Agile Software Requirements

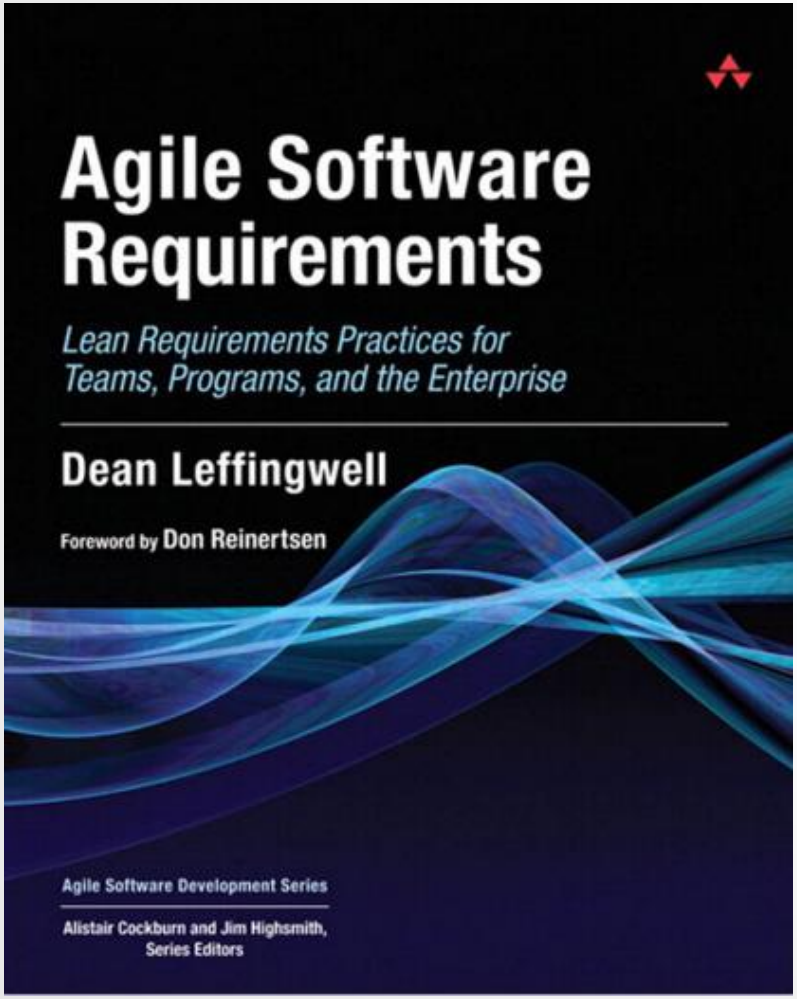
Software Requirements Engineering – 40688

Computer Engineering department

Sharif university of technology

Fall 402

Resource



Dean Leffingwell

Chapter 1:

A Brief History of Software Requirements Methods

Software Requirements in Context (1)

- Software development has become one of the world's most important technologies.
- In support of our efforts over the past 40 to 50 years, we have implemented various software development methodologies.
- Over time, the scope and reach of our endeavors, along with the power of the computers we programmed, increased by 10,000 fold.
- Because what we produce is not physical goods but intangible ideas reflected in binary code, our methods all had a primary focus on "**managing software requirements**."

Software Requirements in Context (2)

- *Software requirements* was the label we applied to the abstractions we use to carry the value stream into development and on to delivery to our customers.
- But over time, the applications we developed became larger and larger still, and the methods we used to control our work became heavier and heavier.
- So we started slowing down the very thing we were trying to speed up our ability to deliver higher-value, higher-quality software at faster and faster rates.

