

## Software Development Life Cycle (SDLC)

## Capability Maturity Model (CMM)

- ▶ A bench-mark for measuring the maturity of an organisation's software process
- ▶ CMM defines 5 levels of process maturity based on certain Key Process Areas (KPA)

## CMM Levels

### Level 5 – Optimizing (< 1%)

- process change management
- technology change management
- defect prevention

### Level 4 – Managed (< 5%)

- software quality management
- quantitative process management

### Level 3 – Defined (< 10%)

- peer reviews
- intergroup coordination
- software product engineering
- integrated software management
- training program
- organisation process definition
- organisation process focus

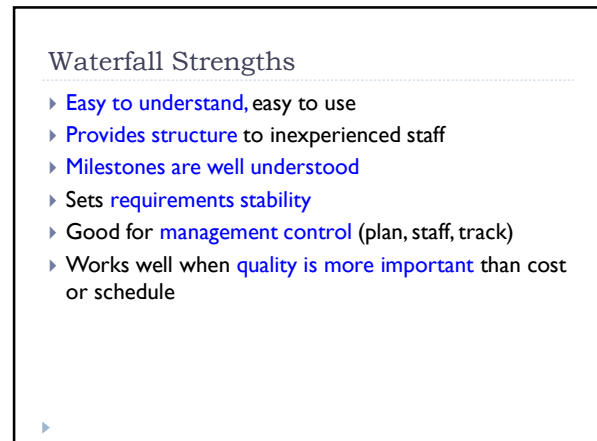
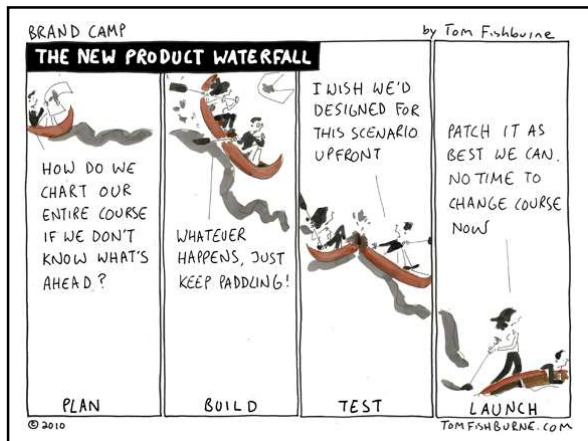
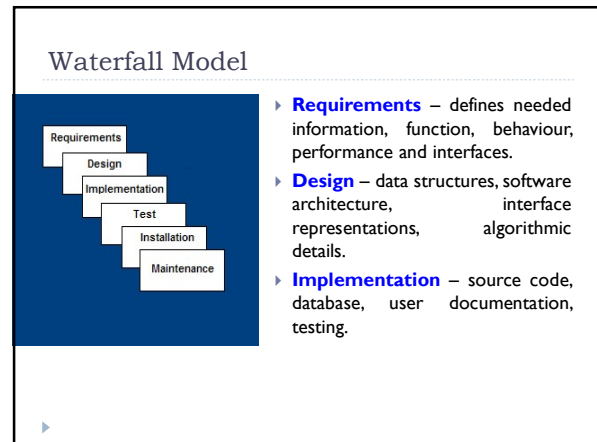
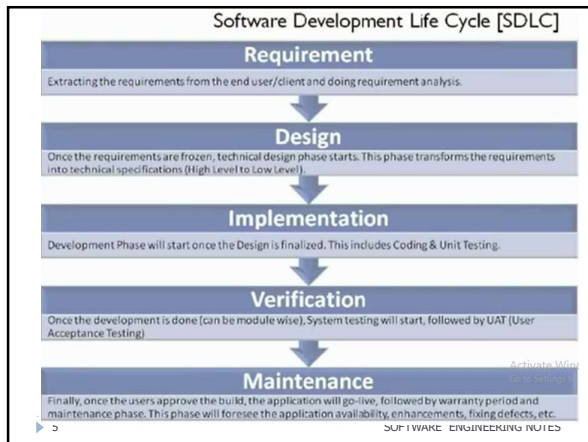
### Level 2 – Repeatable (~ 15%)

- software configuration management
- software quality assurance
- software project tracking and oversight
- software project planning
- requirements management

### Level 1 – Initial (~ 70%)

## SDLC (Software Development Life Cycle) Model

A framework that describes the activities performed at each stage of a software development project.



