

# **SWE 215: Software Requirements Engineering**

## **Requirements Engineering Methods**

**Dr. Jameleddine Hassine**

ICS Department, KFUPM

[jhassine@kfupm.edu.sa](mailto:jhassine@kfupm.edu.sa)

# Objectives

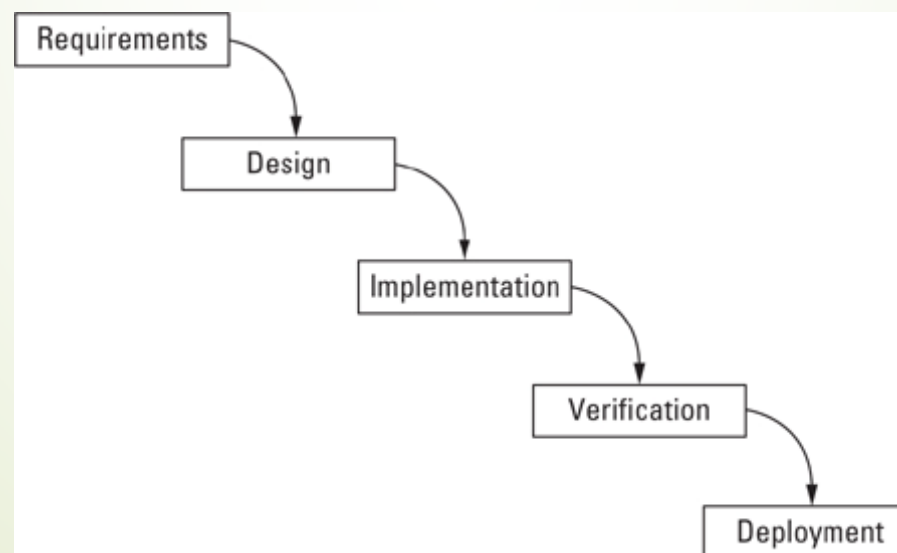
- Traditional requirements methods
- Understand the difference between traditional requirements methods and agile requirements methods

# Outline

- Predictive, Waterfall-like Process
- Problems with the Waterfall model
- Requirements in iterative processes
  - Spiral model
  - Rapid Application Development (RAD)
  - Rational Unified Process (RUP)
- Agile methods
- Requirements management in Agile processes
- Agile optimizes ROI through Incremental Value Delivery

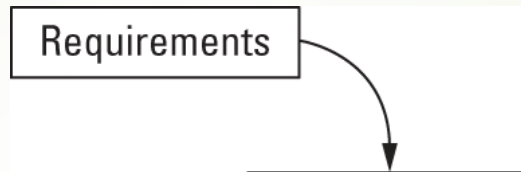
# Predictive, Waterfall-Like Process

- Software development occurred in an orderly series of sequential stages (Progress flows **top to bottom**, like a **waterfall**)
- Requirements were agreed to, a design was created, and code followed thereafter. Lastly, the software was tested to verify its conformance to its requirements and design

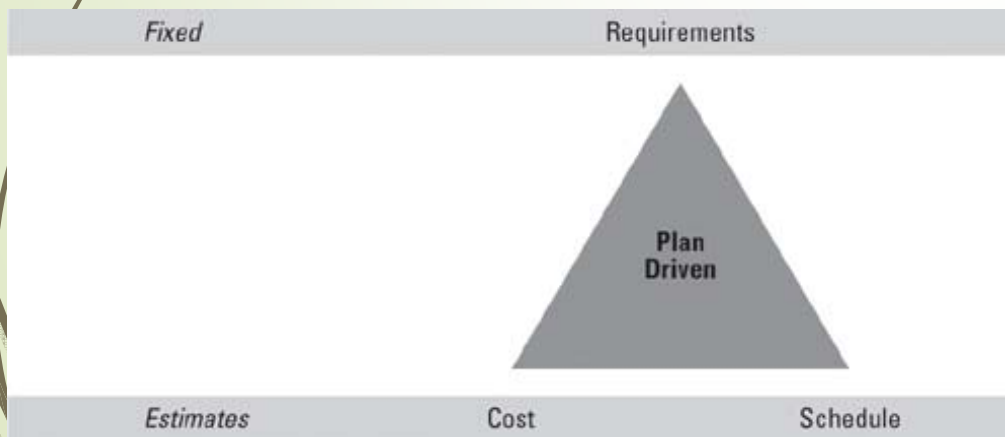


# Requirements in the waterfall model: the Iron Triangle (1)

- The “requirements box” implied that:



- There is a set of requirements that can be reasonably **determined “up front”**
- These requirements can be used as a basis to estimate the **schedule** and **budget** of the project.



