Software Engineering (CSE3004)

Life cycle models

Puneet Kumar Jain

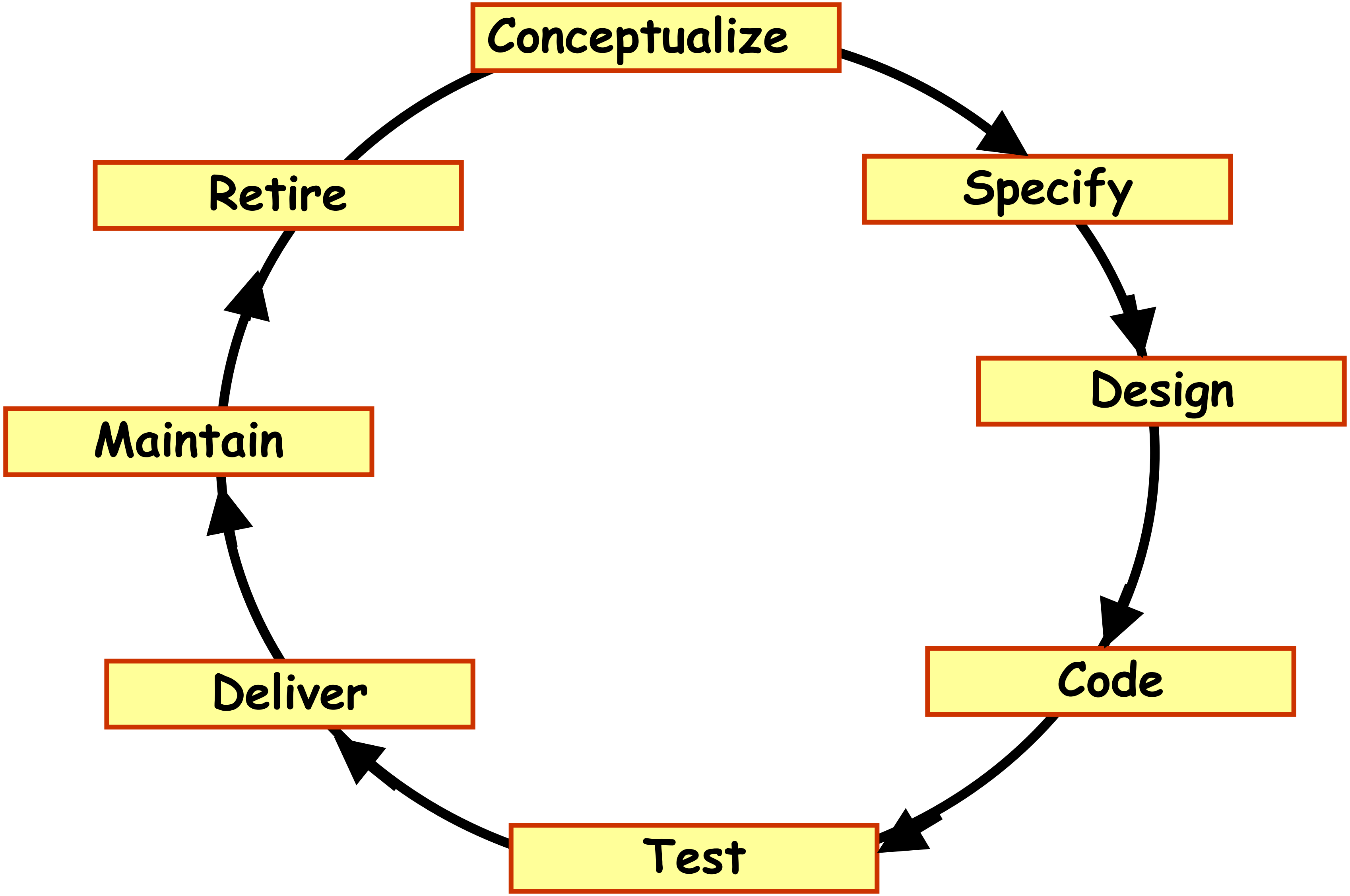
CSE Department

National Institute of Technology Rourkela

Reference

- Most of the slides belongs to the content prepared by Prof Rajib Mall, Fundamentals of Software Engineering, PHI, 2014

Software Life Cycle



Life Cycle Model

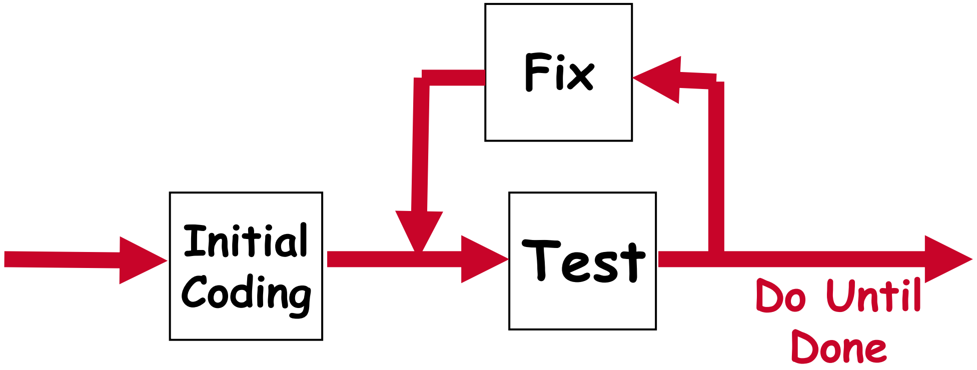
- A software life cycle model (also process model or SDLC):
  - A descriptive and diagrammatic model of software life cycle:
  - Identifies all the activities undertaken during product development,
  - Establishes a precedence ordering among the different activities,
  - Divides life cycle into phases.

Why Model Life Cycle?

- A graphical and written description:
  - **Helps common understanding of activities among the software developers.**
  - **Helps to identify inconsistencies, redundancies, and omissions in the development process.**
  - **Helps in tailoring a process model for specific projects.**

Life Cycle Model (CONT.)

- When a program is developed by a single programmer ---
  - The problem is within the grasp of an individual.
  - He has the freedom to decide his exact steps and still succeed --- called Exploratory model--- One can use it in many ways



  - Code→Test →Design
  - Code→Design →Test → Change Code →
  - Specify →code →Design →Test → etc.

Life Cycle Model (CONT.)

- When software is being developed by a team:
  - There must be a precise understanding among team members as to when to do what,
  - Otherwise, it would lead to chaos and project failure.

- The development team must identify a suitable life cycle model:
  - and then adhere to it.
  - Primary advantage of adhering to a life cycle model:
    - Helps development of software in a systematic and disciplined manner.

Phase Entry and Exit Criteria

- A life cycle model:
  - defines entry and exit criteria for every phase.
  - A phase is considered to be complete:
    - only when all its exit criteria are satisfied.

- For example, what is the phase exit criteria for the software requirements specification phase?
  - Software Requirements Specification (SRS) document is complete, reviewed, and approved by the customer.

Life Cycle Model: Milestones

- Milestones help software project managers:
  - To track the progress of the project.
  - To identify at which stage (e.g., design, code, test, etc.) the project is.
  - Phase entry and exit are important milestones.

Project Management Without Life Cycle Model

- It becomes very difficult to track the progress of the project.
  - The project manager would have to depend on the guesses of the team members.
- This usually leads to a problem:
  - known as the **99% complete syndrome.**

Life Cycle Model

- A descriptive and diagrammatic model of software life cycle:
- We confine our attention to only a few commonly used models.
  - **Waterfall**
  - **V model,**
  - **Prototyping**

Waterfall Approach

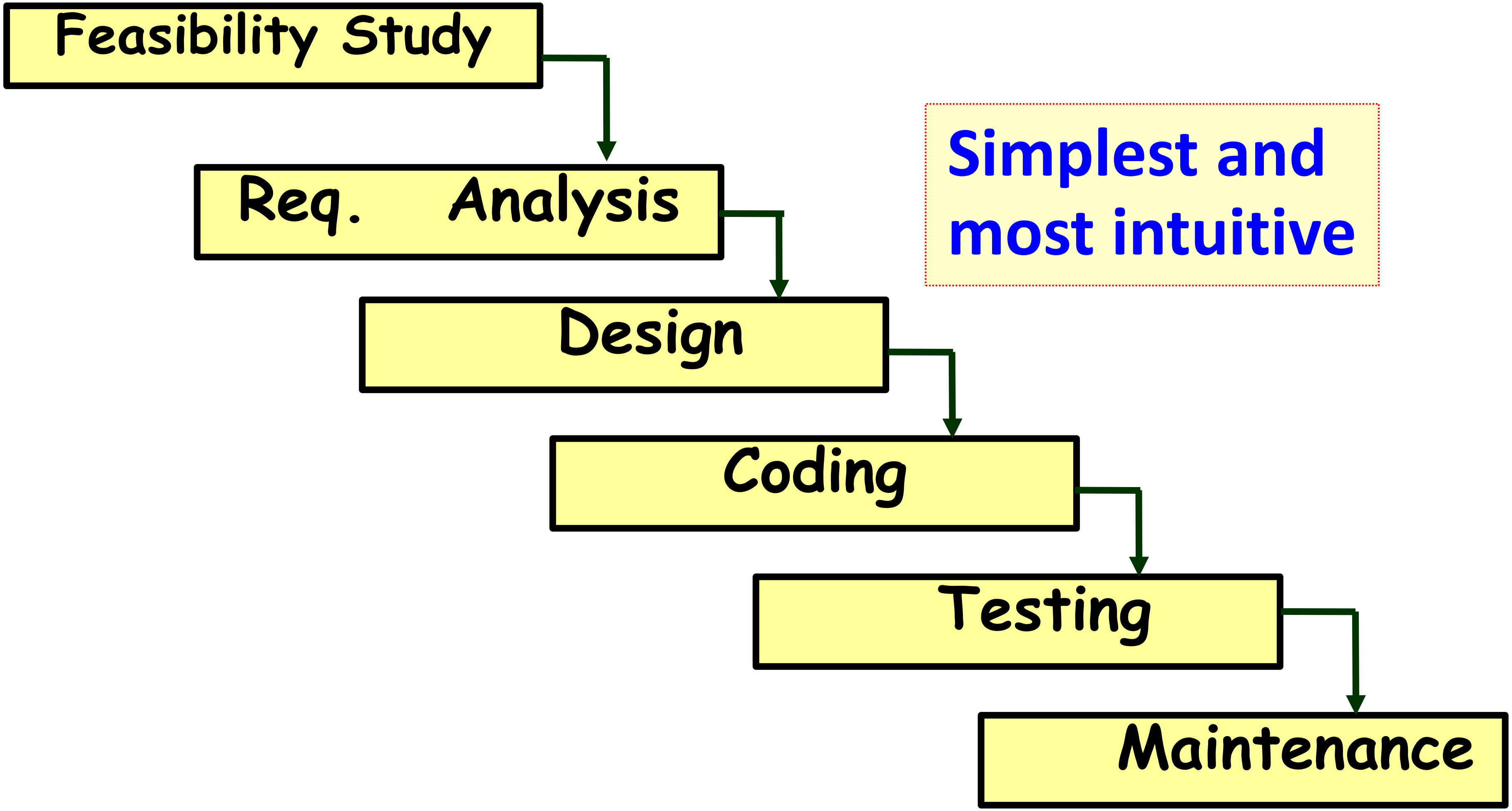
  - **Evolutionary**
  - **RAD**
  - **Unified Process**