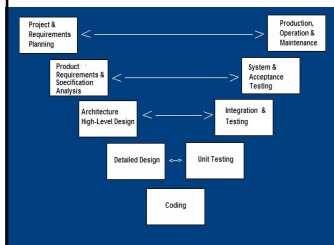
### Waterfall Deficiencies

- ▶ All **requirements must be known** upfront
- ▶ Deliverables created for each phase are considered frozen – **inhibits flexibility**
- ▶ Can give a **false impression of progress**
- ▶ **Does not reflect problem-solving nature** of software development – iterations of phases
- ▶ Integration is **one big bang at the end**
- ▶ **Little opportunity for customer** to preview the system (until it may be too late)

### When to use the Waterfall Model

- Requirements are very **well known**
- Product definition is **stable**
- Technology is **understood**
- New **version of an existing product**
- **Porting an existing product** to a new platform.
  - High risk for new systems because of specification and design problems.
  - Low risk for well-understood developments using familiar technology.

### V-Shaped SDLC Model



- ▶ A variant of the Waterfall that emphasizes the verification and validation of the product.
- ▶ Testing of the product is planned in parallel with a corresponding phase of development

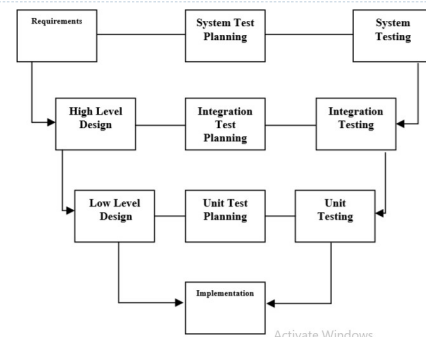
### V-Shaped Model

- ▶ Its life cycle is a sequential path of execution of processes just like the waterfall model.
  - Each phase must be completed before the next phase begins.
- ▶ Testing is emphasised more than in the waterfall model.
- ▶ The testing procedures are developed early in the life cycle **before any coding is done**, during each of the phases preceding implementation.
- ▶ **Requirements** begin the life cycle model just like the waterfall model.
  - Before development is started, a **system test plan** is created.
  - The test plan focuses on meeting the functionality specified in **requirements gathering**.

### V-Shaped Model

- ▶ The **high-level design phase** focuses on system architecture and design.
  - ▶ An **integration test plan** is created in this phase in order to test the pieces of the software systems ability to work together.
- ▶ In the **low-level design phase** the actual software components are designed.
  - ▶ **Unit tests** are created in this phase as well.
- ▶ The **implementation phase** is where all coding takes place.
- ▶ Once coding is complete, the path of execution continues up the right side of the V where the test plans developed earlier are now put to use.

### V-Shaped SDLC Model



▶ 14

### V-Shaped Steps

- |  |   |
|--|---|
| <ul style="list-style-type: none"> <li>▶ <b>Project and Requirements Planning</b> – allocate resources</li> <li><b>Product Requirements and Specification Analysis</b> – complete specification of the software system</li> <li><b>Architecture or High-Level Design</b> – defines how software functions fulfill the design</li> <li><b>Detailed Design</b> – develop each algorithms for each architectural component</li> </ul> | <ul style="list-style-type: none"> <li>▶ <b>Production, operation and maintenance</b> – provide for enhancement and corrections</li> <li>▶ <b>System and acceptance testing</b> – check the entire software system in its environment</li> <li>▶ <b>Integration and Testing</b> – check that modules interconnect correctly</li> <li>▶ <b>Unit testing</b> – check that each module acts as expected</li> <li>▶ <b>Coding</b> – transform algorithms into software</li> </ul> |
|--|---|

### V-Shaped Strengths

- ▶ Emphasize planning for **verification and validation** of the product in early stages of product development
- ▶ **Each deliverable must be testable**
- ▶ Project management can **track progress by milestones**
- ▶ **Easy to use**

### V-Shaped Weaknesses

- ▶ Does not easily handle **concurrent events**
- ▶ Does not handle **iterations** or phases
- ▶ Does not easily handle **dynamic changes in requirements**
- ▶ Does not contain **risk analysis** activities
  - ▶ There is almost always an element of risk involved in the devt of sw. e.g.
    - ▶ key personnel can resign before the product has been adequately documented.
    - ▶ The manufacturer of the hardware on which the product is critically dependent can go bankrupt.
    - ▶ Too much, or too little, can be invested in testing and quality assurance.
    - ▶ After spending millions of shillings on developing a major sw product, technological breakthroughs can render the entire product worthless<sup>©</sup>
  - ▶ One way of minimising certain types of risk is to **construct a prototype**.