**More realistically:  
Fixed resources  
and cost**

## Requirements in the waterfall model: the Iron Triangle (2)

- This “**fixed requirements scope**” assumption has indeed been found to be a root cause of project failures.

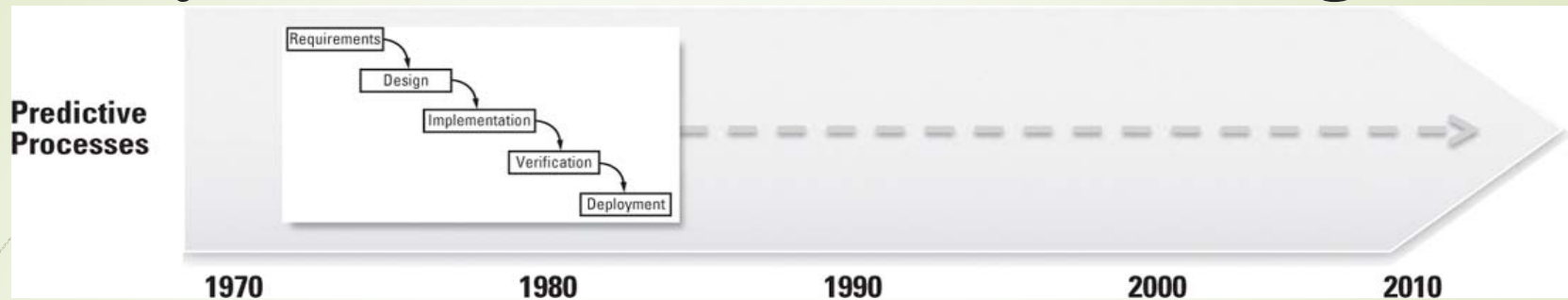
“**Scope management** related to attempting waterfall practices was the single largest contributing factor for failure.”

Study of 1,027 IT projects in the UK [Thomas 2001]

- Here's the study's conclusion:

“This suggests that...the approach of full requirements definition, followed by **a long gap** before those requirements are delivered, is no longer appropriate. **The high ranking of changing business requirements suggests that any assumption that there will be little significant change to requirements once they have been documented is fundamentally flawed.**”

# Why the waterfall model is still amongst Us ?



- There are a number of understandable reasons:
  - The model was itself born as a **fix** to the “**code it, fix-it, code-it- some-more-until-it’s-quickly-not-maintainable**” problem.
  - It appears to be **logical**: Understand requirements. Design a system that conforms. Code it. Test it.
  - It worked to a point (we did and still do ship a *lot* of software using the water model).
  - It reflects a continuing market reality —**customers still do impose fixed-date/Fixed** requirements agreements on suppliers.

# Iterative and Incremental Processes



- Failures of the waterfall model
- **Increasing time-to-market pressures**
- Advances in software development tools and technologies,



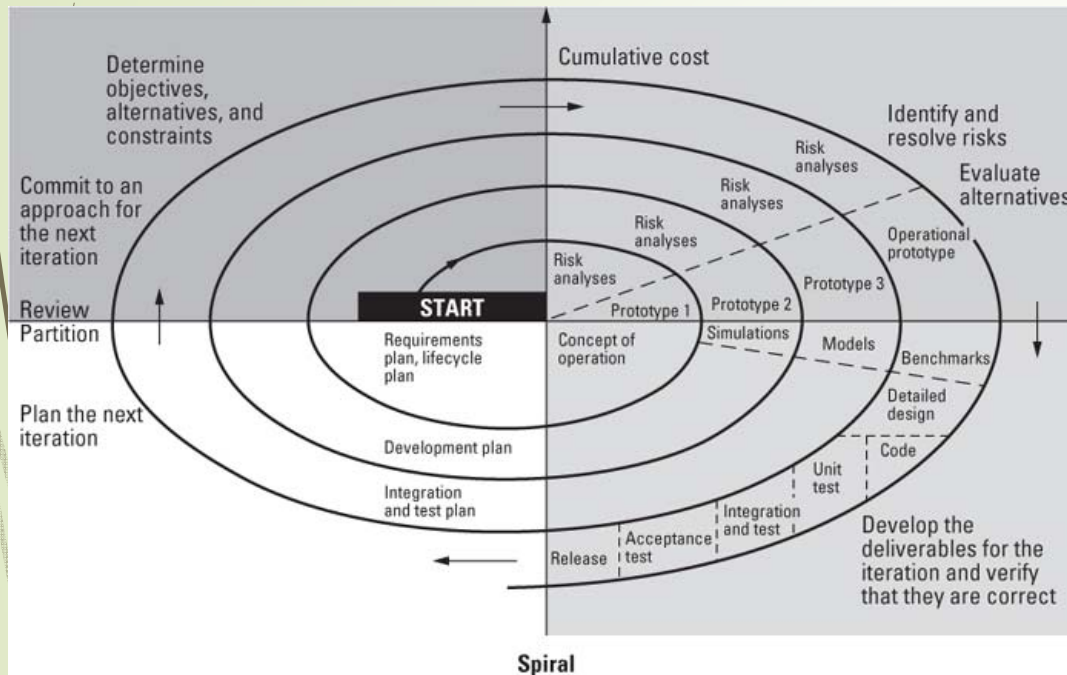
**Drove the need for more innovative, *discovery-based* models**