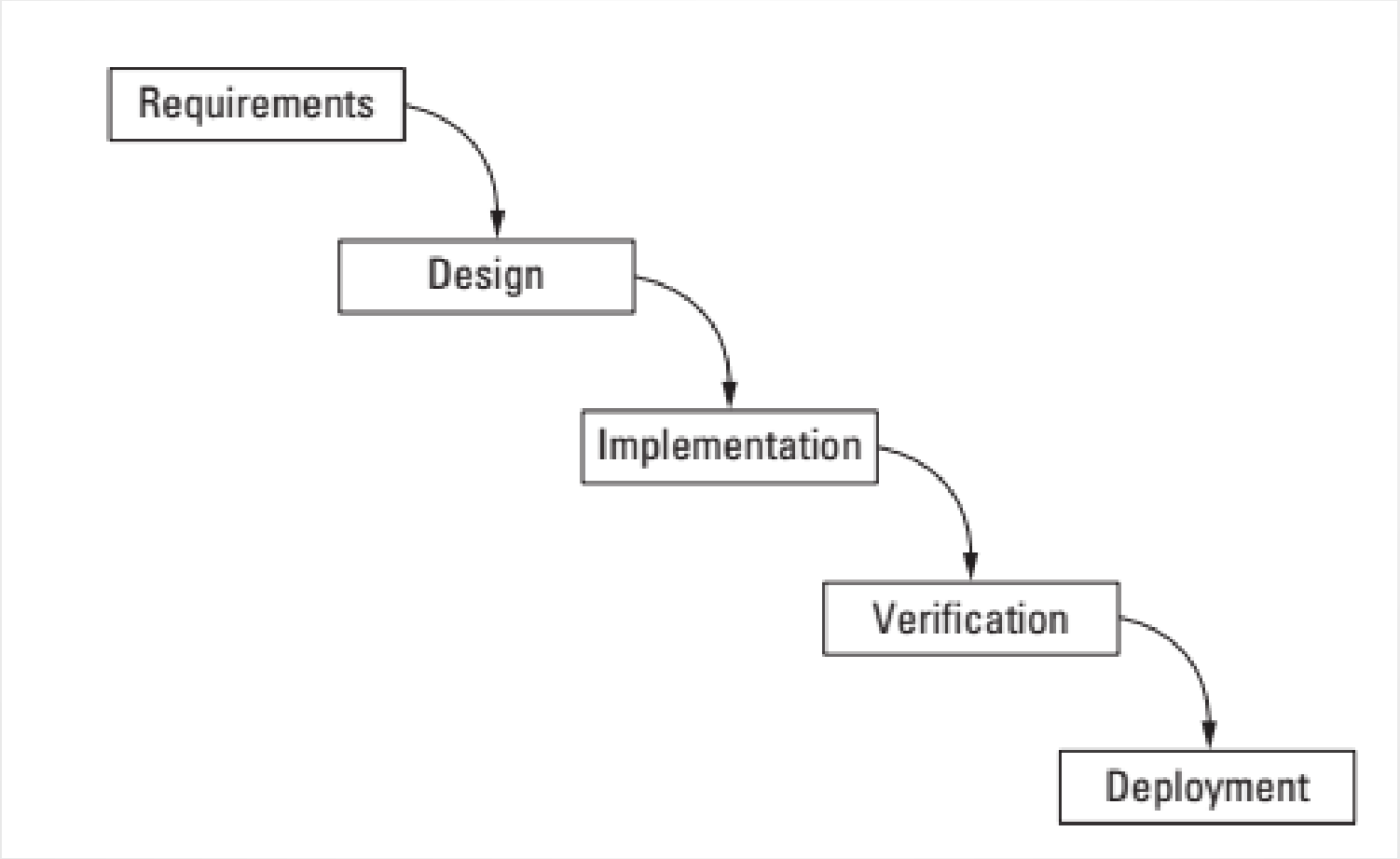
Software Requirements in Context (3)

- So, in the past two decades, the movement to more "**agile**" and "**leaner**" software development methodologies, including lighter weight but still safe and effective treatment of application requirements, has been one of the most significant factors affecting the industry.

Predictive, waterfall-like Processes

- The software industry advanced quickly after its inception in the 1950s and 1960s .
- **Winston Royce**, who was at TRW at the time, is often credited with the creation of this model.
- Ironically, however, Royce actually described it as a model that **could not** be recommended for large-scale software development.

Simplified “waterfall” model.