### When to use the V-Shaped Model

- ▶ Excellent choice for **systems requiring high reliability**
  - ▶ hospital patient control applications
- ▶ **All requirements are known** up-front
- ▶ When it can be modified to **handle changing requirements beyond analysis phase**
- ▶ **Solution and technology are known**

### Prototyping

19

SOFTWARE ENGINEERING NOTES

### Software Prototyping

- ▶ A prototype is an initial version of a system used to demonstrate concepts and try out design options.
- ▶ A prototype can be used in:
  - ▶ The **requirements engineering process** to help with requirements elicitation and validation;
  - ▶ In **design processes** to explore options and develop a UI design;
  - ▶ In the **testing process** to run back-to-back tests.

▶ 20

### Protoyping: Basic Steps

- **Identify basic requirements**
  - Including input and output info
  - Details (e.g., security) generally ignored
- **Develop initial prototype**
  - UI first
- **Review**
  - Customers/end –users review and give feedback
- **Revise and enhance the prototype & specs**
  - Negotiation about scope of contract may be necessary



### Dimensions of prototyping

- ▶ **Horizontal prototype**
  - Broad view of entire system/sub-system
  - Focus is on user interaction more than low-level system functionality (e.g., database access)
  - Useful for:
    - Confirmation of UI requirements and system scope
    - Demonstration version of the system to obtain buy-in from business/customers
    - Develop preliminary estimates of development time, cost, effort



### Dimensions of Prototyping

- ▶ **Vertical prototype**
  - More complete elaboration of a single sub-system or function
  - Useful for:
    - Obtaining detailed requirements for a given function
    - Refining database design
    - Obtaining info on system interface needs
    - Clarifying complex requirements by drilling down to actual system functionality



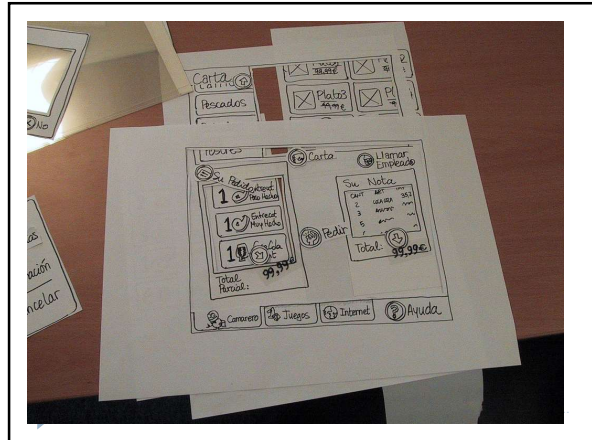
### Types of prototyping

- **Throwaway /rapid/close-ended prototyping**
  - Creation of a model that will be discarded rather than becoming part of the final delivered software
  - After preliminary requirements gathering, used to visually show the users what their requirements may look like when implemented
- **Focus is on quickly developing the model**
  - not on good programming practices
  - Can Wizard of Oz things



## Fidelity of Prototype

- ▶ **Low-fidelity**
  - ▶ Paper/pencil
    - ▶ Mimics the functionality, but does not look like it



## Fidelity of Prototype

- **Medium to High-fidelity**
  - GUI builder
  - “Click dummy” prototype – looks like the system, but does not provide the functionality
  - Or provide functionality, but have it be general and not linked to specific data
  - <http://www.youtube.com/watch?v=VGjcFouSlpk>
  - <http://www.youtube.com/watch?v=5oLmNbxap4&feature=related>

## Throwaway Prototyping steps

- ▶ Write preliminary requirements
- ▶ Design the prototype
- ▶ User experiences/uses the prototype, specifies new requirements
- ▶ Repeat if necessary
- ▶ Write the final requirements
- ▶ Develop the real products

### Evolutionary Prototyping

- ▶ Aka **breadboard** prototyping
- ▶ Goal is to build a very robust prototype in a structured manner and constantly refine it
- ▶ The evolutionary prototype forms the heart of the new system and is added to and refined
- ▶ Allow the development team to add features or make changes that were not conceived in the initial requirements

### Evolutionary Prototyping Model

- ▶ Developers build a prototype during the requirements phase
- ▶ Prototype is evaluated by end users
- ▶ Users give corrective feedback
- ▶ Developers further refine the prototype
- ▶ When the user is satisfied, the prototype code is brought up to the standards needed for a final product.