**The iterative processes of the 1980s and 1990s**



# The Spiral Model

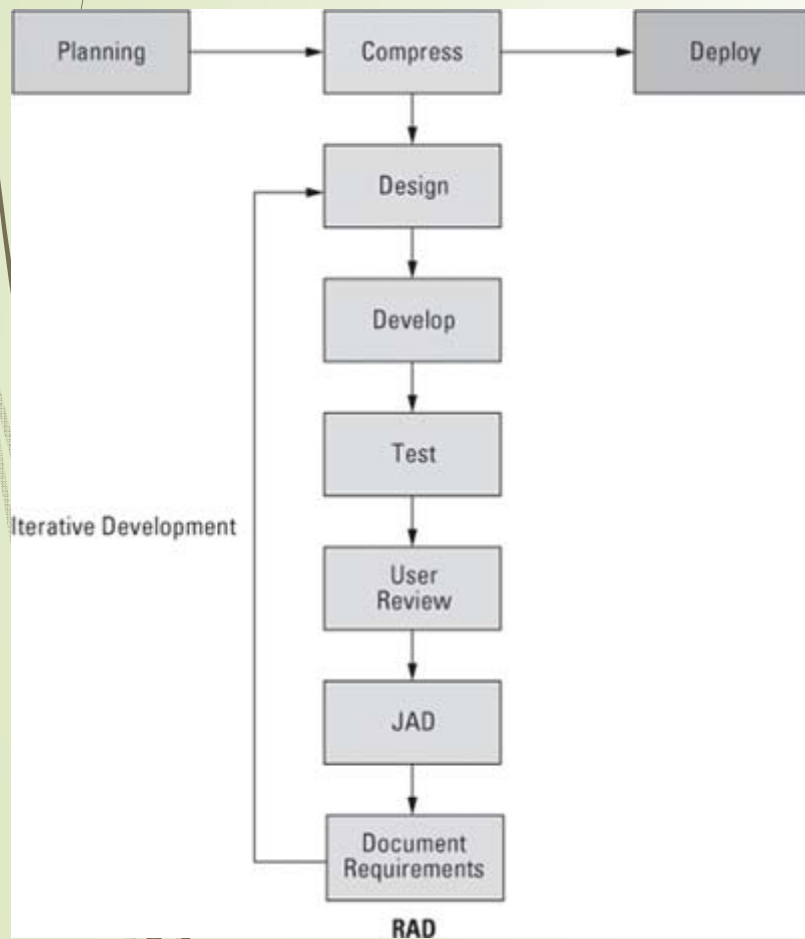


**Requirements still have a strong early placeholder.**

- An initial pass around the spiral is intended primarily to **understand requirements and perform some validation of the requirements** before more serious development begins.
- Thereafter, the model assumed another, larger “spiral” intended to **develop the solution in largely sequential steps of design, coding, integration, and testing.**
- Follows a traditional sequential, waterfall like process, **but incorporates constant feedback.**

