks undertaken by developers throughout the entire project and within each timebox.

UNIVERSITY OF GREENWICH

# Prioritising development for timeboxes (MoSCoW rules)

**Must** have *for requirements that are fundamental to the system*

**Should** have *for important requirements that would probably be classed as mandatory in a less time-constrained environment*

**Could** have *for requirements that can more easily be left out of the increment under development*

**Won't** have – want these but will not have this time round *for those valuable requirements that can wait until later development takes place*

UNIVERSITY OF GREENWICH

# Prioritising development for timeboxes (MoSCoW rules)

Must have

- *The key question to ask*
  - ***what happens if this requirement is not met****?*
  - *If the answer is " Cancel the project"- there is no point in implementing a solution that does not meet this requirement'*
  - *then it is a Must Have requirement.*
  - *If there is some way around it, even if it is a manual and painful workaround, then it is a Should Have or a Could Have requirement.*
  - *Categorising a requirement as a Should Have or Could Have does not mean it won't be delivered; simply that delivery is not guaranteed.*

Source: https://www.agilebusiness.org/dsdm-project-framework/moscow-prioririsation.html

UNIVERSITY OF GREENWICH

# Prioritising development for timeboxes (MoSCoW rules)

**Should** have

*Important but not vital*

- *May be painful to leave out, but the solution is still viable*
- *May need some kind of workaround, e.g. management of expectations, some inefficiency, an existing solution, paperwork etc. The workaround may be just a temporary one*
- *One way of differentiating a Should Have requirement from a Could Have is by reviewing the degree of pain caused by the requirement not being met, measured in terms of business value or numbers of people affected.*

Source: https://www.agilebusiness.org/dsdm-project-framework/moscow-prioririsation.html

UNIVERSITY OF GREENWICH

# Prioritising development for timeboxes (MoSCoW rules)

**Could** have

*Wanted or desirable but less important*

*Less impact if left out (compared with a Should Have)*

*These are the requirements that provide the main pool of contingency, since they would only be delivered in their entirety in a best case scenario.*

*When a problem occurs and the deadline is at risk, one or more of the Could haves provide the first choice of what is to be dropped from this timeframe.*

Source: https://www.agilebusiness.org/dsdm-project-framework/moscow-prioririsation.html

UNIVERSITY OF GREENWICH

# Prioritising development for timeboxes (MoSCoW rules)

**Won't** have

- *These are requirements which the project team has agreed will not be delivered (as part of this timeframe).*
- *They are recorded in the Prioritised Requirements List where they help clarify the scope of the project. This avoids them being informally reintroduced at a later date.*
- *This also helps to manage expectations that some requirements will simply not make it into the Deployed Solution, at least not this time around.*
- *Won't Haves can be very powerful in keeping the focus at this point in time on the more important Could Haves, Should Haves and particularly the Must Haves.*

Source: https://www.agilebusiness.org/dsdm-project-framework/moscow-prioririsation.html

UNIVERSITY OF GREENWICH

# Technique 3 - Facilitated Workshop

UNIVERSITY OF GREENWICH

# JAD (Joint Application Development) *Facilitated Workshops*

Basic idea is to:

| | |
|---|---|
| Select key end users | Conduct workshops that progress through structured set of steps for planning and designing a system |

Most methods now take Facilitated workshop to be any point in development process from inception to delivery

UNIVERSITY OF GREENWICH

# Faciliated Workshop – what's it all about?

Aim of a Facilitated Workshop is to produce **something** and to achieve **consensus** among participants as to the **content** of that thing.

Particularly useful at beginning of project, when trying to define scope etc.

It is a **facilitated** meeting designed to overcome the problems of traditional requirements gathering.

Aim to specify high-level requirements

UNIVERSITY OF GREENWICH

# Who attends a Facilitated Workshop?

- Different types of personnel must be involved in a Facilitated Workshop:
  - *Executive sponsor* – the person who has made a commitment to getting the system built. Kicks the meeting off, doesn't interfere!
  - *RAD workshop leader* – does preparation, leads the workshop etc.
  - *I.S professionals* – should be one or more of these. Must be involved in the project

UNIVERSITY OF GREENWICH

# Who attends a Facilitated Workshop?

*End users* - those commandeered to the project development plus other end-user reps.