IID Approach

  - **Spiral model**
  - **Agile models**

Waterfall Model

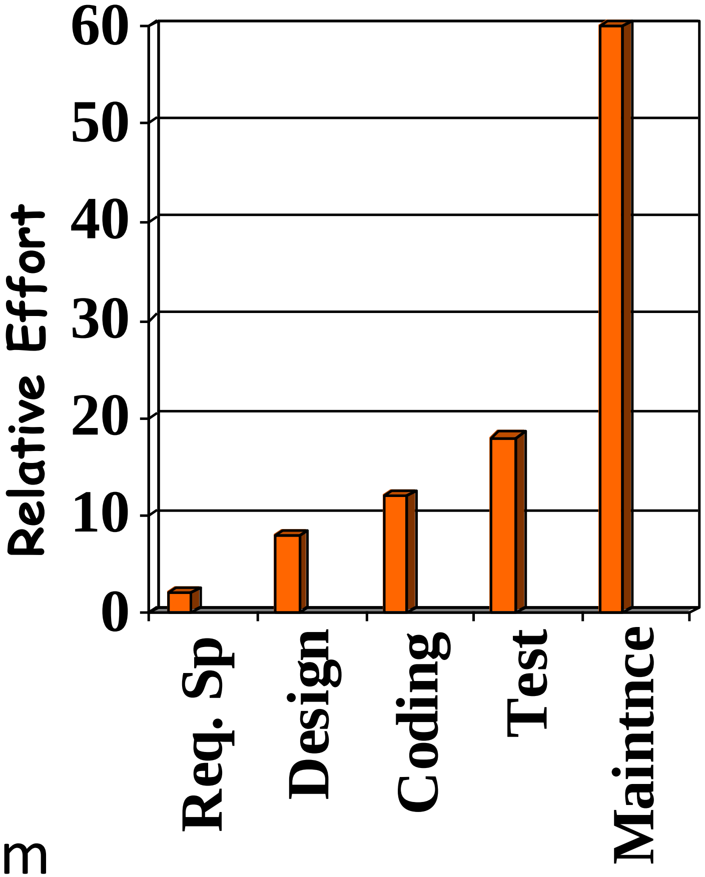
Classical Waterfall Model



Simplest and most intuitive

Relative Effort for Phases

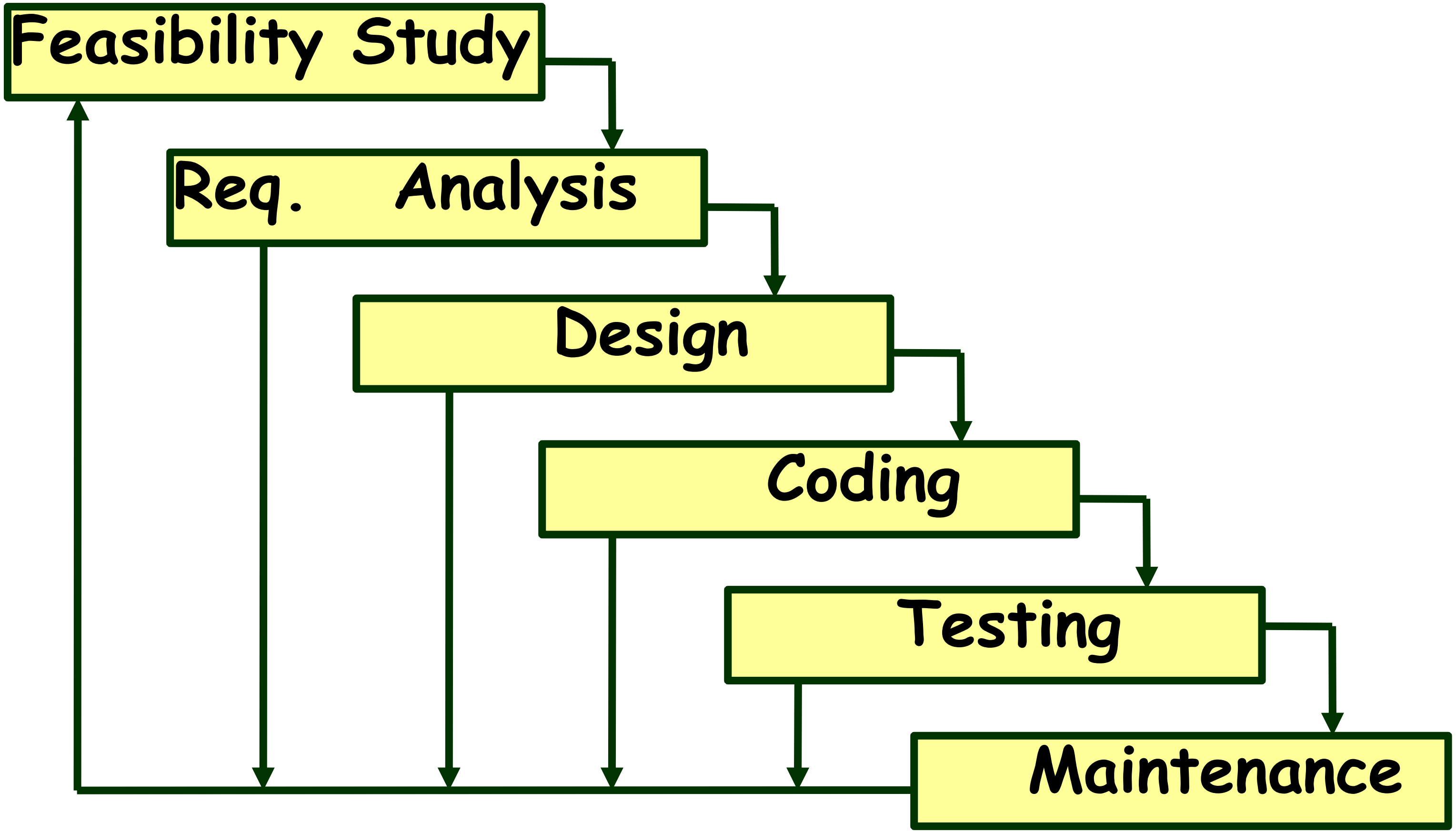
- Phases between feasibility study and testing Called development phases.
- Among all life cycle phases
  - Maintenance phase consumes maximum effort.
- Among development phases,
  - Testing phase consumes the maximum effort.



Need of an Iterative model

- Classical waterfall model is idealistic:
  - Assumes that no defect is introduced during any development activity.
  - In practice, defects do get introduced in almost every phase of the life cycle.
- Defects usually get detected much later in the life cycle:
  - **The later the phase in which the defect gets detected, the more expensive is its removal**
- Once a defect is detected:
  - The phase in which it occurred needs to be reworked.
  - Redo some of the work done during that and all subsequent phases.
- Therefore need feedback paths in the classical waterfall model.

Iterative Waterfall Model



Iterative waterfall model is by far the most widely used model.

Almost every other model is derived from the waterfall model.

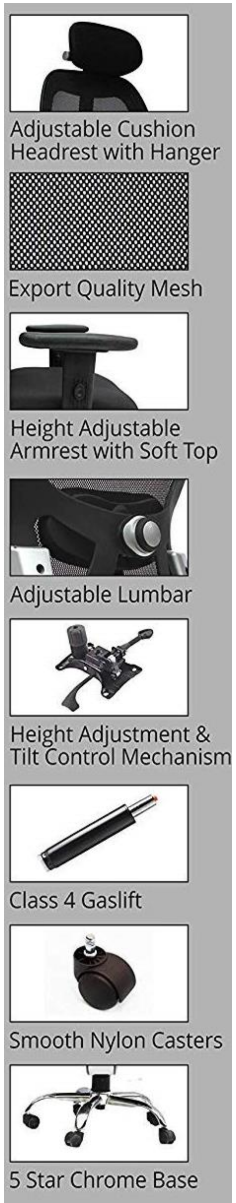
Phase Containment of Errors

- **Phase containment of errors:**
  - The principle of detecting errors as close to its point of introduction as possible.

- **Errors should be detected In the same phase in which they are introduced.**

- For example:
  - If a design problem is detected in the design phase itself,
    - The problem can be taken care of much more easily
    - Than say if it is identified at the end of the integration and system testing phase.

Waterfall model example


Adjustable Cushion Headrest with Hanger
Export Quality Mesh
Height Adjustable Armrest with Soft Top
Adjustable Lumbar
Height Adjustment & Tilt Control Mechanism
Class 4 Gaslift
Smooth Nylon Casters
5 Star Chrome Base

SDLC STEPS →

Image Ref: https://www.amazon.in/Chairs-Apollo-Chrome-Office-Adjustable/dp/B07D5F2NLB/

Waterfall Strengths

- Easy to understand, easy to use, especially by inexperienced staff
- Milestones are well understood by the team
- Provides requirements stability during development
- Facilitates strong management control (plan, staff, track)

Waterfall Deficiencies

- All requirements must be known upfront – in **most projects requirement change occurs after project start**
- Can give a false impression of progress
- Integration is one big bang at the end
- Little opportunity for customer to pre-view the system.

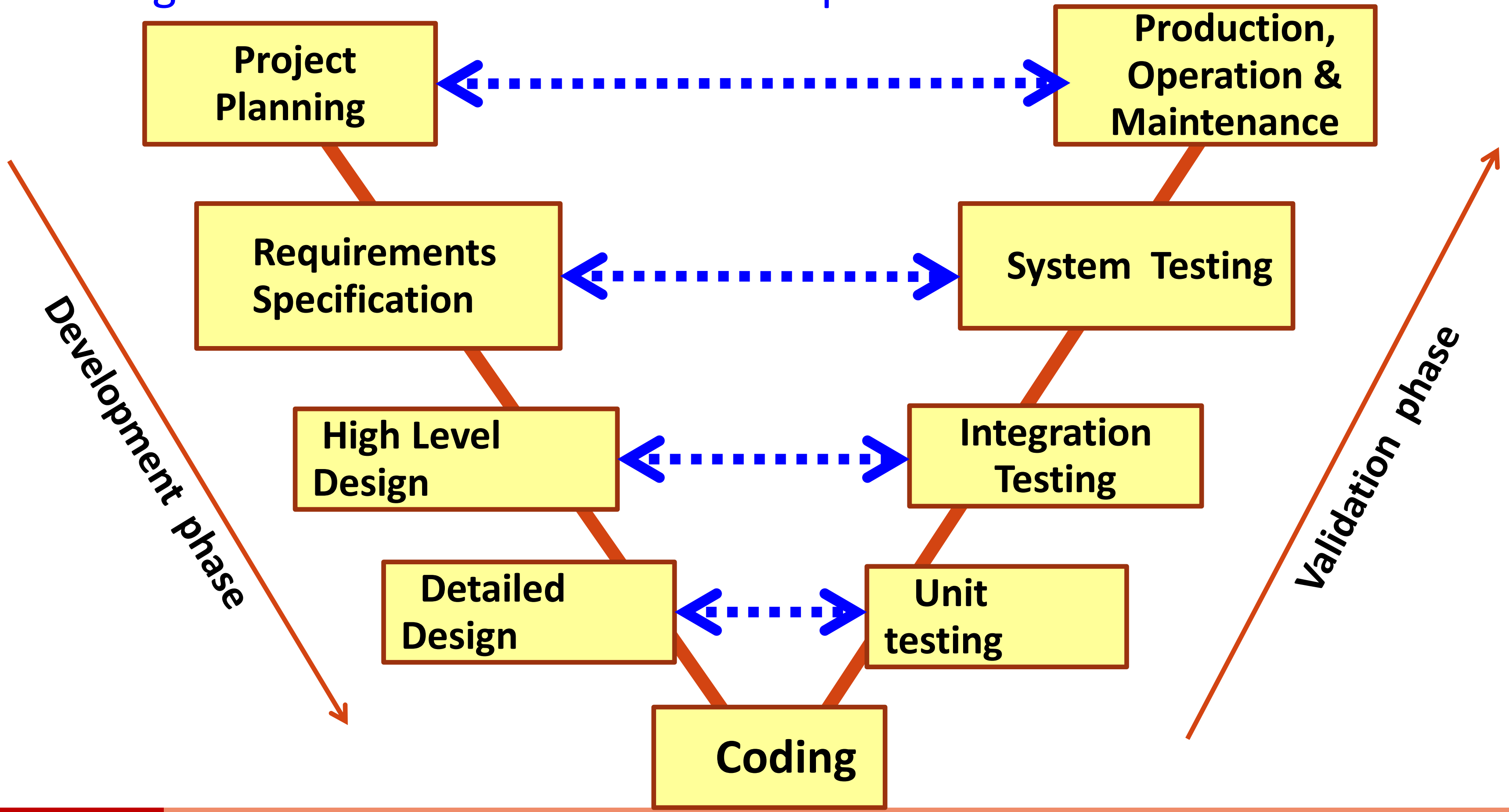
When to use the Waterfall Model?

- Requirements are well known and stable
- Technology is understood
- Development team have experience with similar projects

V Model

V Model

- Also known as the **V-Model or Verification and Validation model**
- A variant of the Waterfall: In every phase of development, testing activities are planned in parallel with development, whereas actual testing is carried out in the validation phase.