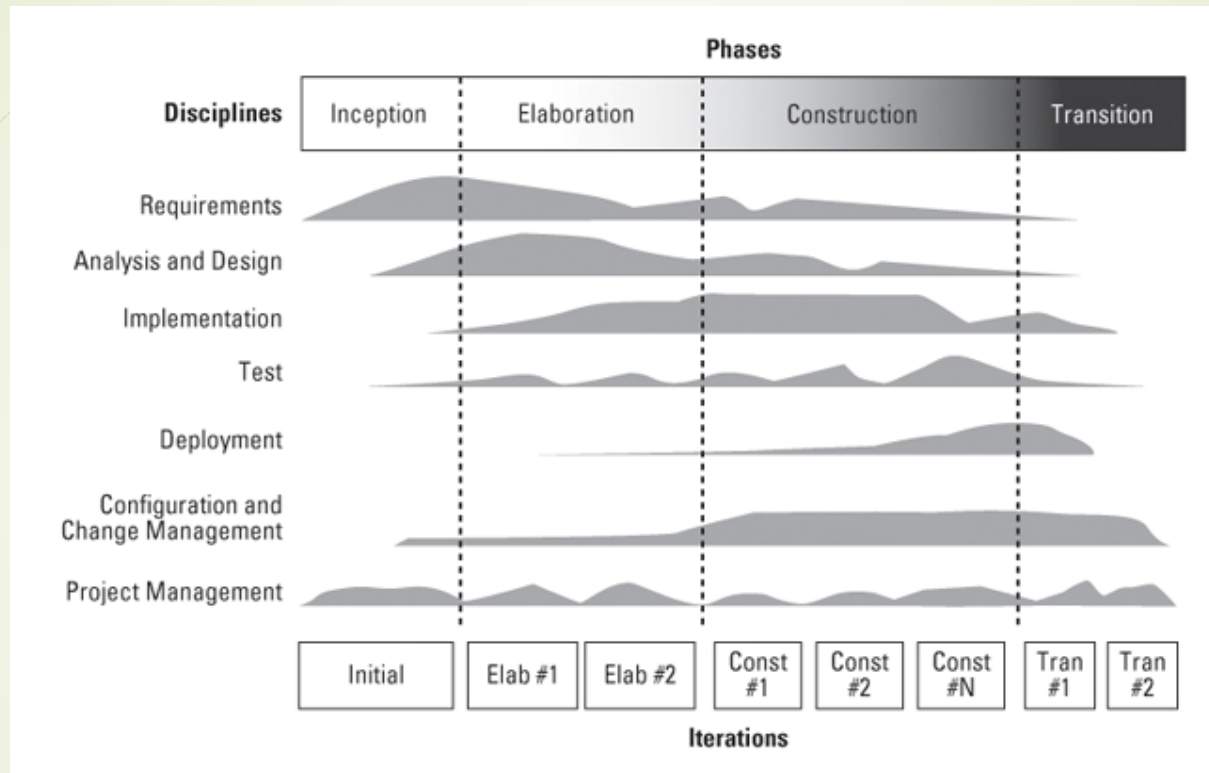
# The Rapid Application Development (RAD)

10



- Focuses on the iterative development and construction of **an increasingly capable series of prototypes**.
- It generally stands for any number of **lighter-weight approaches**, using fourth generation languages and frameworks (such as web application frameworks), which accelerate the availability of working software.
- **From a requirements perspective, the assumption was that if you could build it fast enough before the requirements changed, you would be more successful. And if you did get it wrong, the tools are sufficiently facile and lightweight that you could build it again faster than you could use traditional, paper-based requirements discovery methods.**

# The Rational Unified Process (RUP)



- **Widely adopted iterative and incremental** software process model (more than a million practitioners)
- Intended for large-scale applications where robustness, scalability, and extensibility are mandatory

# The Rational Unified Process (RUP)

- RUP recognized the **necessary overlap of the various *activities*** that occurred during the life cycle *phases* of *inception*, *elaboration*, *construction*, and *transition*.
- For example, activities such as “requirements” were no longer **relegated to a single phase**.
- Requirements activities were particularly intensive during the early inception and elaboration phases (as illustrated by the size of the “humps” in the diagram).
- **Requirements elaboration and requirements change are considered to be a continuous process that occurs throughout the life cycle.**

# Requirements in Iterative Processes

13

- No traditional *big, upfront design* (BUFD) requirements and design artifacts, such as software requirements specifications, design specifications, and the like.
- In its place, we see a “discovery based” approach.
- Apply lighter-weight documents and models such as vision documents, use case models, and so on, which are used to initially define what is to be built.
- The iterative process is applied to more quickly discover the “real user requirements” in early iterations, thus substantially reducing the overall risk profile of the project.

# Adaptive (Agile) Processes

14

## ➤ Adaptive models assume that:

**With the right development tools and practices—it was simply more cost effective to write the code quickly, have it evaluated by customers in actual use, be “wrong” (if necessary ), and quickly refactor it than it was to try to anticipate and document all the requirements up front.**

## Examples of adaptive methods:

- Dynamic Systems Development Method (DSDM), Feature-Driven Development (FDD), Adaptive Software Development, Scrum, Extreme Programming (XP), Open Unified Process (Open UP), Agile RUP, Kanban, Lean, Crystal Methods, etc.