### EP Steps

- ▶ A preliminary project plan is developed
- ▶ An partial high-level paper model is created
- ▶ The model is source for a partial requirements specification
- ▶ A prototype is built with basic and critical attributes
- ▶ The designer builds
  - ▶ the database
  - ▶ user interface
  - ▶ algorithmic functions
- ▶ The designer demonstrates the prototype, the user evaluates for problems and suggests improvements.
- ▶ This loop continues until the user is satisfied

### EP Strengths

- ▶ Customers can “see” the system requirements as they are being gathered
- ▶ Developers learn from customers
- ▶ A more accurate end product
- ▶ Unexpected requirements accommodated
- ▶ Allows for flexible design and development
- ▶ Steady, visible signs of progress produced
- ▶ Interaction with the prototype stimulates awareness of additional needed functionality



### Incremental prototyping

- ▶ Final product built as separate prototypes
- ▶ At the end, the prototypes are merged into a final design



### Extreme Prototyping

- Often used for web applications
- Development broken down into 3 phases, each based on the preceding 1
  1. Static prototype consisting of HTML pages
  2. Screens are programmed and fully functional using a simulated services layer
    - Fully functional UI is developed with little regard to the services, other than their contract
  3. Services are implemented



### Prototyping advantages

- Reduced time and cost
  - Can improve the quality of requirements and specifications provided to developers
    - Early determination of what the user really wants can result in faster and less expensive software
- Improved/increased user involvement
  - User can see and interact with the prototype, allowing them to provide better/more complete feedback and specs
  - Misunderstandings/miscommunications revealed
  - Final product more likely to satisfy their desired look/feel/performance



### Disadvantages of prototyping 1

- ▶ Insufficient analysis
  - ▶ Focus on limited prototype can distract developers from analyzing complete project
  - ▶ May overlook better solutions
  - ▶ Conversion of limited prototypes into poorly engineered final projects that are hard to maintain
  - ▶ Limited functionality may not scale well if used as the basis of a final deliverable
    - ▶ May not be noticed if developers too focused on building prototype as a model



### Disadvantages of prototyping 2

- ▶ **User confusion of prototype and finished system**
  - ▶ Users can think that a prototype (intended to be thrown away) is actually a final system that needs to be polished
    - ▶ Unaware of the scope of programming needed to give prototype robust functionality
  - ▶ Users can become attached to features included in prototype for consideration and then removed from final specification



### Disadvantages of prototyping 3

- ▶ **Developer attachment to prototype**
  - ▶ If spend a great deal of time/effort to produce, may become attached
  - ▶ Might try to attempt to convert a limited prototype into a final system
    - ▶ Bad if the prototype does not have an appropriate underlying architecture



### Disadvantages of prototyping 4

- ▶ **Excessive development time of the prototype**
  - ▶ Prototyping supposed to be done quickly
  - ▶ If developers lose sight of this, can try to build a prototype that is too complex
  - ▶ For throw away prototypes, the benefits realized from the prototype (precise requirements) may not offset the time spent in developing the prototype – expected productivity reduced
  - ▶ Users can be stuck in debates over prototype details and hold up development process



### Disadvantages of prototyping 5

- ▶ **Expense of implementing prototyping**
  - ▶ Start up costs of prototyping may be high
  - ▶ Expensive to change development methodologies in place (re-training, re-tooling)
  - ▶ Slow development if proper training not in place
    - ▶ High expectations for productivity unrealistic if insufficient recognition of the learning curve
  - ▶ Lower productivity can result if you overlook the need to develop corporate and project specific underlying structure to support the technology
    - ▶ ...the org has to offer corporate support for the process e.g. training and tools

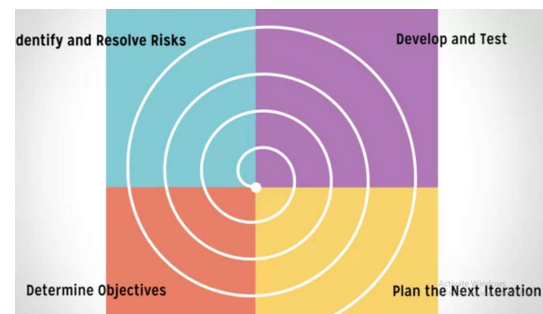


### Best uses of prototyping

- ▶ Most beneficial for systems that will have many interactions with end users
- ▶ The greater the interaction between the computer and the user, the greater the benefit of building a quick system for the user to play with
- ▶ Especially good for designing good human-computer interfaces