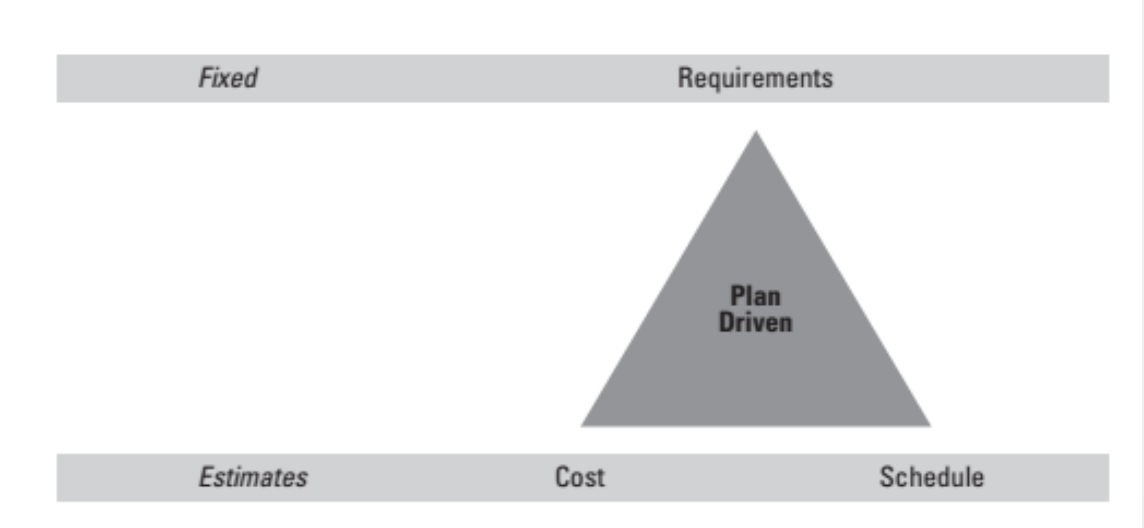


Problems with the Model

- As Royce would have likely predicted, the model hasn't worked all that well for **large software projects**, and we have all struggled under the burden.
- **31%** of projects will be **canceled before** they are **completed**.
- **53%** of the projects will **cost** more than **189%** of their **estimates**.
- Only **16%** of projects were **completed on time** and **on budget**.
- For the largest companies, completed projects delivered only **42%** of the **original features** and **functions**.

Requirements in the Waterfall Model

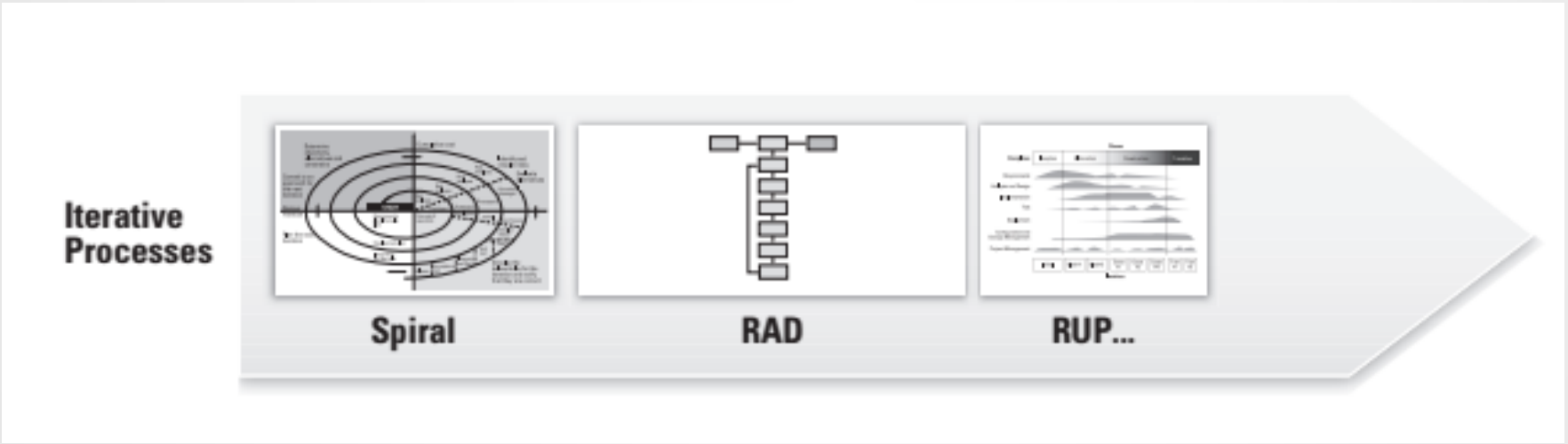
- The Iron Triangle



- What about **quality**?

Iterative and incremental Processes

- Led us to the iterative processes of the 1980s and 1990s.



Requirements in Iterative Processes (1)

- In iterative processes, we see a purposeful move away from the traditional *big, upfront design* (BUFD) requirements and design artifacts.
- In its place, we see a "discovery-based" approach.
- In the **iterative** model, we applied **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.

Requirements in Iterative Processes (2)

- Based on these initial understandings, the iterative process itself is then **applied to** more quickly **discover** the "**real user requirements**" **in early iterations**, thus substantially **reducing** the **overall risk** profile of the project.
- Once better defined in early iterations, these **requirements** were then implemented in a **fairly robust** but **mostly traditional build-out of code, tests**, and so on, to implement the requirements and **provide assurances** that the system conformed to the agreed-to behaviors.
- Clearly, this was a giant step forward for the industry and one that started to soften the boundaries of the iron triangle.