V Model: Strengths and weakness

Strengths:

- Easy to use
- Starting from early stages of software development:
  - **Emphasizes planning for verification and validation of the software**
- Each deliverable is made testable

Weaknesses

- Does not support overlapping of phases
- Does not handle iterations of phases
- Does not easily accommodate later changes to requirements
- Does not provide support for effective risk handling
- No early prototypes available

When to use V Model

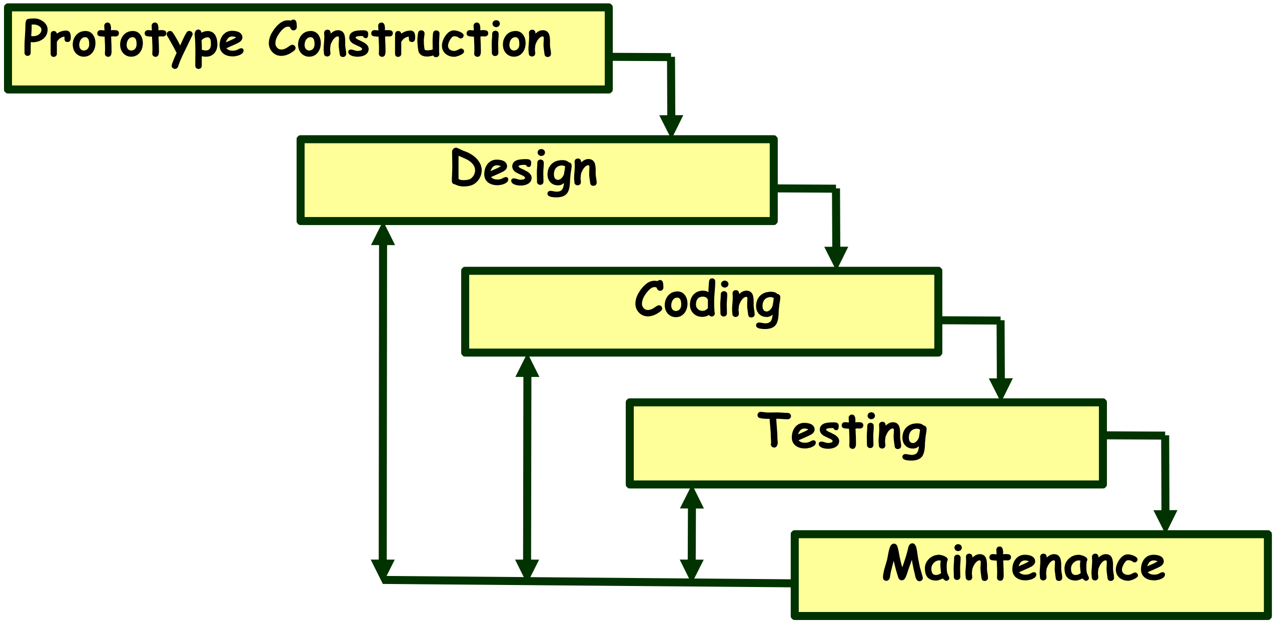
- Natural choice for systems requiring high reliability:
  - Embedded control applications, safety-critical software
- All requirements are known up-front
- Solution and technology are known

Prototyping Model

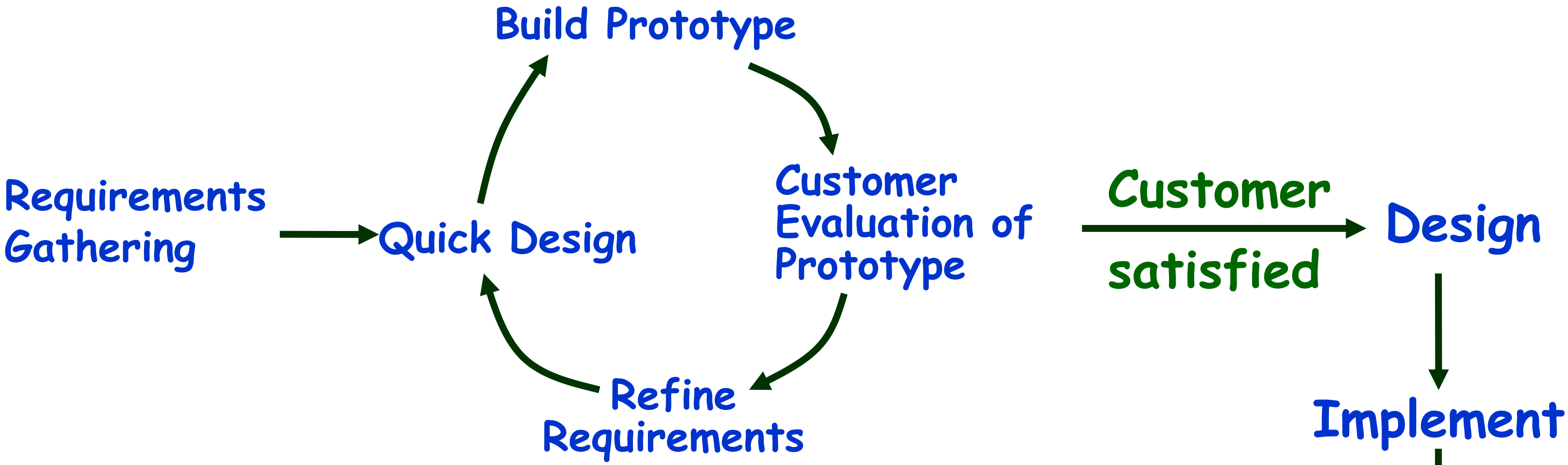
Prototyping Model

- Before starting actual development
  - A working prototype of the system should first be built.

- A prototype is a toy implementation of a system:
  - Limited functional capabilities
  - Low reliability
  - Inefficient performance

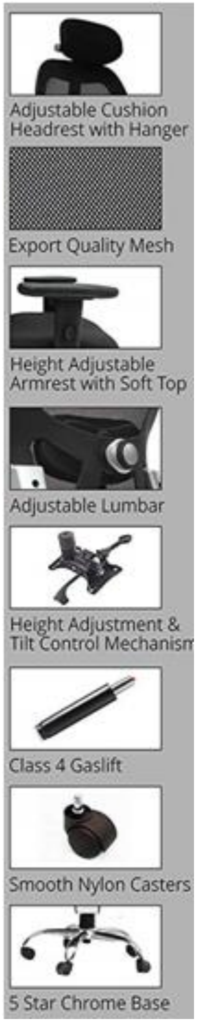


Prototyping Model



- Start with approximate requirements and carry out a quick design.
- The developed prototype is submitted to the customer for his evaluation
- Based on the user feedback, the prototype is refined
- This cycle continues until the user approves the prototype.
- The actual system is developed using the waterfall model.
- **Design and code for the prototype is usually thrown away**

Example:


Adjustable Cushion Headrest with Hanger
Export Quality Mesh
Height Adjustable Armrest with Soft Top
Adjustable Lumbar
Height Adjustment & Tilt Control Mechanism
Class 4 Gaslift
Smooth Nylon Casters
5 Star Chrome Base

Requirement Specification

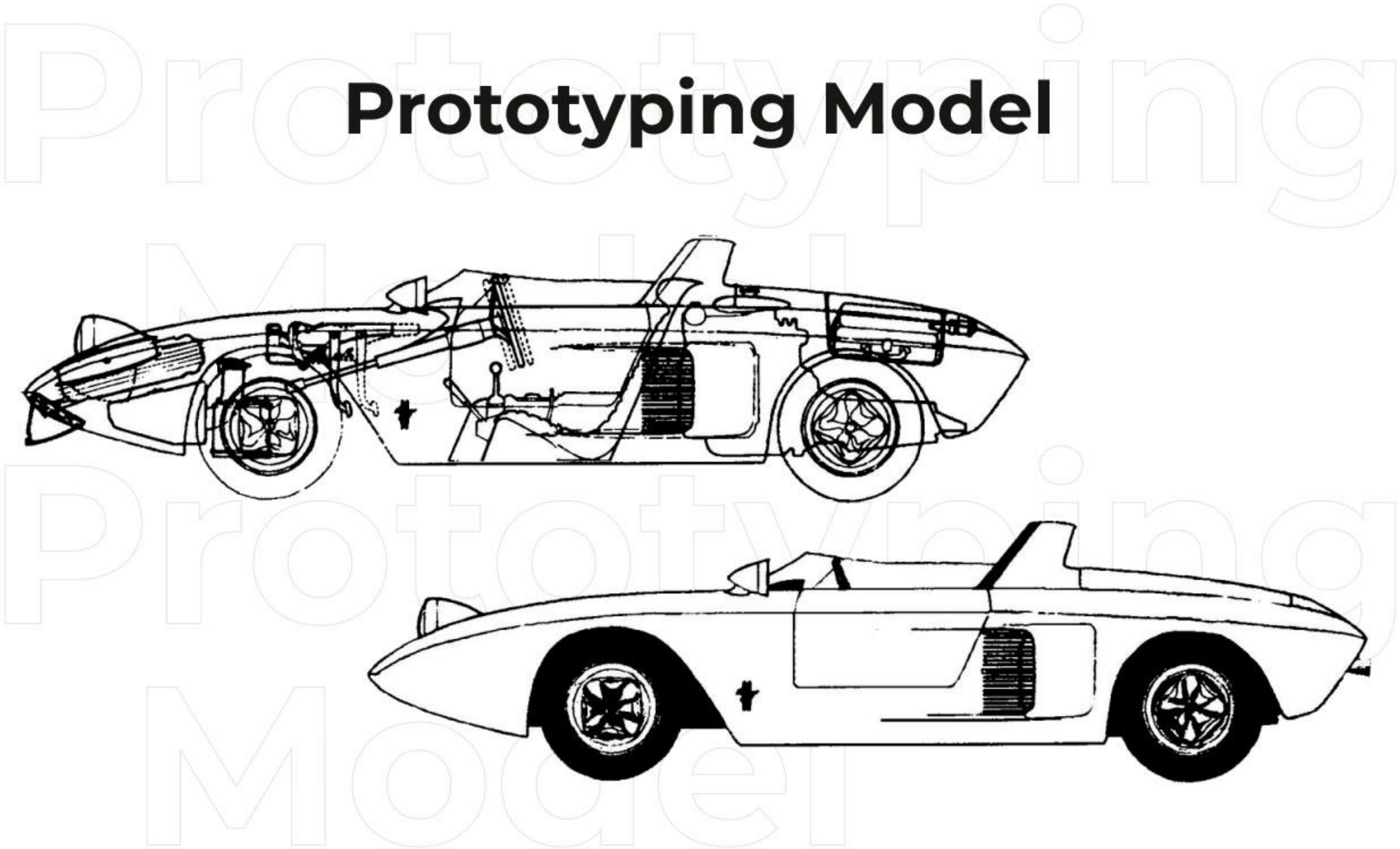
Prototype-1

Prototype-2

Prototype-3

SDLC →

Example:



Ref: https://themindstudios.com/blog/software-development-life-cycle-models/

Prototyping Model

- Even though construction of a working prototype model involves additional cost --- overall development cost usually lower for:
  - Systems with unclear user requirements.
  - Systems with unresolved technical issues.

- Another reason for developing a prototype:
  - It is impossible (difficult) to “get it right” the first time,

- Best for:
  - Using in parallel with any other SDLC model
  - Products with a lot of user interactions
  - Products that should be approved by users at early stages

Prototyping: advantages

- Learning by doing: useful where requirements are only partially known
- Improved user involvement and hence useful for the applications including GUI development
- Better understanding of functionality by users as they take part in the development process
- Reduced need for documentation
- Reduced maintenance costs
- The resulting software is usually more usable
- The design is of higher quality

Prototyping: disadvantages

- For some projects, it is expensive

- Susceptible to over-engineering:
  - Designers start to incorporate sophistications that they could not incorporate in the actual development.

Major difficulties of Waterfall-Based Models

1. Difficulty in accommodating change requests during development.
   - Capers Jones's research on 8000 projects: **40% of the requirements change during development**

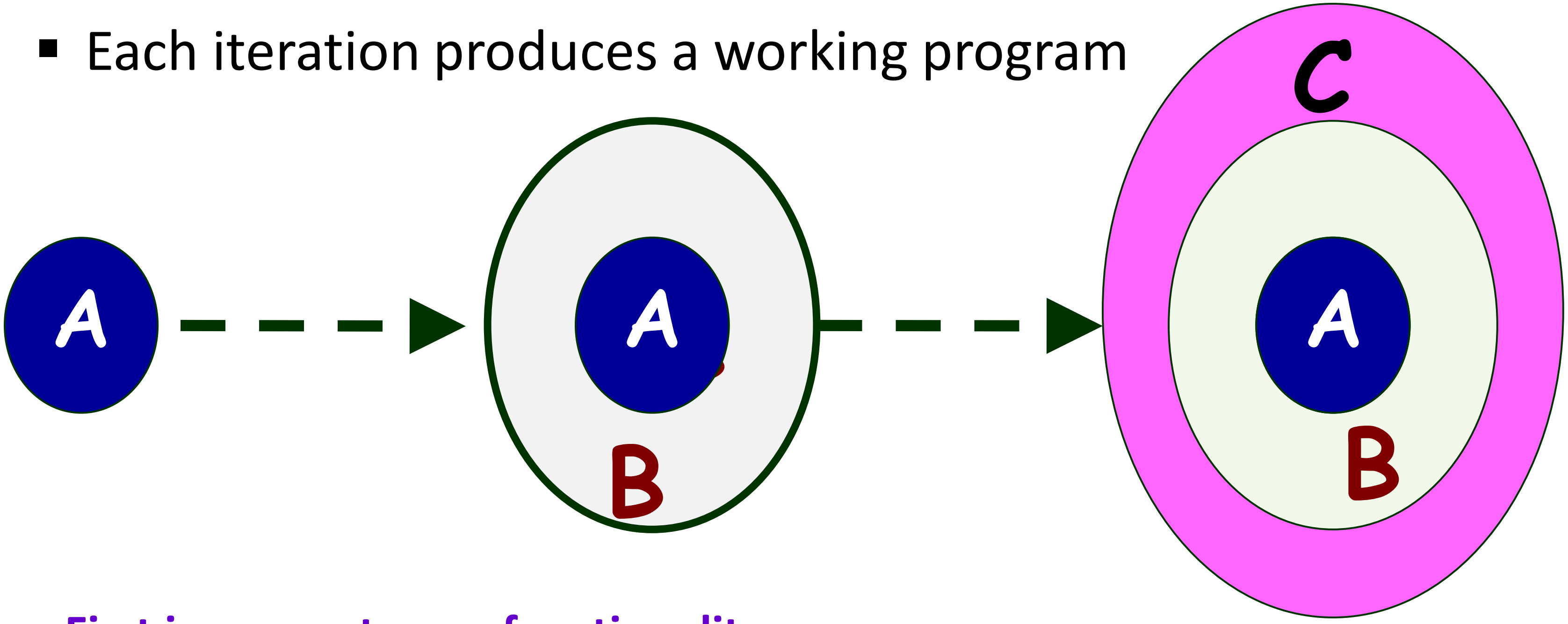
2. High cost incurred in developing custom applications.