▶

### The Spiral SDLC Model



▶ 42

SOFTWARE ENGINEERING NOTES

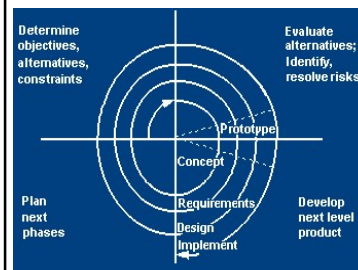
### The Spiral SDLC Model

- ▶ An attribute of this model is that it's not really a very defined or a step by step process.
  - ▶ i.e. it's a **process model generator**, not a process itself.
- ▶ So you actually use other models wherever appropriate to execute.
- ▶ E.g., in the development and test approach you could be using a **waterfall** approach.

▶ 43

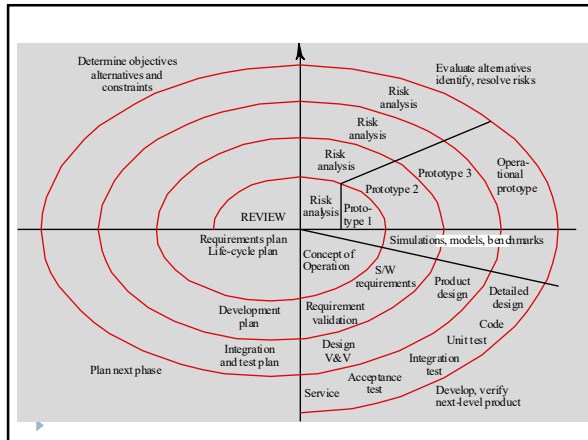
SOFTWARE ENGINEERING NOTES

### Spiral SDLC Model



- Combines features of the prototyping and the waterfall models.
  - Adds risk analysis, and 4GL RAD prototyping to the waterfall model
- Aids in risk management at regular stages and is iterative.
- Each cycle involves the same sequence of steps as the waterfall process model.
  - Iterations of stages (requirements, design, development, testing etc) go to  $n$  number of cycles until you reach the final goal,

▶



Spiral Quadrant: **Determine objectives, alternatives and constraints**

- ▶ **Objectives:** functionality, performance, hardware/software interface, critical success factors, etc.
- ▶ **Alternatives:** build, reuse, buy, sub-contract, etc.
- ▶ **Constraints:** cost, schedule, interface, etc.

Spiral Quadrant: **Evaluate alternatives, identify and resolve risks**

- ▶ **Study alternatives** relative to objectives and constraints
- ▶ **Identify risks** (lack of experience, new technology, tight schedules, poor process, etc.
- ▶ **Resolve risks** (evaluate if money could be lost by continuing system development)

Spiral Quadrant: **Develop next-level product**

- ▶ **Typical activities:**
  - ▶ Create a design
  - ▶ Review design
  - ▶ Develop code
  - ▶ Inspect code
  - ▶ Test product

#### Spiral Quadrant: Plan next phase

##### ▶ Typical activities

- ▶ Develop project plan
- ▶ Develop configuration management plan
- ▶ Develop a test plan
- ▶ Develop an installation plan

▶

#### Spiral Model Strengths

- ▶ Provides early indication of insurmountable risks, without much cost
- ▶ Users see the system early because of rapid prototyping tools
- ▶ Critical high-risk functions are developed first
- ▶ The design does not have to be perfect
- ▶ Users can be closely tied to all lifecycle steps
- ▶ Early and frequent feedback from users
- ▶ Cumulative costs assessed frequently

▶

#### Spiral Model Weaknesses

- ▶ Time spent for evaluating risks too large for small or low-risk projects
- ▶ Time spent planning, resetting objectives, doing risk analysis and prototyping may be excessive
- ▶ The model is complex
- ▶ Risk assessment expertise is required
- ▶ Spiral may continue indefinitely
- ▶ Developers must be reassigned during non-development phase activities
- ▶ May be hard to define objective, verifiable milestones that indicate readiness to proceed through the next iteration

▶

#### When to use Spiral Model

- It is intended for large, expensive and complicated projects. It should be used:
  - When creation of a prototype is appropriate
  - When costs and risk evaluation is important
  - For medium to high-risk projects
  - Long-term project commitment unwise because of potential changes to economic priorities
  - Users are unsure of their needs
  - Requirements are complex
  - New product line
  - Significant changes are expected (research and exploration)

▶

### The Rise and Fall of Waterfall

- ▶ <http://www.youtube.com/watch?v=XIc2--sP3o0&NR=1&feature=fvwp>
- ▶ Warning: bad language at 3:50! (hands over ears if easily offended!)