*Scribe* – person responsible for the project documentation. Minutes this workshop or builds models using case tool etc….

*Visiting specialists* – Specialists may attend sessions as and when required

*Project manager* - may be present but shouldn't lead the session……

UNIVERSITY OF GREENWICH

# Reasons for using Facilitated Workshops

- Harness know-how of users
- Cut across organisational barriers
- Group dynamics drive the design
- Organised, controlled, structured process
- Experienced session leader facilitates discussion & drives the session to complete its agenda
- Includes management direction
- Utilises I.S advice and perspective

UNIVERSITY OF GREENWICH

# Technique 4 - Prototyping

UNIVERSITY OF GREENWICH

# What is Prototyping?

A definition:

- *The process of building an experimental system **quickly** and **inexpensively** for demonstration and evaluation so that users can better determine information requirements.*

Prototyping is an *iterative process.*

- *Iteration is the repetitive process of revisiting and refining the steps involved in building a system to achieve improvements or desired outcomes iteratively.*

UNIVERSITY OF GREENWICH

# Why prototype?

It is a response to user dissatisfaction found when using traditional approaches

It is seen as an improved form of systems investigation & analysis. Particularly useful where:

| | | |
|---|---|---|
| application area is not well defined | organisation not familiar with the technology | communication between analysts & users has not been good. |

UNIVERSITY OF GREENWICH

# What does it address?

Some of the problems of traditional systems analysis:

- Users only saw their information systems at point of implementation
- Systems Analyst  was really experimenting on the user
- System doesn’t always do what the users were expecting it to do.

UNIVERSITY OF GREENWICH

# Why prototype? (cont.)

Cost of rejection by users would be very high and is essential to ensure that the final version has got users' needs right.

There is a requirement to assess the impact of prospective information systems.

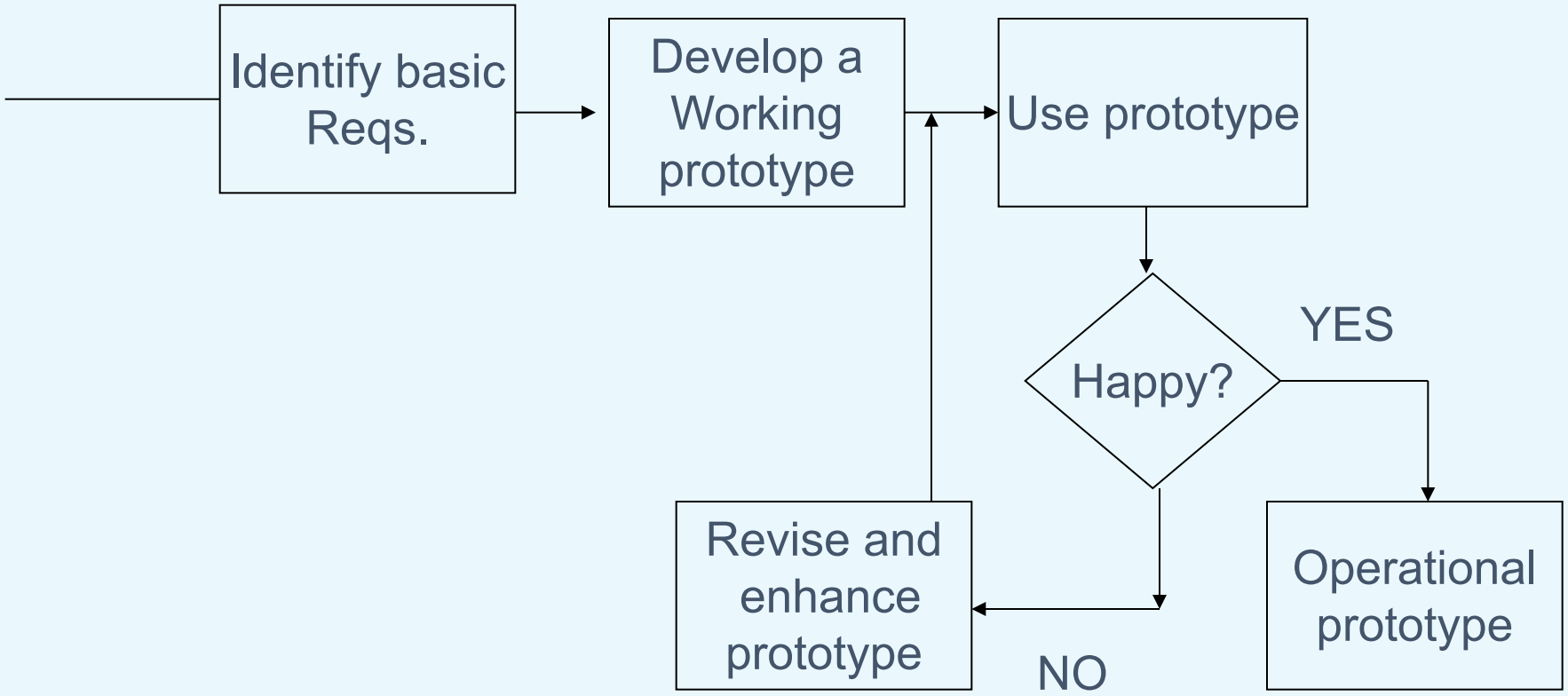
# Evolutionary or throwaway?