# Agile Core Principles

15

- **Highest priority** is to **satisfy the customer** through **early** and **continuous delivery** of **valuable software**.
- **Welcome changing requirements**, even late in development.
- **Deliver working software** (primary measure of progress) frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
- **Business people and developers must work together daily throughout the project.**
- Build projects around **motivated individuals**: Give them the environment and support they need, and trust them to get the job done.

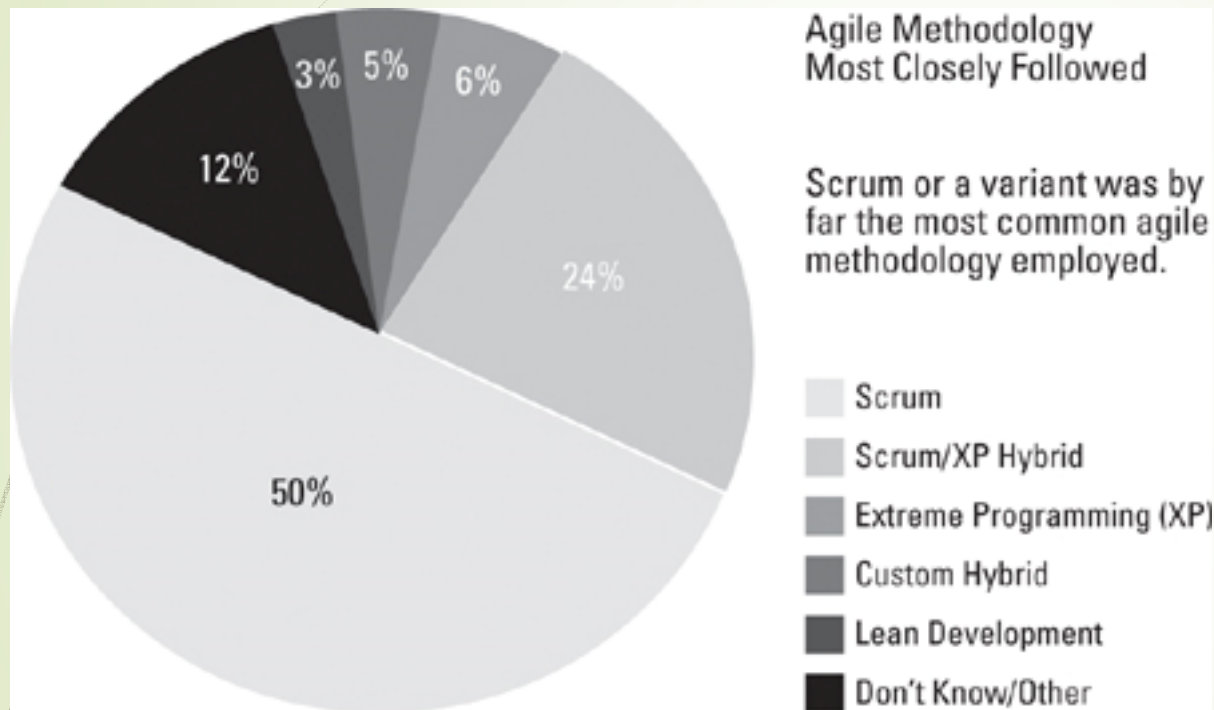
# Agile Core Principles

- Conveying information to and within a development team is **face-to-face conversation**.
- Agile processes promote **sustainable development**. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
- Continuous attention to technical excellence and good design enhances agility.
- Simplicity is essential.
- The best architectures, requirements, and designs emerge from self-organizing teams.
- At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.



# Agile Methods

17



- The most widely adopted agile methods are **Scrum** and **XP**.
- Scrum (with or without combination with XP) is now applied in 74% of agile implementations

Survey of most widely adopted agile methods. Fourth Annual State of Agile Development Survey 2009