### Agile software development life cycles

### Agile SDLC's

- ▶ **Speed up or bypass** one or more life cycle phases
- ▶ Usually **less formal** and **reduced scope**
- ▶ Used for **time-critical** applications
- ▶ Used in organisations that employ **disciplined methods**

### The Agile Manifesto—a statement of values

Individuals and interactions	over	Process and tools
Working software	over	Comprehensive documentation
Customer collaboration	over	Contract negotiation
Responding to change	over	Following a plan

Source: [www.agilemanifesto.org](http://www.agilemanifesto.org)

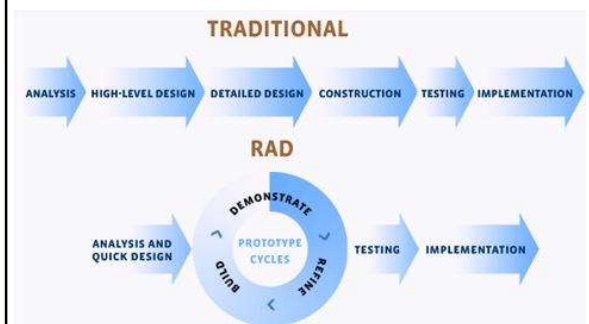
### Some Agile Methods

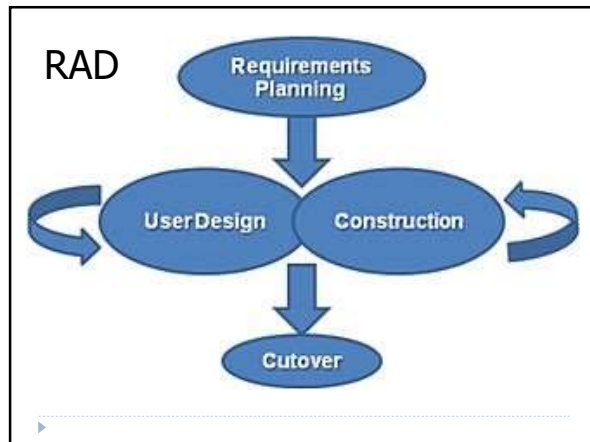
1. Rapid Application Development (RAD)
2. Incremental SDLC
3. Scrum
4. Extreme Programming (XP)
5. Adaptive Software Development (ASD)
6. Feature Driven Development (FDD)
7. Crystal Clear
8. Dynamic Software Development Method (DSDM)
9. Rational Unify Process (RUP)

### Agile vs Waterfall Propaganda

► <http://www.youtube.com/watch?v=gDDO3ob-4ZY&feature=related>

### Rapid application development (RAD) Model





### Rapid Application Model (RAD)

- ▶ **Requirements planning phase** (a workshop utilizing structured discussion of business problems)
- ▶ **User description phase** – automated tools capture information from users
- ▶ **Construction phase** – productivity tools, such as code generators, screen generators, etc. inside a time-box. (“Do until done”)
- ▶ **Cutover phase** -- installation of the system, user acceptance testing and user training

### Timeboxing

- ▶ A planning technique common in planning projects (typically for software development), where the schedule is divided into a number of separate time periods (**timeboxes**, normally two to six weeks long), with each part having its own **deliverables, deadline and budget**.
- ▶ A core aspect of **rapid application development (RAD)** software development processes such as dynamic systems development method (**DSDM**) and other **agile** software development.
- ▶ Timeboxes are used as a form of **risk management**, especially for tasks that may easily extend past their deadlines.
  - ▶ The end date (deadline) is one of the primary drivers in the planning and should not be changed as it is usually linked to a delivery date of the product.
  - ▶ If the team exceeds the deadline, the team failed in proper planning and / or effective execution of the plan.
  - ▶ This can be the result of the wrong people on the wrong job (lack of communication between teams, lack of experience, lack of commitment / drive / motivation, lack of speed) or underestimation of the (complexity of the) requirements.
- ▶ When the team exceeds the deadline, the following actions might be taken after conferring with the Client:
  - ▶ Dropping requirements of lower impact (the ones that will not be directly missed by the user)
  - ▶ Working overtime to compensate for the time lost
  - ▶ Moving the deadline

▶ 63

SOFTWARE ENGINEERING NOTES

### Requirements Planning Phase

- ▶ Combines elements of the system planning and systems analysis phases of the System Development Life Cycle (SDLC).
- ▶ Users, managers, and IT staff members discuss and agree on **business needs, project scope, constraints, and system requirements**.
- ▶ It ends when the team agrees on the key issues and obtains management authorisation to continue.