Adaptive (agile) Processes (1)

- Starting in the late 1990s and through the current decade, software process has seen an explosion of **lighter-weight** and **ever-more-adaptive models**.
- Based on some fundamental changing software implementation paradigms such as object orientation, 3G languages, and test-driven development, these models were based on a different economic foundation.
- These models assumed 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.

Adaptive (agile) Processes (2)

- Indeed, the number of methods—including 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, and so on—speaks to the industry's thirst and constant drive for more effective and lighter-weight processes.

Agile manifesto

We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

1. Individuals and **interactions** over processes and tools.
2. **Customer collaboration** over contract negotiation.
3. Working software over **comprehensive documentation**.
4. **Responding** to **change** over following a plan.

Agile principles (1)

1. Our highest priority is to satisfy the **customer** through **early** and **continuous delivery** of valuable software.
2. **Welcome changing requirements**, even late in development. Agile processes harness change for the customer's competitive advantage.
3. **Working software** is the primary measure of progress.
4. **Deliver** working software **frequently**, from a couple of weeks to a couple of months, with a preference to the **shorter timescale**.

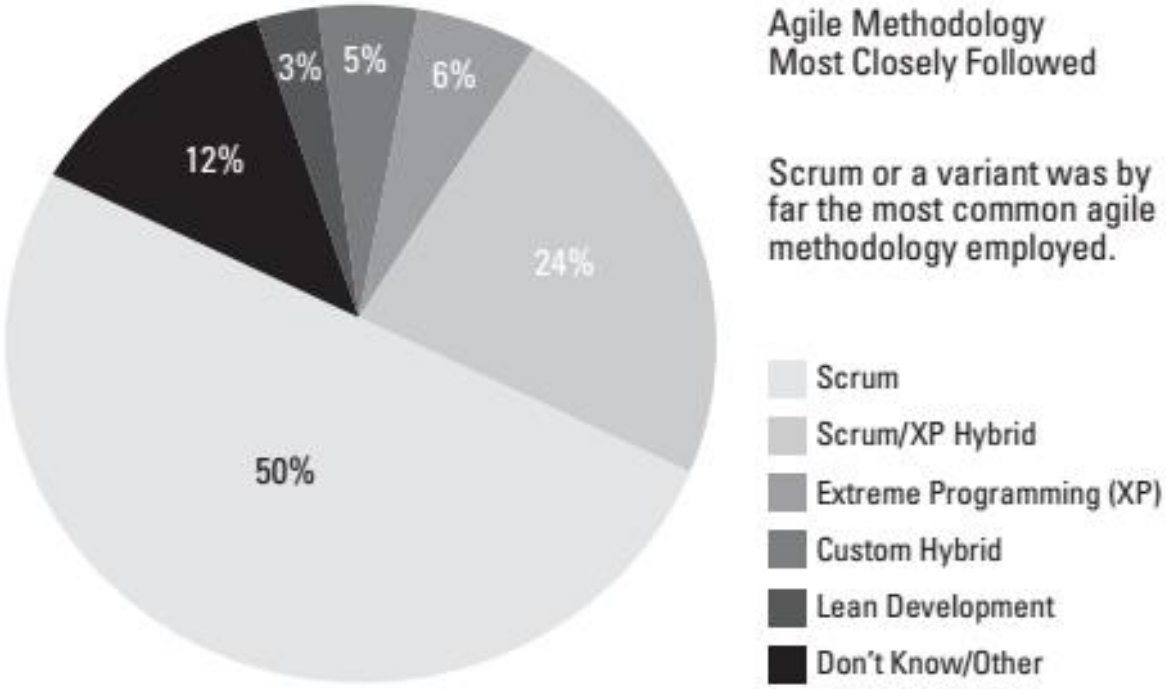
Agile principles (2)

5. **Business people** and **developers** must **work together** daily throughout the project.
6. Build projects around **motivated individuals**. Give them the environment and support they need, and trust them to get the job done.
7. The most efficient and effective method of conveying information to and within a development team is **face-to-face conversation**.
8. Agile processes promote **sustainable development**. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.

Agile principles (3)

9. **Continuous attention** to technical excellence and good design enhances agility.
10. **Simplicity**—the art of maximizing the amount of work not done—is essential.
11. The best architectures, requirements, and designs emerge from **self-organizing teams**.
12. At regular intervals, the team reflects on **how to become more effective**, then tunes and adjusts its behavior accordingly.

A Comparison



Extreme Programming (XP) (1)

XP is a widely used agile software development method that is described in a number of books by Beck and others.

Key practices of XP include the following:

1. A team of **five** to **ten programmers** work at one location with customer representation on-site.
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.

Extreme Programming (XP) (2)

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.