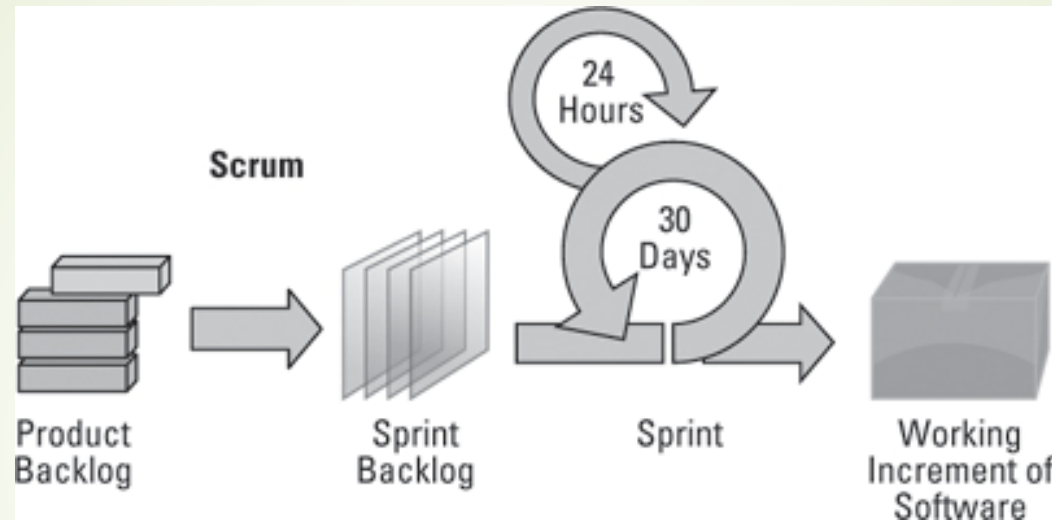
# Extreme Programming (XP)

18

Key practices of XP include the following:

1. A team of five to ten programmers work at one location with **customer representation onsite**.
2. Development occurs in **frequent builds or iterations**, which may or may not be releasable, and delivers incremental functionality.
3. **Requirements** are specified as **user stories**, each a chunk of new functionality the user requires.
4. Programmers work in **pairs**, follow **strict coding standards**, and do their own **unit testing**. Customers participate in **acceptance testing**.
5. **Requirements, architecture, and design emerge over the course of the project**.

# Scrum



**Scrum is an agile project management method.**

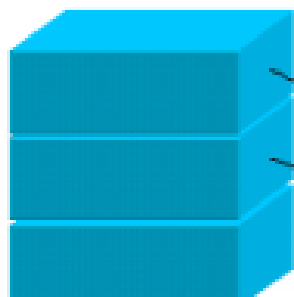
- Work is done in “**sprints**,” which are timeboxed iterations of a fixed 30 days or fewer duration.
- Work within a sprint is **fixed**. Once the scope of a sprint is committed, no additional functionality can be added, except by the development team.

# Scrum

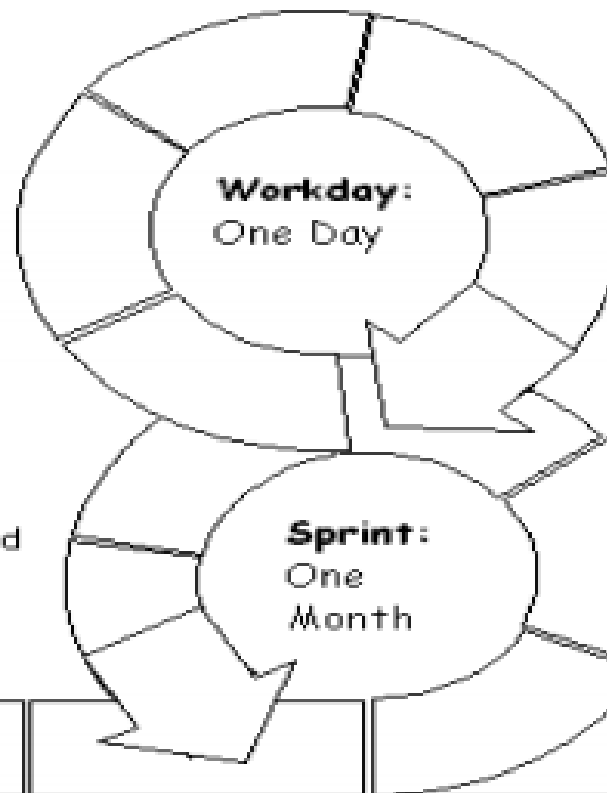
20

## SCRUM Sprint Cycle

**Product Backlog:**  
Prioritized list of features required by the customer



**Sprint Backlog:**  
Features to be done this sprint  
Features are expanded into smaller tasks.



**Every Day,** a 15-minute meeting is held, and the SCRUM Master asks the 3 questions:

- 1) What have you accomplished since the last meeting?
- 2) Are there any obstacles in the way of meeting your goal?
- 3) What will you accomplish before the next meeting?