▶



### User Design Phase

- ▶ Users interact with systems analysts and develop models and prototypes that represent all system processes, inputs, and outputs.
- ▶ Typically use a combination of Joint Application Development (JAD) techniques and CASE tools to translate user needs into working models.
  - ▶ JAD Techniques
    - ▶ [http://en.wikipedia.org/wiki/Joint\\_application\\_design](http://en.wikipedia.org/wiki/Joint_application_design)
  - ▶ CASE Tools
    - ▶ [http://en.wikipedia.org/wiki/Computer-aided\\_software\\_engineering](http://en.wikipedia.org/wiki/Computer-aided_software_engineering)
- ▶ A continuous interactive process that allows users to understand, modify, and eventually approve a working model of the system that meets their needs.

▶

### Construction Phase

- ▶ Focuses on program and application development task similar to the SDLC.
- ▶ However, users continue to participate and can still suggest changes or improvements as actual screens or reports are developed.
- ▶ Its tasks are
  - ▶ programming and application development,
  - ▶ coding,
  - ▶ unit-integration, and
  - ▶ system testing.

▶

### Cutover Phase

- ▶ Resembles the final tasks in the SDLC implementation phase.
- ▶ Compared with traditional methods, the entire process is compressed. As a result, the new system is built, delivered, and placed in operation much sooner.
- ▶ Tasks are
  - ▶ data conversion,
  - ▶ full-scale testing,
  - ▶ system changeover,
  - ▶ user training.

▶

### RAD Strengths

- ▶ Reduced cycle time and improved productivity with fewer people means lower costs
- ▶ Time-box approach mitigates cost and schedule risks
- ▶ Customer involved throughout the complete cycle minimises risk of not achieving customer satisfaction and business needs
- ▶ Focus moves from documentation to code (WYSIWYG).
- ▶ Uses modeling concepts to capture information about business, data, and processes.

▶

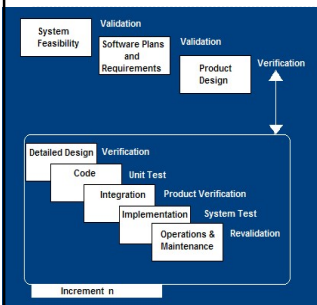
### RAD Weaknesses

- ▶ Accelerated development process **must give quick responses** to the user
- ▶ Risk of **never achieving closure**
- ▶ Hard to use with **legacy systems**
- ▶ Requires a system that can be **modularised**
- ▶ Developers and customers must be **committed to rapid-fire activities** in an abbreviated time frame.

### When to use RAD

- ▶ Reasonably **well-known requirements**
- ▶ User involved **throughout the life cycle**
- ▶ Project can be **time-boxed**
- ▶ Functionality delivered in **increments**
- ▶ **High performance** not required
- ▶ **Low technical risks**
- ▶ System **can be modularised**

### Incremental SDLC Model



- ▶ Construct a partial implementation of a total system
- ▶ Then slowly add increased functionality
- ▶ The incremental model prioritises requirements of the system and then implements them in groups.
- ▶ Each subsequent release of the system adds function to the previous release, until all designed functionality has been implemented.

### Incremental Model Strengths

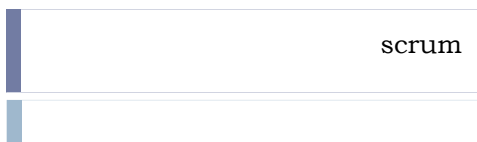
- ▶ Develop high-risk or **major functions first**
- ▶ Each release delivers an **operational product**
- ▶ Customer can **respond to each build**
- ▶ Uses “divide and conquer” **breakdown of tasks**
- ▶ Lowers **initial delivery cost**
- ▶ Initial **product delivery is faster**
- ▶ Customers get **important functionality early**
- ▶ Risk of **changing requirements** is reduced

### Incremental Model Weaknesses

- ▶ Requires good planning and design
- ▶ Requires early definition of a complete and fully functional system to allow for the definition of increments
- ▶ Well-defined module interfaces are required (some will be developed long before others)
- ▶ Total cost of the complete system is not lower