3. Requirements for the system are determined at the start:
   - Are assumed to be fixed from that point on.
   - Long term planning is made based on this.

IID Model

Incremental and Iterative Development (IID)

- **Key characteristics**
  - Builds system incrementally
  - Consists of a planned number of iterations
  - Each iteration produces a working program

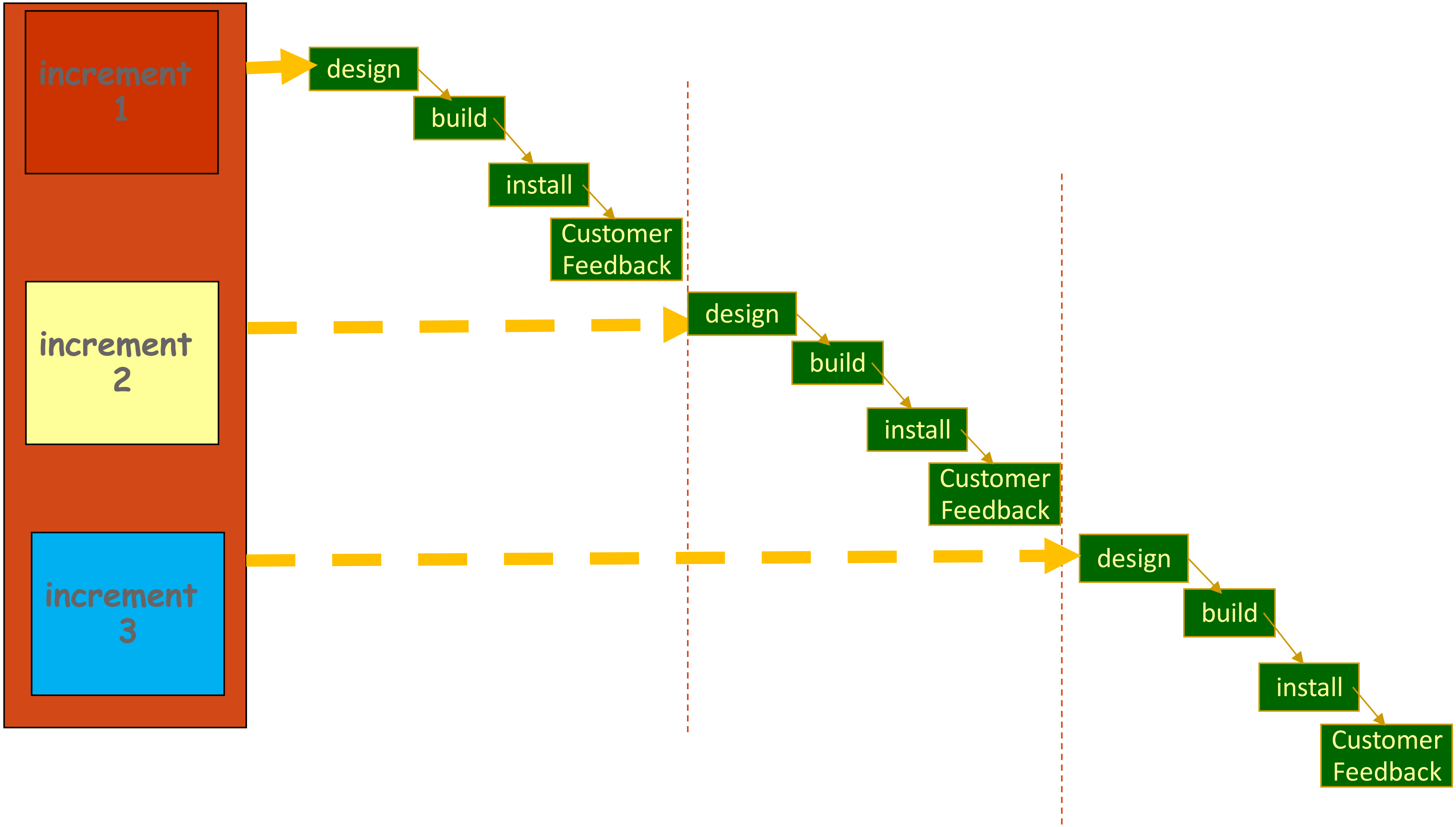


First increment: core functionality

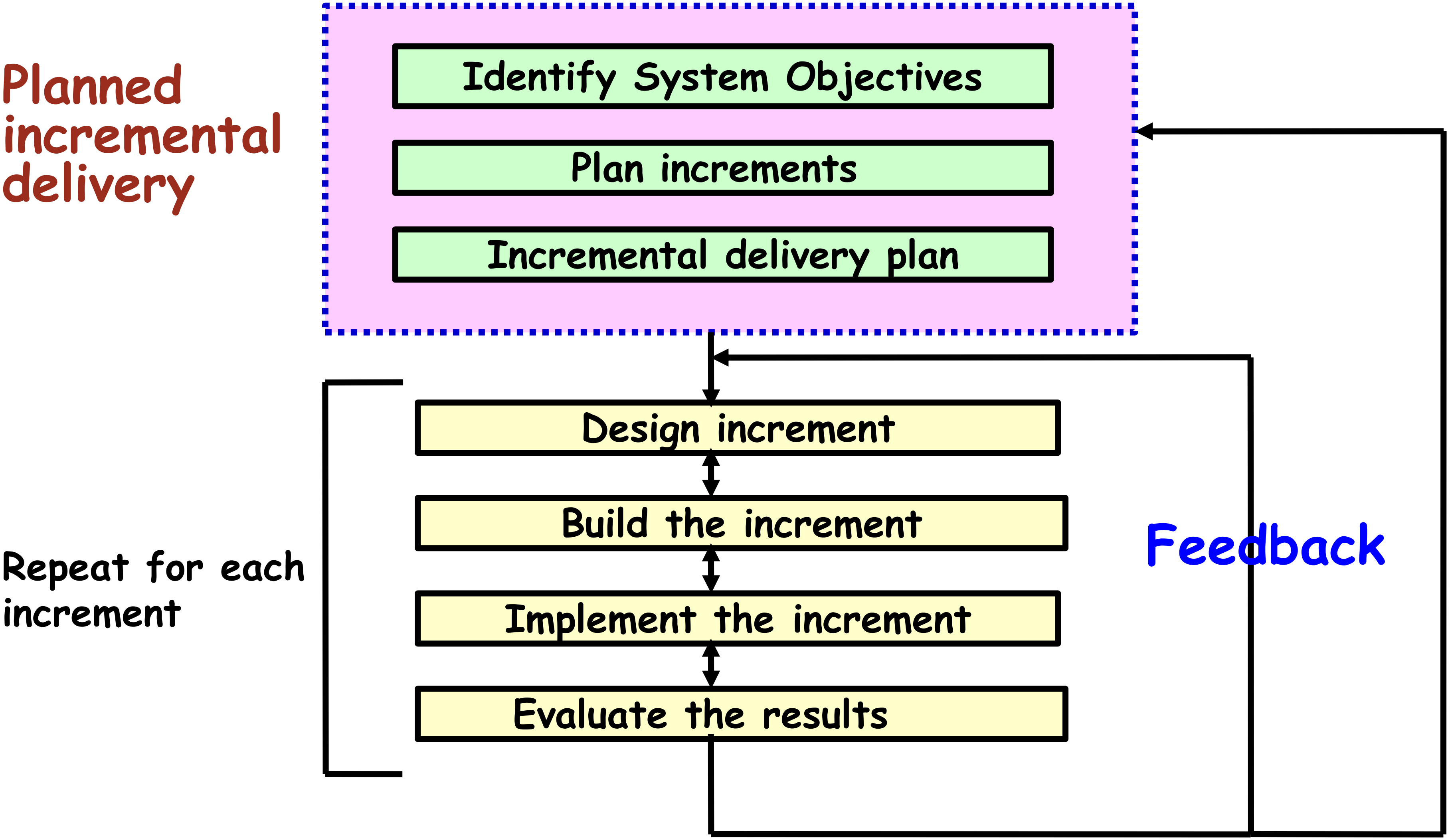
Successive increments: add/fix functionality

Final increment: the complete product

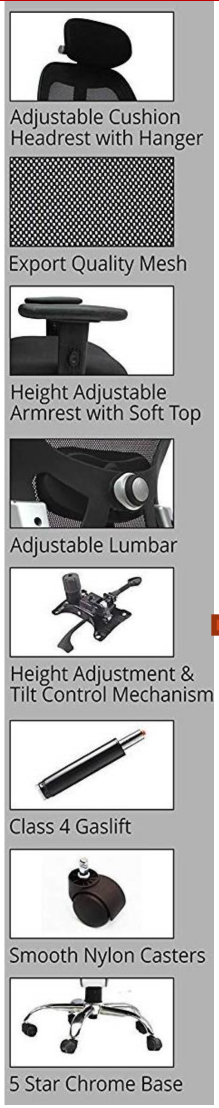
Incremental delivery



Incremental process


Planned incremental delivery
Identify System Objectives
Plan increments
Incremental delivery plan
Design increment
Build the increment
Implement the increment
Evaluate the results
Repeat for each increment
Feedback

Example


Adjustable Cushion Headrest with Hanger
Export Quality Mesh
Height Adjustable Armrest with Soft Top
Adjustable Lumbar
Height Adjustment & Tilt Control Mechanism
Class 4 Gaslift
Smooth Nylon Casters
5 Star Chrome Base

Requirement Specification

SDLC STEPS

SDLC STEPS

SDLC STEPS

Image Ref: https://www.amazon.in/Chairs-Apollo-Chrome-Office-Adjustable/dp/B07D5F2NLB/

Which step first?

- Some steps will be pre-requisite because of physical dependencies
- Others may be in any order
- Value to cost ratios may be used
  - V/C where
  - V is a score 1-10 representing value to customer
  - C is a score 0-10 representing cost to developers

V/C ratios: an example

step	value	cost	ratio	
profit reports	9	2	4.5	2nd
online database	1	9	0.11	5th
ad hoc enquiry	5	5	1	4th
purchasing plans	9	4	2.25	3rd
profit-based pay for managers	9	1	9	1st

IID model

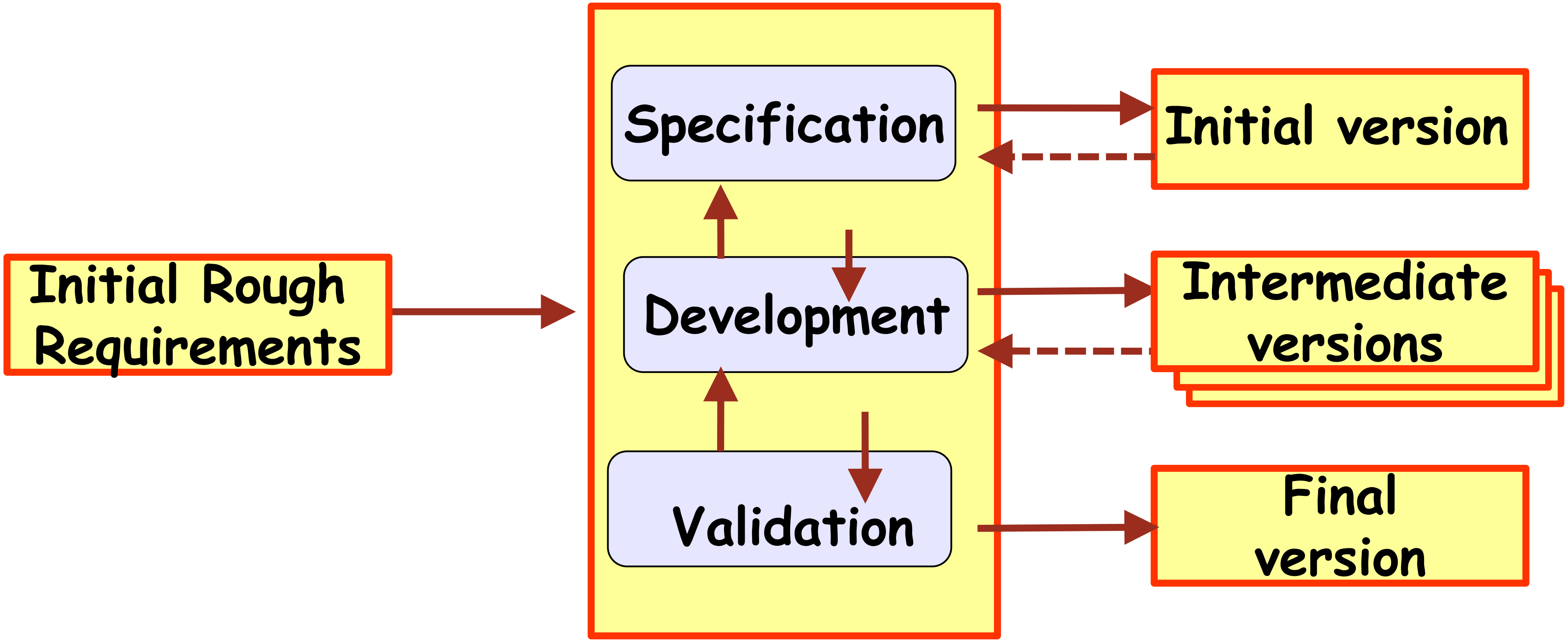
- **Best for:**
  - Complicated and mission-critical projects like ERP systems
  - Projects with strict requirements for the final product but with space for additional enhancements
  - Projects where major requirements are defined but some functionalities may evolve or enhancements may be made
  - Projects where the required technology is new and hasn’t been mastered yet or is only planned for some part of the product
  - Products with high-risk features that may need to be changed

Evolutionary Model

Evolutionary Model

“Plan a little, design a little, and code a little”

- First develop the core modules of the software.
- The initial skeletal software is refined into increasing levels of capability: (Iterations)
- Software developed over several “mini waterfalls”.