**New Functionality** is demonstrated at the end of each sprint.

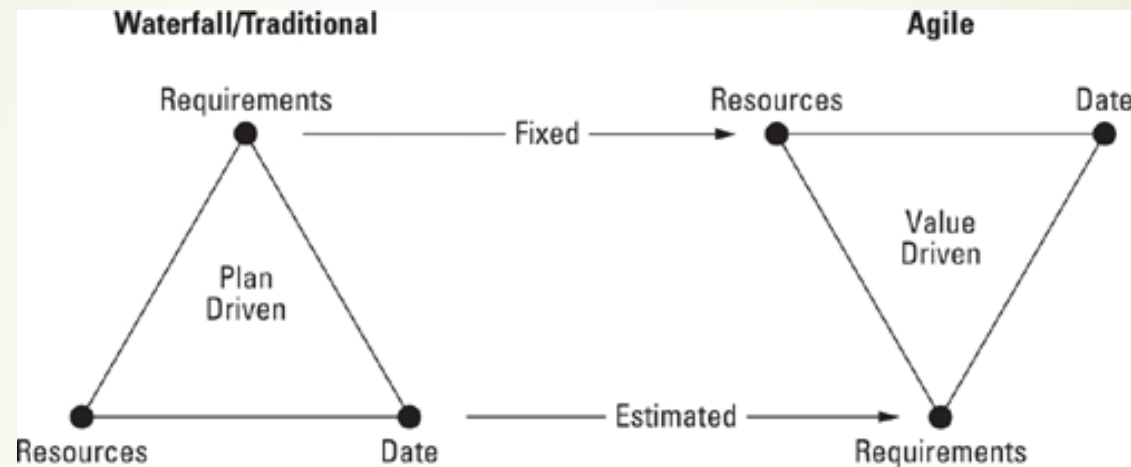


# Scrum

21

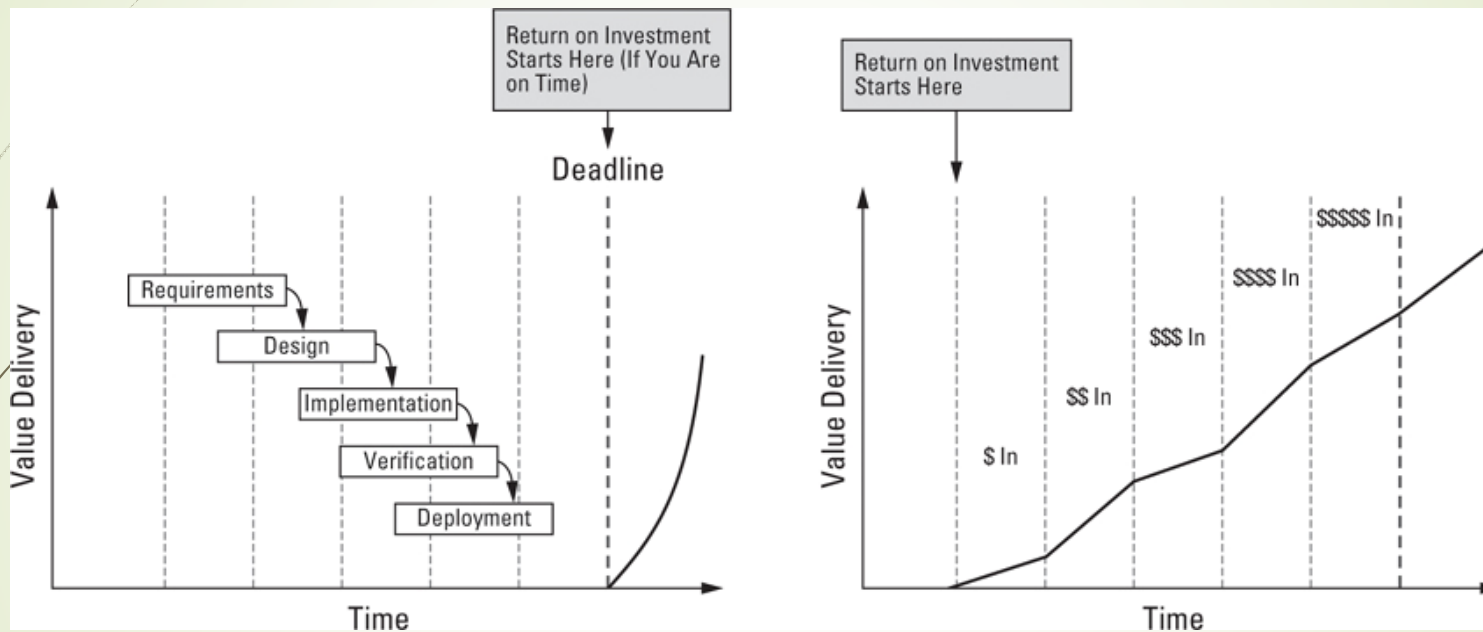
- **All work to be done is characterized as product backlog**, which includes new requirements to be delivered, the defect workload, and infrastructure and design activities.
- A *Scrum Master* mentors the empowered, self-organizing, and self-accountable teams that are responsible for delivery of successful outcomes at each sprint.
- A *product owner* plays the role of the customer proxy .
- A **daily stand-up meeting** is a primary communication method.
- Typical Scrum guidance calls for fixed **30 day sprints**, with approximately 3 sprints per release, thus supporting incremental market releases on a 90 day time frame.