### When to use the Incremental Model

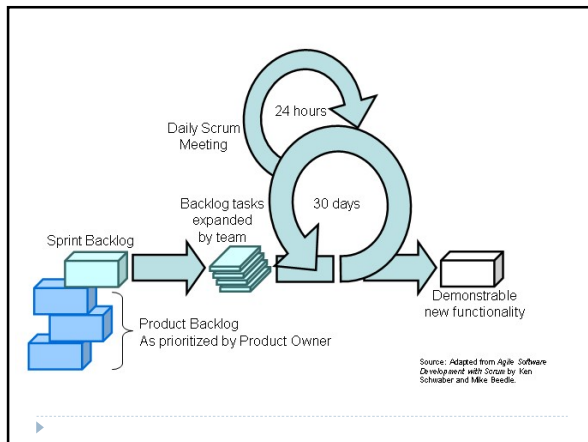
- ▶ Risk, funding, schedule, program complexity, or need for early realisation of benefits.
- ▶ Most of the requirements are known up-front but are expected to evolve over time
- ▶ A need to get basic functionality to the market early
- ▶ On projects which have lengthy development schedules
- ▶ On a project with new technology



### We're losing the relay race

"The... 'relay race' approach to product development...may conflict with the goals of maximum speed and flexibility. Instead a holistic or 'rugby' approach—where a team tries to go the distance as a unit, passing the ball back and forth—may better serve today's competitive requirements."

Hiroataka Takeuchi and Ikujiro Nonaka, "The New New Product Development Game", *Harvard Business Review*, January 1986.



#### Scrum in 13 seconds:

▶ <http://www.youtube.com/watch?v=9DKM9HcRnZ8&feature=related>

#### Scrum in 10 minutes:

▶ <http://www.youtube.com/watch?v=Q5k7a9YEoUJ>

#### More scrum slides:

▶ <http://www.mountaingoatsoftware.com/system/presentation/file/129/Getting-Agile-With-Scrum-Cohn-NDC2010.pdf?1276712017>

▶ Scalability of scrum addressed on slides 39-41

### Scrum advantages

- ▶ Agile scrum helps the company in saving time and money.
- ▶ Scrum methodology enables projects where the business requirements documentation is hard to quantify to be successfully developed.
- ▶ Fast moving, cutting edge developments can be quickly coded and tested using this method, as a mistake can be easily rectified.

### Scrum advantages

- ▶ It is a **lightly controlled** method which insists on frequent updating of the progress in work through regular meetings.
  - ▶ Thus there is clear visibility of the project development.
- ▶ Like any other agile methodology, this is also iterative in nature.
  - ▶ It requires continuous feedback from the user.
- ▶ Due to short sprints and constant feedback, it becomes easier to cope with the changes.

### Scrum advantages

- ▶ Daily meetings make it possible to measure individual productivity.
  - ▶ This leads to the improvement in the productivity of each of the team members.
- ▶ Issues are identified well in advance through the daily meetings and hence can be resolved in speedily
- ▶ It is easier to deliver a quality product in a scheduled time.

### Scrum advantages

- ▶ Agile Scrum can work with any technology/ programming language but is particularly useful for fast moving web 2.0 or new media projects.
- ▶ The overhead cost in terms of process and management is minimal thus leading to a quicker, cheaper result.

### Scrum disadvantages

- ▶ Agile Scrum is one of the leading causes of **scope creep** because unless there is a definite end date, the project management stakeholders will be tempted to keep demanding new functionality is delivered.
- ▶ If a task is not well defined, estimating project costs and time will not be accurate.
  - ▶ In such a case, the task can be spread over several sprints.
- ▶ If the team members are not committed, the project will either never complete or fail.

### Scrum disadvantages

- ▶ It is good for small, fast moving projects as it works well **only** with small team.
- ▶ This methodology needs experienced team members only.
  - ▶ If the team consists of people who are novices, the project cannot be completed in time.
- ▶ Scrum works well when the Scrum Master trusts the team they are managing.
  - ▶ If they practice control that is too strict over the team members, it can be extremely frustrating for them, leading to demoralisation and the failure of the project.

#### Scrum disadvantages

- ▶ If any of the team members leave during a development it can have a huge inverse effect on the project development
- ▶ Project quality management is hard to implement and quantify unless the test team are able to conduct regression testing after each sprint.

▶