Evolutionary Model with Iteration

- Outcome of each iteration: tested, integrated, executable system

- Iteration length is short and fixed
  - **Usually between 2 and 6 weeks**
  - **Development takes many iterations (for example: 10-15)**

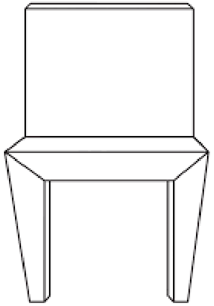
- Does not "freeze" requirements and then conservatively design :

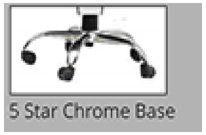
- Successive versions:
  - Functioning systems capable of performing some useful work.
  - A new release may include new functionality:
  - Also existing functionality in the current release might have been enhanced.

- **Multiple versions until the final version.**

Example

Requirement Specification





SDLC STEPS

SDLC STEPS

SDLC STEPS

SDLC STEPS

Image Ref: https://www.amazon.in/Chairs-Apollo-Chrome-Office-Adjustable/dp/B07D5F2NLB/

Advantages of Evolutionary Model

- Users get a chance to experiment with a partially developed system: Much before the full working version is released,

- **Helps finding exact user requirements:**

- Can get customer feedback and incorporate them much more efficiently

- **Core modules get tested thoroughly:**
  - Reduces chances of errors in final delivered software.

- Better management of complexity by developing one increment at a time.

Evolutionary Model: Problems

- **Feature division into incremental parts can be non-trivial**
- **The process is intangible:**
  - No regular, well-defined deliverables.
- **The process is unpredictable:**
  - Hard to manage, e.g., scheduling, workforce allocation, etc.
- **Systems are rather poorly structured:**
  - Continual, unpredictable changes tend to degrade the software structure.
- **Systems may not even converge to a final version.**
- **Requires heavy documentation**
- **Hard to predict the end of the project**

RAD Model

Rapid Application Development (RAD) Model

- Sometimes referred to as the **rapid prototyping model**.

- Major aims:
  - Decrease the time taken and the cost incurred to develop software systems.
  - Facilitate accommodating change requests as early as possible:
    - Before large investments have been made in development

Important Underlying Principle:

Make only short term plans and make heavy reuse of existing code.

Methodology

- Plans are made for one increment at a time.
  - The time planned for each iteration is called a **time box**.

- Each iteration (increment): Enhances the implemented functionality of the application a little.

- During each iteration,
  - A quick-and-dirty prototype-style software for some selected functionality is developed.
  - The customer evaluates the prototype and gives his feedback.
  - The prototype is refined based on the customer feedback.

How Does RAD Facilitate Faster Development?

- **Core modules get tested thoroughly:**
  - Reduces chances of errors in final delivered software.
  - Through use of specialized tools.

- These specialized tools usually support the following features:
  - **Visual style of development.**
  - **Use of reusable components.**
  - **Use of standard APIs (Application Program Interfaces).**

For which Applications is RAD Suitable?

- Customized product developed for one or two customers only
- Performance and reliability are not critical.
- Highly constrained project schedule
- The system can be split into several independent modules.

For Which Applications RAD is Unsuitable?

- Few plug-in components are available
- High performance or reliability required
- No precedence for similar products exists
- The system cannot be modularized.

Prototyping versus RAD

- In prototyping model:
  - The developed prototype is primarily used to gain insights into the solution
  - The developed prototype: Usually thrown away.

- **In RAD** the developed prototype evolves into deliverable software.

- RAD leads to faster development compared to traditional models:
  - However, the quality and reliability would possibly be poorer.

RAD versus Iterative Waterfall Model

- In the iterative waterfall model,
  - All product functionalities are developed together.
- In the RAD model on the other hand,
  - Product functionalities are developed incrementally through heavy code and design reuse.
  - Customer feedback is obtained on the developed prototype after each iteration:
    - Based on this the prototype is refined.
- Iterative waterfall model does have some important advantages:
  - Use of the iterative waterfall model leads to production of good documentation.
  - Also, the developed software usually has better quality and reliability than that developed using RAD.

RAD versus Evolutionary Model

- Incremental development:
  - Occurs in both evolutionary and RAD models.
- However, in RAD:
  - Each increment is a quick and dirty prototype,
  - Whereas in the evolutionary model each increment is systematically developed using the iterative waterfall model.
- Also, RAD develops software in shorter increments:
  - The incremental functionalities are fairly large in the evolutionary model.

Unified Process Model

Unified Process

- Developed **Ivar Jacobson, Grady Booch and James Rumbaugh**
- Rational Unified Process (RUP) is version tailored by Rational Software: Acquired by IBM in February 2003.
- Four phases
  - **Inception:** Scope of project is defined & prototypes may be developed about the project.
  - **Elaboration:** Functional & Non-Functional requirements are captured.
  - **Construction:** Analysis, Design & Implementation activities are carried out.
    - System features are implemented in iterations.
    - Each iteration results in executable release of the software.
  - **Transition:** product is installed in the user's environment and maintained.

Unified process work products

Inception phase

vision document
initial use-case model
initial business case
initial risk list
project plan
prototype(s)
...

Elaboration phase

use-case model
requirements
analysis model
preliminary model
revised risk list
preliminary manual
...

Construction phase

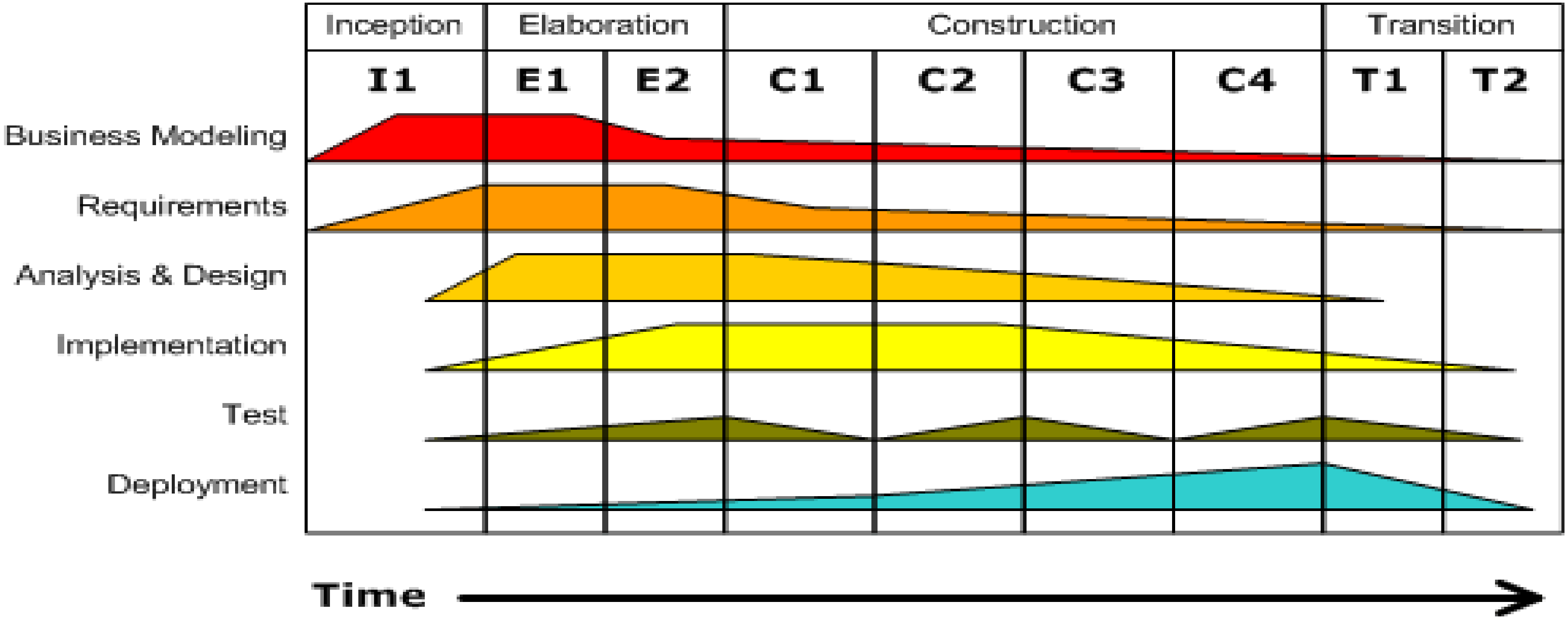
design model
SW components
test plan
test procedure
test cases
user manual
installation manual
...

Transition phase

SW increment
beta test reports
user feedback
...

Structure of RUP Process

- Two dimensions.
- Horizontal axis: Represents time and shows the lifecycle aspects of the process.
- Vertical axis: Represents core process workflows.



Spiral Model

Spiral Model

- Proposed by Boehm in 1988.

- Each loop of the spiral represents a phase of the software process:
  - the innermost loop might be concerned with system feasibility.
  - the next loop with system requirements definition.
  - the next one with system design, and so on.

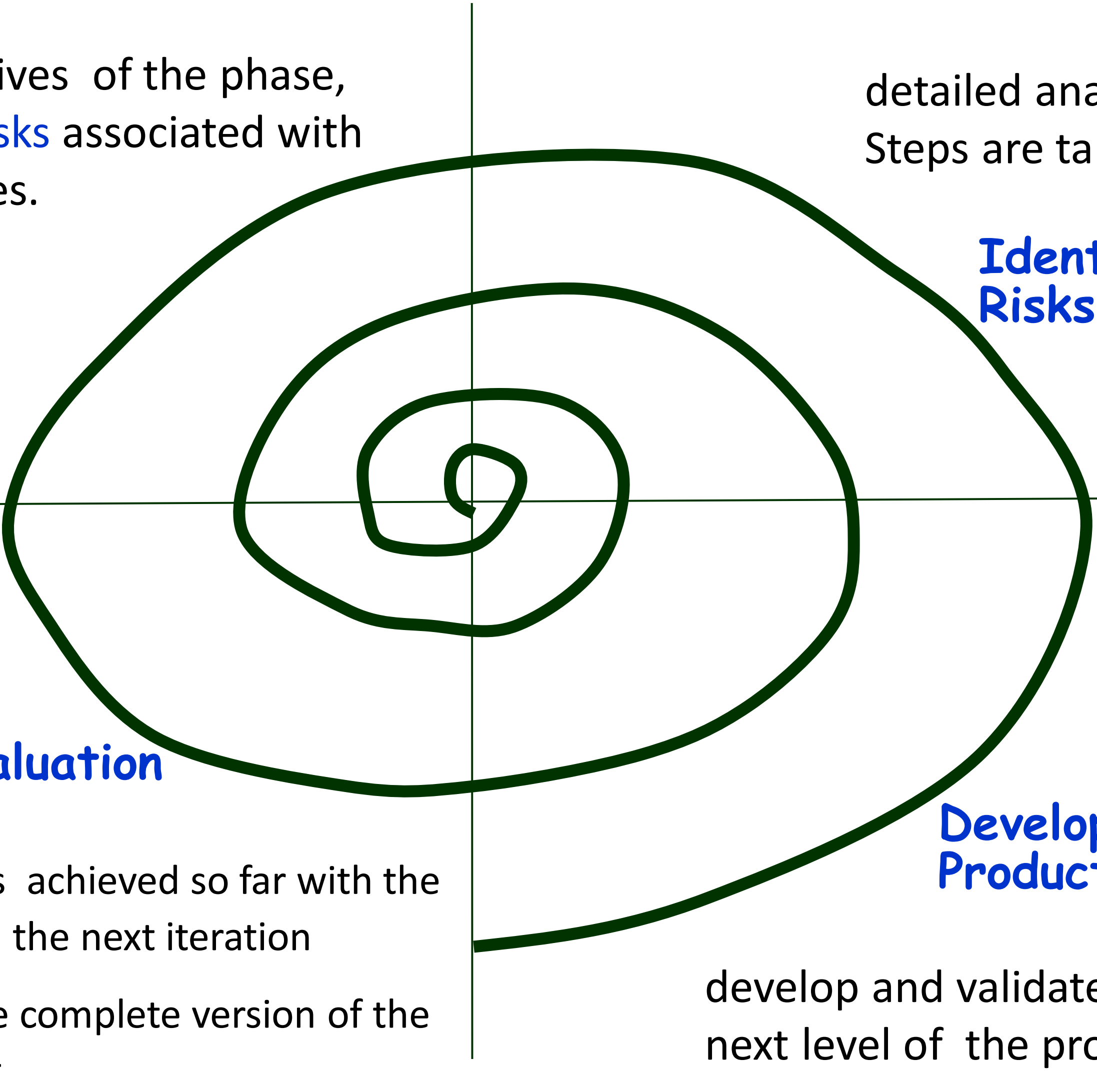
- There are no fixed phases in this model, the phases shown in the figure are just examples.