- At the end of the prototyping cycle there are two choices to be made:
  - Throwaway the prototype but use the prototype design as the basis for building an operational system - usually in a different software environment (throwaway)
  - Develop the prototype into a fully working system that will become operational within the organisation (evolutionary)

# Evolutionary or throwaway (2)

- Approach taken depends upon:
  - Software environment required for final operational system
  - How much of the final system is to be prototyped
  - Whether there is time to redevelop the system using another software environment

# Basic prototyping cycle




UNIVERSITY OF GREENWICH

# Different types of prototype(1)

- Different types with different objectives
  - **Requirements analysis prototypes**
    - Examine areas where users and analysts are unsure of requirements
    - Look at a physical approximation of a system to gather ideas
    - Prototyping viewed as a an improvement on traditional requirements gathering techniques

# Different types of prototype(2)

| | |
|---|---|
| **Functional prototypes** | • These are used for demonstrating, testing and evaluating the functionality of a system |
| **Process prototypes** | • These are used for demonstrating, testing and evaluating the processes, sequences, responses etc. of a system |
| **Design prototypes** | • For demonstrating and evaluating a variety of alternate UI designs or solutions |
| **Performance prototypes** | • These are used for testing, response times, loads, volumes etc. |

# Problems with prototyping

## Must be properly managed and controlled

- Too many iterations can lead to a ‘runaway project’.
- Between 3 – 5 iterations is usually considered sensible.

## Prototyping can lead to the proliferation of requirements

## It can be difficult to get good user participation

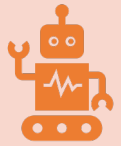
- Lack of management commitment
- Users not given sufficient time or incentive to co-operate

UNIVERSITY OF GREENWICH

# Problems with prototyping cont..

Expectations with the prototyping process can be too high

Users may think that they will get the system they want through prototyping process



Prototype originally designated as ‘throwaway’ may become operational

Documentation may be neglected because ‘it was only supposed to be a prototype’.

UNIVERSITY OF GREENWICH

# Prototyping – things to remember

Prototyping must be carefully managed

There must be understanding of, and commitment to, the process by both management and users.

Controls during the prototyping process must be maintained

**Prototyping is not a substitute for thorough analysis and design**

UNIVERSITY OF GREENWICH

# In summary (1)

There are several key techniques associated with Rapid Development (Agile methods)

- Prototyping
- Facilitated workshops
- Timeboxing
- MoSCoW rules

These techniques can be used standalone as well as part of RAD

# In summary (2)

These techniques are designed to:

- Incorporate users into all stages of the life cycle
- Enable an incremental approach to development
- Enable an iterative approach to delivery

# Thank you.

UNIVERSITY OF GREENWICH