# Requirements Management in Agile is Fundamentally Different



- In the agile battle of date versus scope, the date wins. In other words, with agile methods, we'll fix two things, **schedule** and **resources**, and we'll float the **scope** (requirements).

# Agile Optimizes ROI Through Incremental Value Delivery



Waterfall Return on Investment

Agile Return on Investment

➡ Sound economic principle:

**The sooner we deliver a feature,  
the sooner our customers will pay us for it**



## Agile Optimizes ROI Through Incremental Value Delivery

- In waterfall, **investment (cost) starts immediately and continues until delivery is reached**. No return on investment is possible until such time as all committed requirements have been delivered to the customer, or the deadline is reached.
- In agile, **value delivery starts with the first shippable increment**. Therefore, whether business value is measured in **customer retention** or **incremental pricing**, return on investment starts then too.
- If we assume the investment is constant:

$$\text{ROI } \$\$ \text{ (agile)} > \text{ROI } \$ \text{ (waterfall)}$$



## Agile Optimizes ROI Through Incremental Value Delivery

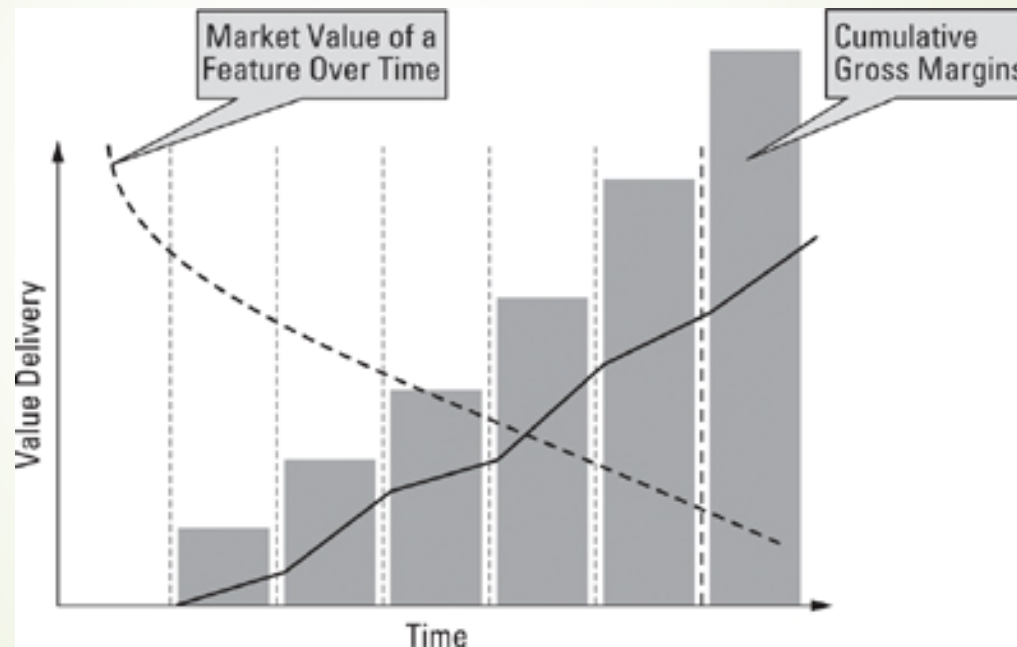
- Previous figure doesn't take into account **the differential value of early market features.**
- **Example:** Early iPhone was \$600 (few months of launch). Twenty four months later, you could buy a much more powerful version for about \$199, which is *one third the price*.
- Any one entering the market later with a **“me too” product** had **to compete at a much lower price.**
- Moreover, they had to **invest heavily to disrupt an incumbent market of early adopters** who are unlikely to switch as the iPhone makes its way into its users' daily lives.

**The value of any marketable feature decreases over time.**

# Agile Optimizes ROI Through Incremental Value Delivery

26

- To capture the **maximum gross profit**, you have to **be in the market first**, or at least **early enough** to where the **value/pricing differential** is still in effect.



- ROI actually increases at a rate even **faster than the linear rate** implied by the previous figure.