Spiral Model

- The team must decide:
  - how to structure the project into phases.
- Start work using some generic model:
  - add extra phases for specific projects or when problems are identified during a project.
- Each loop in the spiral is split into four sectors (quadrants).

Spiral Model

- Each loop in the spiral is split into four sectors (quadrants).



Identify objectives of the phase, Examine the risks associated with these objectives.

Determine Objectives

detailed analysis is carried out, Steps are taken to reduce the risk

Identify & Resolve Risks

Develop Next Level of Product

develop and validate the next level of the product.

Customer Evaluation of Prototype

Review the results achieved so far with the customer and plan the next iteration

progressively more complete version of the software gets built.

Spiral Model as a Meta Model

- Subsumes all discussed models:
  - A single loop spiral represents waterfall model.
  - Uses an evolutionary approach --
    - Iterations over the spiral are evolutionary levels.
  - Enables understanding and reacting to risks during each iteration along the spiral.
  - Uses:
    - prototyping as a risk reduction mechanism
    - retains the step-wise approach of the waterfall model.

Agile Models

What is Agile Software Development?

- Agile: Easily moved, light, nimble, active software processes

- How agility achieved?
  - Fitting the process to the project
  - Avoidance of things that waste time

  Proposed in mid-1990s

- The agile model was primarily designed:
  - To help projects to adapt to change requests

- In the agile model:
  - The requirements are decomposed into many small incremental parts that can be developed over one to four weeks each.

Ideology: Agile Manifesto

Individuals and interactions *over*

process and tools

Working Software *over*

comprehensive documentation

Customer collaboration *over*

contract negotiation

Responding to change *over*

following a plan

http://www.agilemanifesto.org

Agile Methodologies

- XP
- Scrum
- Unified process
- Crystal
- DSDM
- Lean

Agile Model: Principal Techniques

- **User stories:**
  - Simpler than use cases.
- **Metaphors:**
  - Based on user stories, developers propose a common vision of what is required.
- **Spike:**
  - Simple program to explore potential solutions.
- **Refactor:**
  - Restructure code without affecting behavior, improve efficiency, structure, etc.

Agile Software Development

- At a time, only one increment is planned, developed, deployed at the customer site.
  - **No long-term plans are made.**
- An iteration may not add significant functionality,
  - But still a new release is invariably made at the end of each iteration
  - Delivered to the customer for regular use.
- **Face-to-face communication favoured over written documents**

Agile Model: Principles

- The primary measure of progress:
  - **Incremental release of working software**

- Important principles behind agile model:
  - Frequent delivery of versions --- once every few weeks.
  - Requirements change requests are easily accommodated.
  - Close cooperation between customers and developers.
  - Face-to-face communication among team members.

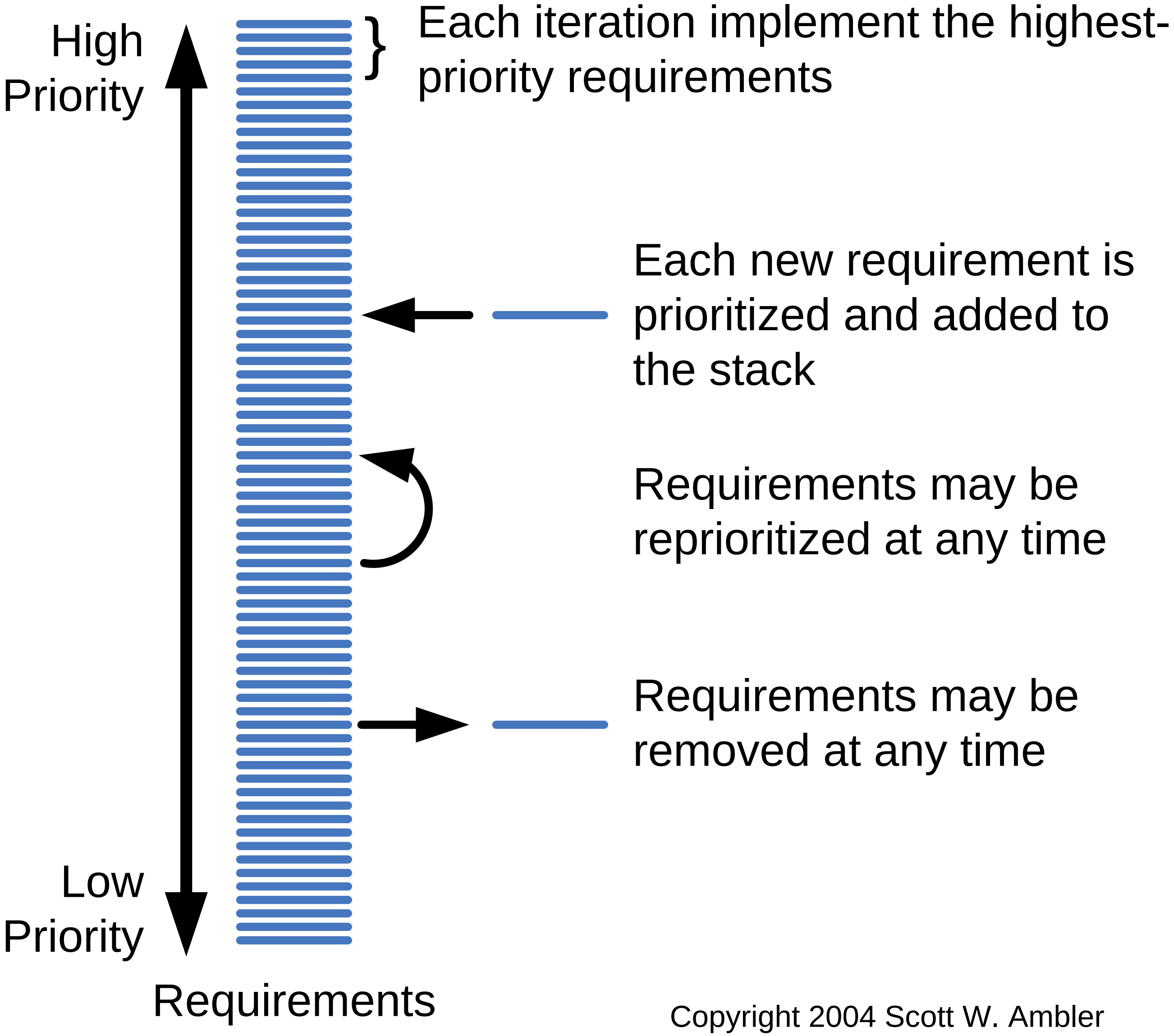
Agile Documentation

- Travel light:
  - You need far less documentation than you think.

- Agile documents:
  - Are concise
  - Describe information that is less likely to change
  - Describe “good things to know”
  - Are sufficiently accurate, consistent, and detailed

- Valid reasons to document:
  - Project stakeholders require it
  - To define a contract model
  - To support communication with an external group
  - To think something through

Agile Software Requirements Management



Each iteration implement the highest-priority requirements

Each new requirement is prioritized and added to the stack

Requirements may be reprioritized at any time

Requirements may be removed at any time

Agile Model Shortcomings

- Derives agility through developing tacit knowledge within the team, rather than any formal document:
  - Can be misinterpreted…
  - External review difficult to get…
  - When project is complete, and team disperses, maintenance becomes difficult…

Agile Model versus Waterfall Model

- Steps of Waterfall model are a planned sequence:
  - Requirements-capture, analysis, design, coding, and testing .

- Progress is measured in terms of delivered artefacts:
  - Requirement specifications, design documents, test plans, code reviews, etc.

- In contrast agile model sequences:
  - Delivery of working versions of a product in several increments.

Agile Model versus Iterative Waterfall Model

- As regards to similarity:
  - We can say that Agile teams use the waterfall model on a small scale.

Agile versus RAD Model

- Agile model does not recommend developing prototypes:
  - Systematic development of each incremental feature is emphasized.

- In contrast:
  - RAD is based on designing quick-and-dirty prototypes, which are then refined into production quality code.

Extreme Programming Model

Extreme Programming Model

- Extreme programming (XP) was proposed by Kent Beck in 1999.

- The methodology got its name from the fact that:
  - **Recommends taking the best practices to extreme levels.**

If something is good, why not do it all the time.

Taking Good Practices to Extreme

- **If code review is good:**
  - Always review --- **pair programming**
- **If testing is good:**
  - Continually write and execute test cases --- **test-driven development**
- **If incremental development is good:**
  - Come up with new increments every few days
- **If simplicity is good:**
  - Create the simplest design that will support only the currently required functionality.
- **If design is good,**
  - everybody will design daily (refactoring)
- **If architecture is important,**
  - everybody will work at defining and refining the architecture (metaphor)
- **If integration testing is important,**
  - build and integrate test several times a day (continuous integration)

4 Values

1. **Communication**:
   - Enhance communication among team members and with the customers.

2. **Simplicity**:
   - Build something simple that will work today rather than something that takes time and yet never used
   - May not pay attention for tomorrow

3. **Feedback**:
   - System staying out of users is trouble waiting to happen

4. **Courage**:
   - Don’t hesitate to discard code

Full List of XP Practices

1. **Planning** – determine scope of the next release by combining business priorities and technical estimates

2. **Small releases** – put a simple system into production, then release new versions in very short cycles

3. **Metaphor** – all development is guided by a simple shared story of how the whole system works

4. **Simple design** – system is to be designed as simple as possible

5. **Testing** – programmers continuously write and execute unit tests

Full List of XP Practices

7. **Refactoring** – programmers continuously restructure the system without changing its behavior to remove duplication and simplify

8. **Pair-programming** -- all production code is written with two programmers at one machine

9. **Collective ownership** – anyone can change any code anywhere in the system at any time.

10. **Continuous integration** – integrate and build the system many times a day – every time a task is completed.

Full List of XP Practices

11. **40-hour week** – work no more than 40 hours a week as a rule

12. **On-site customer** – a user is a part of the team and available full-time to answer questions

13. **Coding standards** – programmers write all code in accordance with rules emphasizing communication through the code

Emphasizes Test-Driven Development (TDD)

- Based on user story develop test cases
- Implement a quick and dirty feature every couple of days:
  - **Get customer feedback**
  - **Alter if necessary**
  - **Refactor**
- Take up next feature

Suitability of Extreme Programming

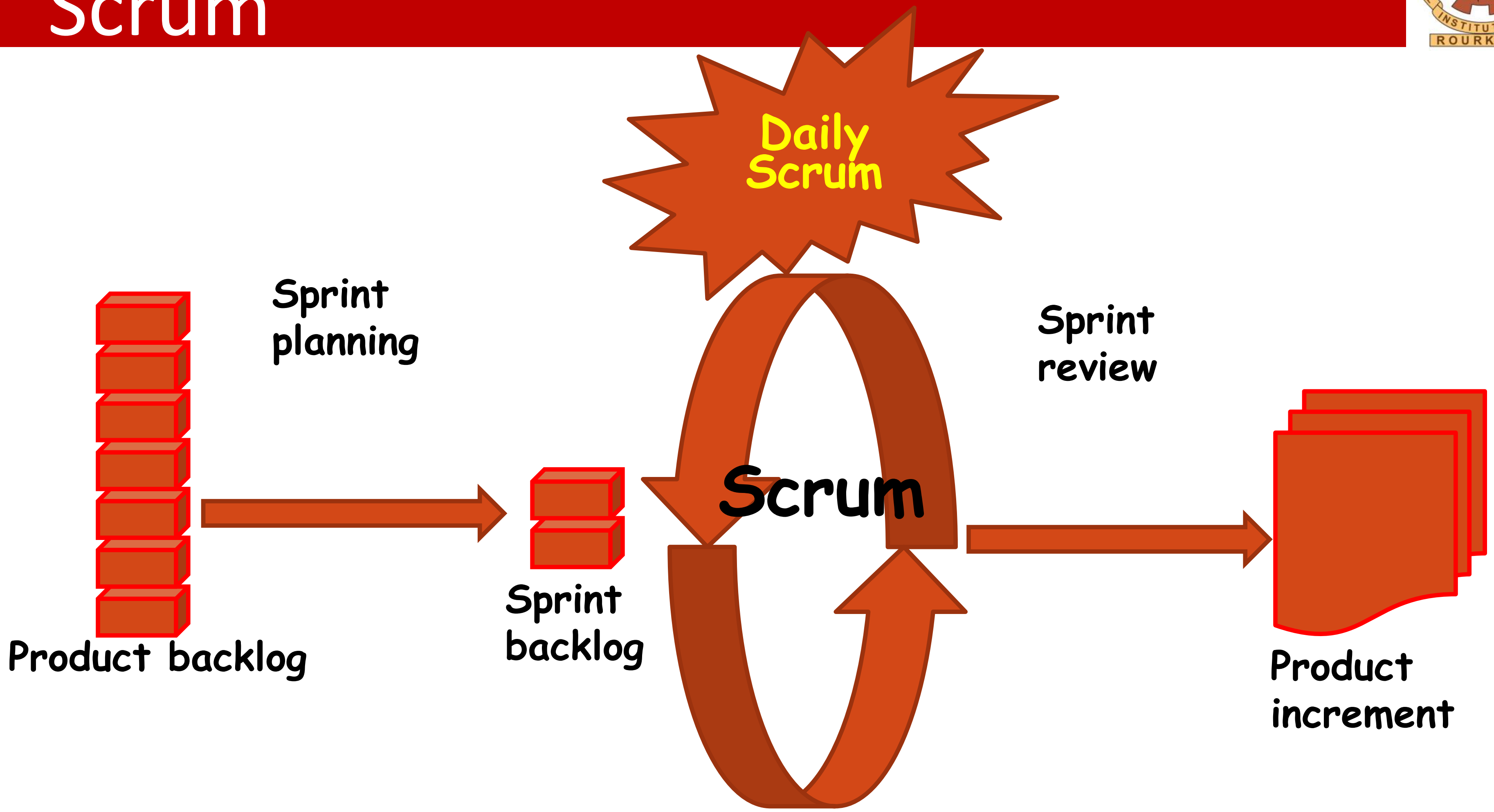
- Projects involving new technology or research projects.
  - In this case, the requirements change rapidly and unforeseen technical problems need to be resolved.
- Small projects:
  - These are easily developed using extreme programming.

Scrum Model

Scrum: Characteristics

- Self-organizing teams
- Product progresses in a series of month-long sprints
- Requirements are captured as items in a list of product backlog
- One of the agile processes

Scrum


Daily Scrum
Sprint planning
Product backlog
Sprint backlog
Scrum
Sprint review
Product increment

Sprint

- Scrum projects progress in a series of “sprints”
  - Analogous to XP iterations or time boxes
  - Target duration is one month

- Software increment is designed, coded, and tested during the sprint

- No changes entertained during a sprint

Scrum Framework

- **Roles :** Product Owner, Scrum Master, Team
- **Ceremonies :** Sprint Planning, Sprint Review, Sprint Retrospective, and Daily Scrum Meeting
- **Artifacts :** Product Backlog, Sprint Backlog, and Burndown Chart

Key Roles and Responsibilities in a Scrum Team

- **Product Owner**
  - Represents customers’ views and interests.
- **Development Team**
  - Team of five-nine people with cross-functional skill sets.
- **Scrum Master (aka Project Manager)**
  - Facilitates scrum process and resolves impediments at the team and organization level by acting as a buffer between the team and outside interference.

Product Owner

- Defines the features of the product
- Decides on release date and content
- Prioritizes features according to usefullness
- Adjusts features and priority every iteration, as needed
- Accepts or reject work results.

The Scrum Master

- Represents management in the project
- Removes impediments
- Ensures that the team is fully functional and productive
- Enables close cooperation across all roles and functions
- Shields the team from external interferences

Scrum Team

- Typically 5-10 people
- Cross-functional
  - QA, Programmers, UI Designers, etc.
- Teams are self-organizing
- Membership can change only between sprints

Sprint

Fundamental process flow of Scrum

- It is usually a month-long iteration:
  - during this time an incremental product functionality completed
- NO outside influence allowed to interfere with the Scrum team during the Sprint
- Each day begins with the Daily Scrum Meeting

Ceremonies

- Sprint Planning Meeting
- Daily Scrum
- Sprint Review Meeting

Sprint Planning

- Goal is to produce Sprint Backlog
- Product owner works with the Team to negotiate what Backlog Items
- Scrum Master ensures Team agrees to realistic goals

Daily Scrum

- Daily
- 15-minutes
- Stand-up meeting
- Not for problem solving
- Three questions:
  1. What did you do yesterday
  2. What will you do today?
  3. What obstacles are in your way?

Daily Scrum

- Is NOT a problem solving session
- Is NOT a way to collect information about WHO is behind the schedule
- Is a meeting in which team members review what is done and make informal commitments to each other and to the Scrum Master
- Is a good way for a Scrum Master to track the progress of the Team

Sprint Review Meeting

- Team presents what it accomplished during the sprint
- Typically takes the form of a demo of new features
- Informal
  - 2-hour prep time rule
- Participants
  - Customers
  - Management
  - Product Owner
  - Other team members

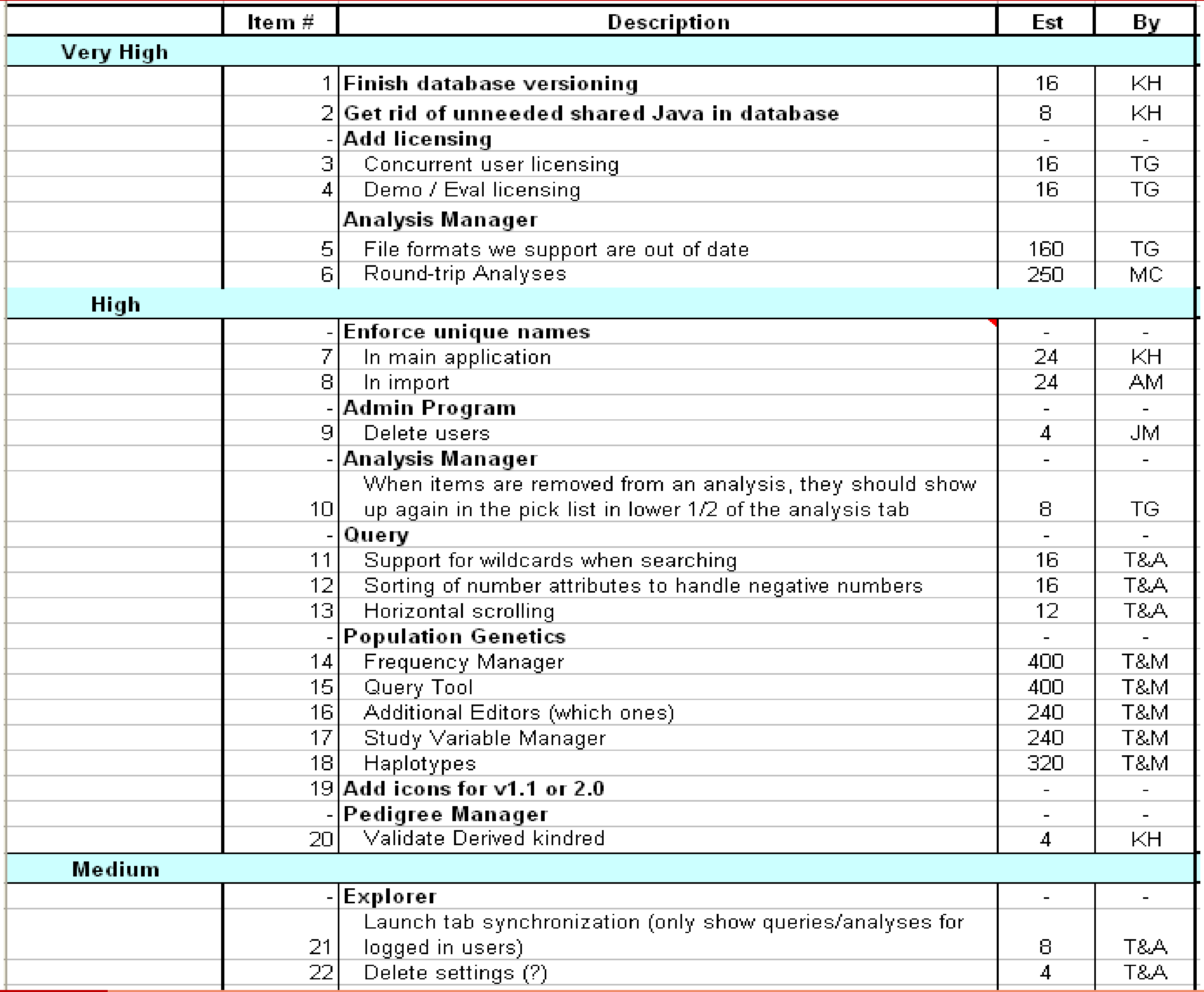
Product Backlog

- A list of all desired work on the project
  - Usually a combination of
    - story-based work (“let user search and replace”)
    - task-based work (“improve exception handling”)
- List is prioritized by the Product Owner
  - Typically a Product Manager, Marketing, Internal Customer, etc

Product Backlog

- Requirements for a system, expressed as a prioritized list of Backlog Items
  - **Managed and owned by Product Owner**
  - **Spreadsheet (typically)**

Sample Product Backlog



	Item #	Description	Est	By
Very High				
	1	**Finish database versioning**	16	KH
	2	**Get rid of unneeded shared Java in database**	8	KH
	-	**Add licensing**	-	-
	3	Concurrent user licensing	16	TG
	4	Demo / Eval licensing	16	TG
		Analysis Manager		
	5	File formats we support are out of date	160	TG
	6	Round-trip Analyses	250	MC
High				
	-	**Enforce unique names**	-	-
	7	In main application	24	KH
	8	In import	24	AM
	-	**Admin Program**	-	-
	9	Delete users	4	JM
	-	**Analysis Manager**	-	-
	10	When items are removed from an analysis, they should show up again in the pick list in lower 1/2 of the analysis tab	8	TG
	-	**Query**	-	-
	11	Support for wildcards when searching	16	T&A
	12	Sorting of number attributes to handle negative numbers	16	T&A
	13	Horizontal scrolling	12	T&A
	-	**Population Genetics**	-	-
	14	Frequency Manager	400	T&M
	15	Query Tool	400	T&M
	16	Additional Editors (which ones)	240	T&M
	17	Study Variable Manager	240	T&M
	18	Haplotypes	320	T&M
	19	**Add icons for v1.1 or 2.0**	-	-
	-	**Pedigree Manager**	-	-
	20	Validate Derived kindred	4	KH
Medium				
	-	**Explorer**	-	-
	21	Launch tab synchronization (only show queries/analyses for logged in users)	8	T&A
	22	Delete settings (?)	4	T&A

Sprint Backlog

- A subset of Product Backlog Items, which define the work for a Sprint
  - **Created by Team members**
  - **Each Item has it’s own status**
  - **Updated daily**

Sprint Backlog during the Sprint

- Changes occur:
  - Team adds new tasks whenever they need to in order to meet the Sprint Goal
  - Team can remove unnecessary tasks
  - But: Sprint Backlog can only be updated by the team
- Estimates are updated whenever there's new information

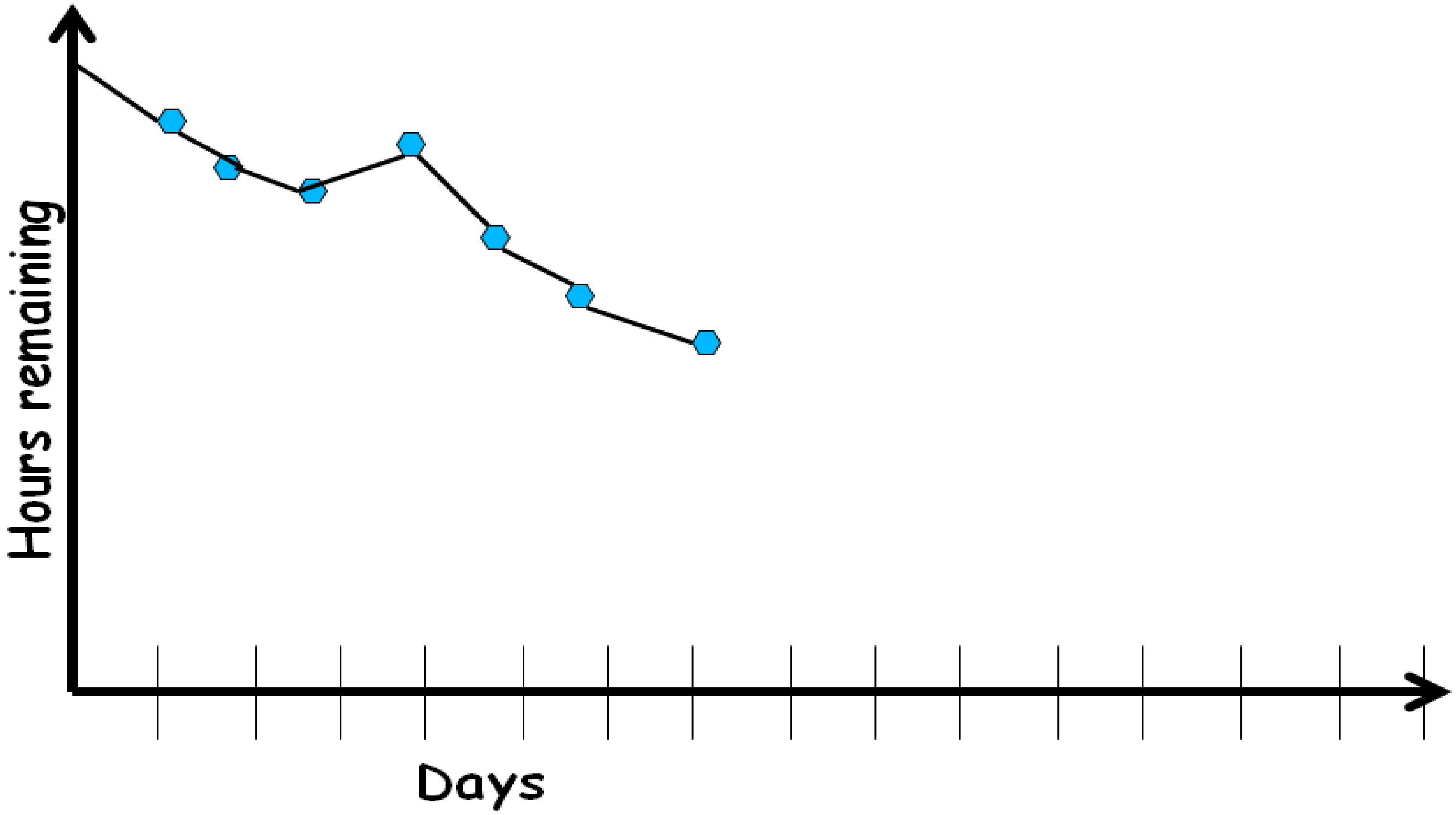
Burn down Charts

- Are used to represent "work done".
- Are remarkably simple but effective Information disseminators
- 3 Types:
  - **Sprint Burn down Chart (progress of the Sprint)**
  - **Release Burn down Chart (progress of release)**
  - **Product Burn down chart (progress of the Product)**

Sprint Burn down Chart

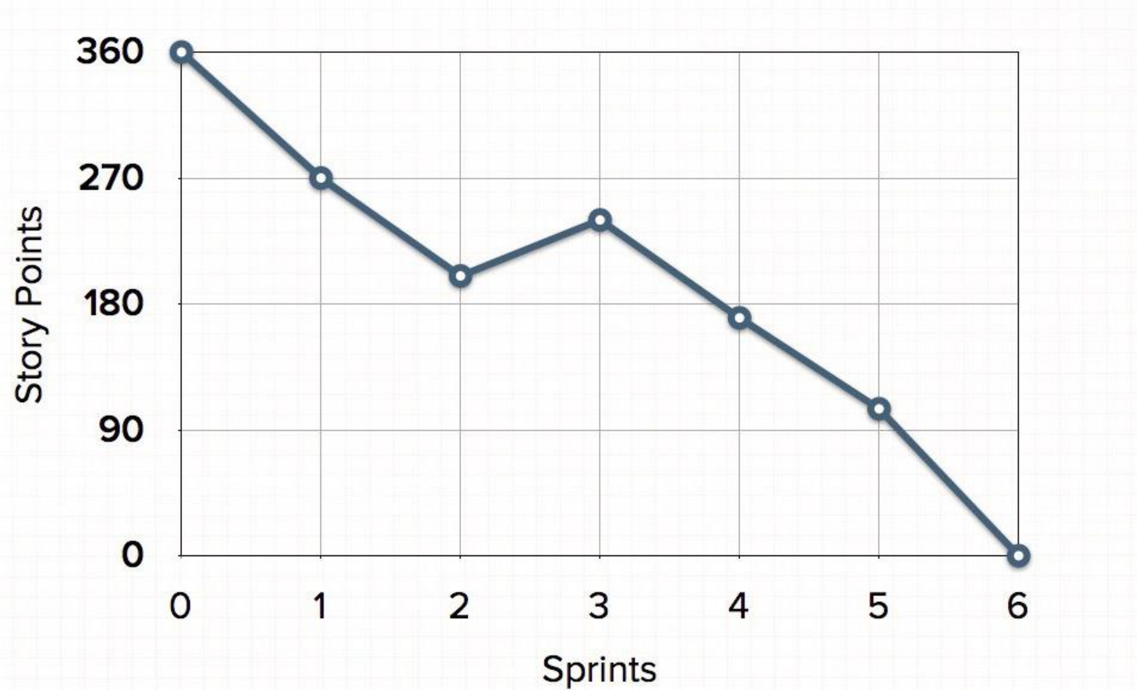
- Depicts the total Sprint Backlog hours remaining per day
- Shows the estimated amount of time to complete
- Ideally should burn down to zero to the end of the Sprint
- Actually is not a straight line

Sprint Burndown Chart



Release Burndown Chart

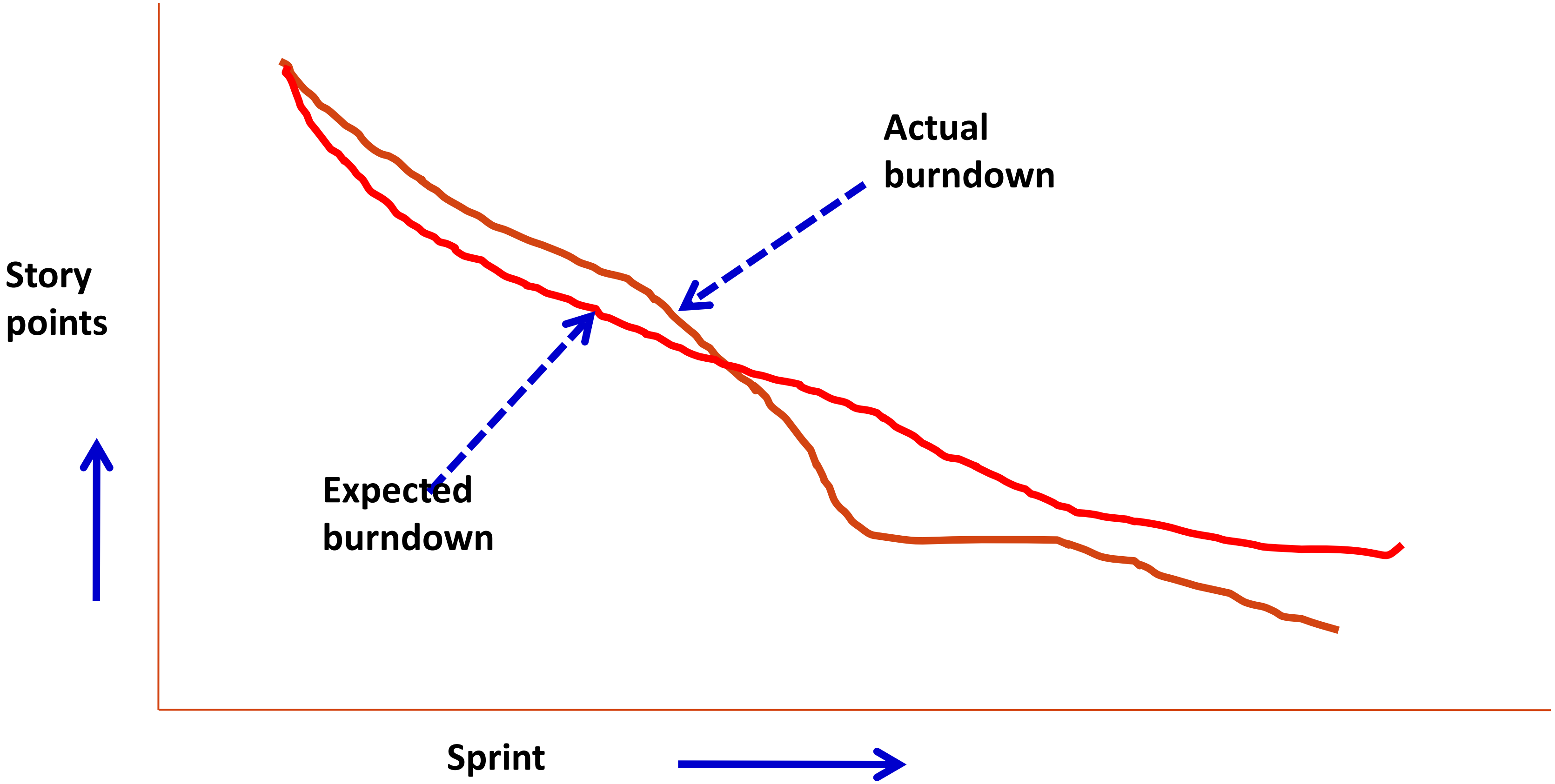
- Will the release be done on right time?
- How many more sprints?



- X-axis: sprints
- Y-axis: amount of story points remaining

Product Burndown Chart

- Is a "big picture" view of project's progress (all the releases)

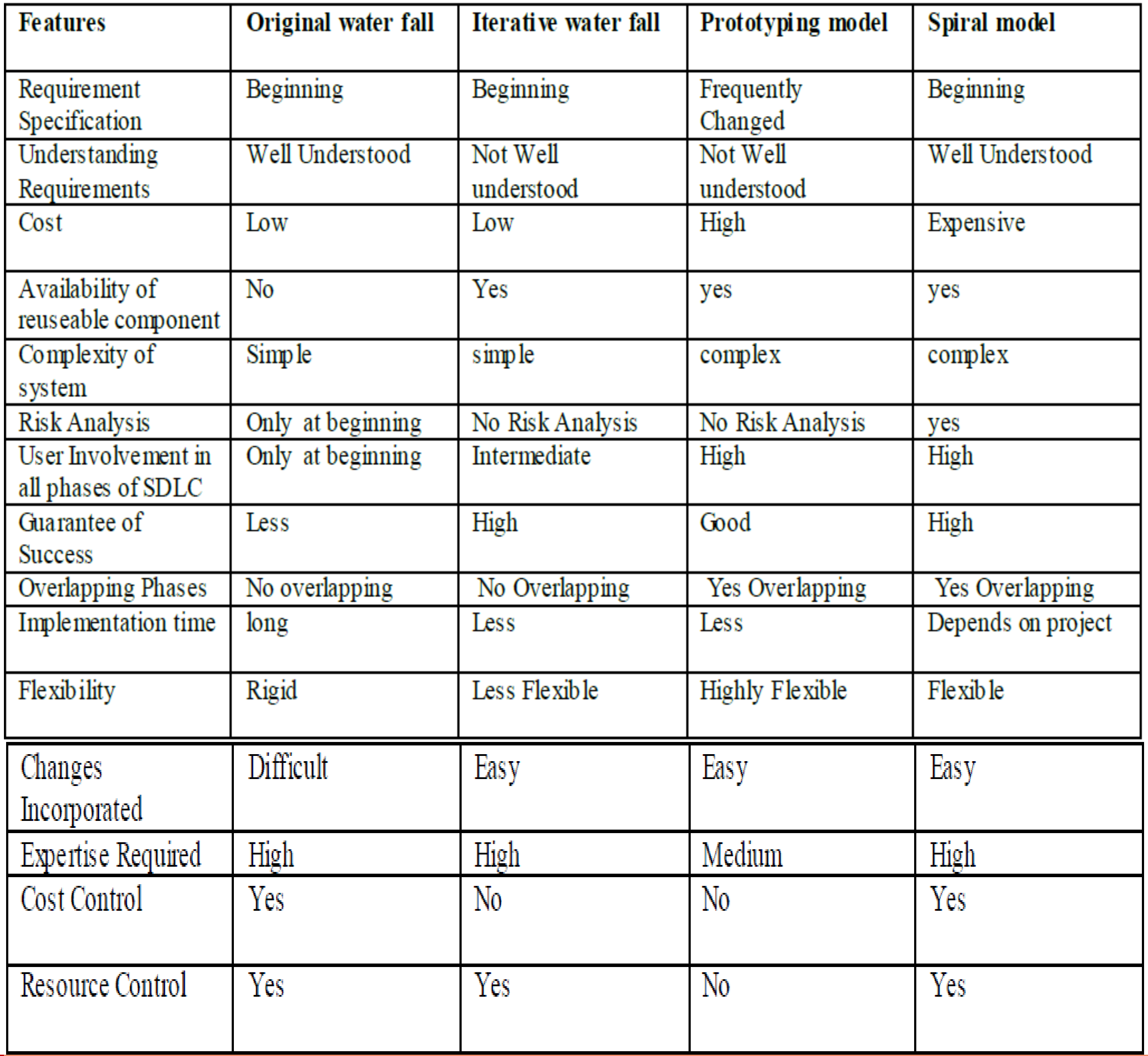


Scalability of Scrum

- A typical Scrum team is 6-10 people
- Jeff Sutherland - up to over 800 people
- "Scrum of Scrums" or "Meta-Scrum“

- End of Chapter

Thanks



Features	Original water fall	Iterative water fall	Prototyping model	Spiral model
Requirement Specification	Beginning	Beginning	Frequently Changed	Beginning
Understanding Requirements	Well Understood	Not Well understood	Not Well understood	Well Understood
Cost	Low	Low	High	Expensive
Availability of reuseable component	No	Yes	yes	yes
Complexity of system	Simple	simple	complex	complex
Risk Analysis	Only at beginning	No Risk Analysis	No Risk Analysis	yes
User Involvement in all phases of SDLC	Only at beginning	Intermediate	High	High
Guarantee of Success	Less	High	Good	High
Overlapping Phases	No overlapping	No Overlapping	Yes Overlapping	Yes Overlapping
Implementation time	long	Less	Less	Depends on project
Flexibility	Rigid	Less Flexible	Highly Flexible	Flexible
Changes Incorporated	Difficult	Easy	Easy	Easy
Expertise Required	High	High	Medium	High
Cost Control	Yes	No	No	Yes
Resource Control	Yes	Yes	No	Yes