XP is prescriptive in scope and is typically applied in small teams of less than ten developers, where the customer is integral to the team or readily accessible.

In addition, the *P* in XP stands for *programming*, and as opposed to other methods, XP describes some strict practices for coding that have been shown to produce extremely high-quality output.

Scrum (1)

Scrum is an **agile project management method** [Schwaber 2004] that is enjoying increasing widespread use.

Key Scrum practices include the following:

1. Work is done in "**sprints**," which are time boxed iterations of a fixed 30 days or fewer duration.
2. **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.
3. 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.

Scrum (2)

5. A **Scrum Master** mentors the empowered, self-organizing, and self-accountable teams that are responsible for delivery of successful outcomes at each sprint.
6. A **product owner** plays the role of the **customer proxy.**
7. A **daily stand-up meeting** is a primary communication method.
8. A **heavy focus** is placed on **time boxing**. Sprints, stand-up meetings, release review meetings, and the like are all completed in prescribed times.

Requirements Management in Agile is fundamentally different (1)

No matter the specific method, agile's treatment of requirements is fundamentally different. We see it immediately in the core principles:

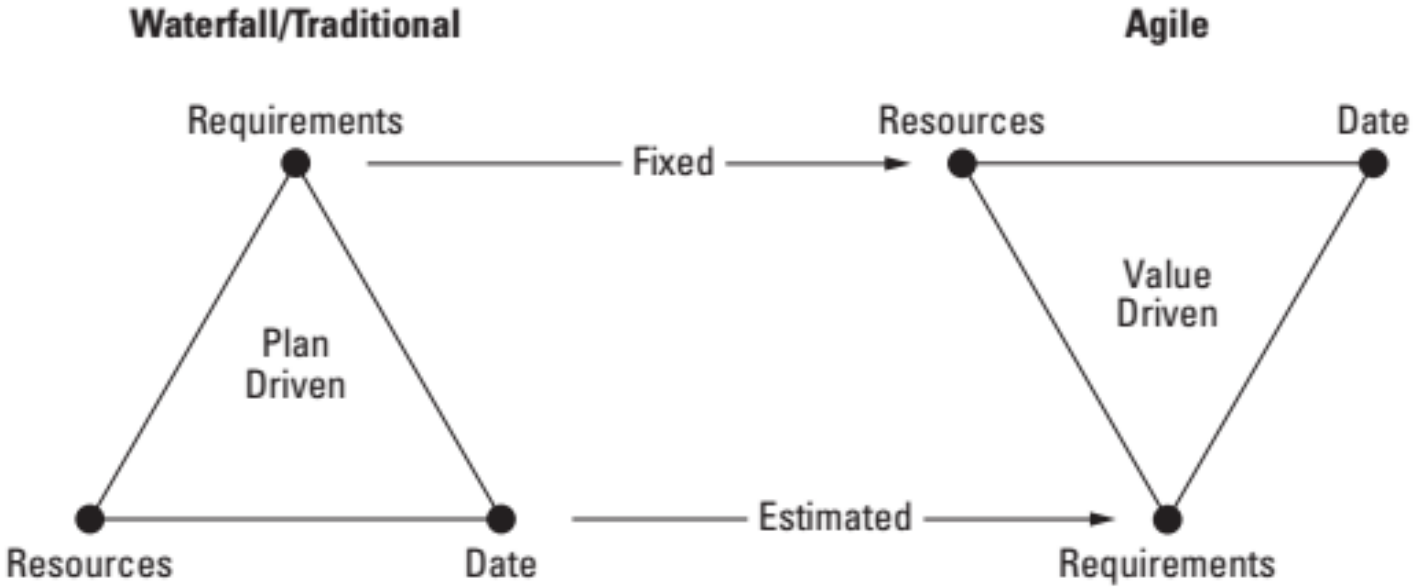
- *Manifesto principle #1—Our **highest priority** is **to satisfy** the **customer** through **early** and **continuous delivery** of valuable software.*
- *Manifesto principle #2—**Welcome changing requirements**, **even late** in development. Agile processes harness change for the customer's competitive advantage.*

Requirements Management in Agile is fundamentally different (2)

- (with agile) instead of **investing months in building detailed software requirements specifications** ...teams focus on delivering early, value-added stories into an integrated baseline. Early delivery serves to test the requirements and architectural assumptions, and it drives risk out by proving or disproving assumptions about integration of features and components.
- No longer do management and the user community wait breathlessly for months, hoping the team is building the right thing. At worst, the next checkpoint is only a week or so away, and ...users may be able to deploy even the earliest iterations in their own working environment.

Goodbye Iron Triangle (1)

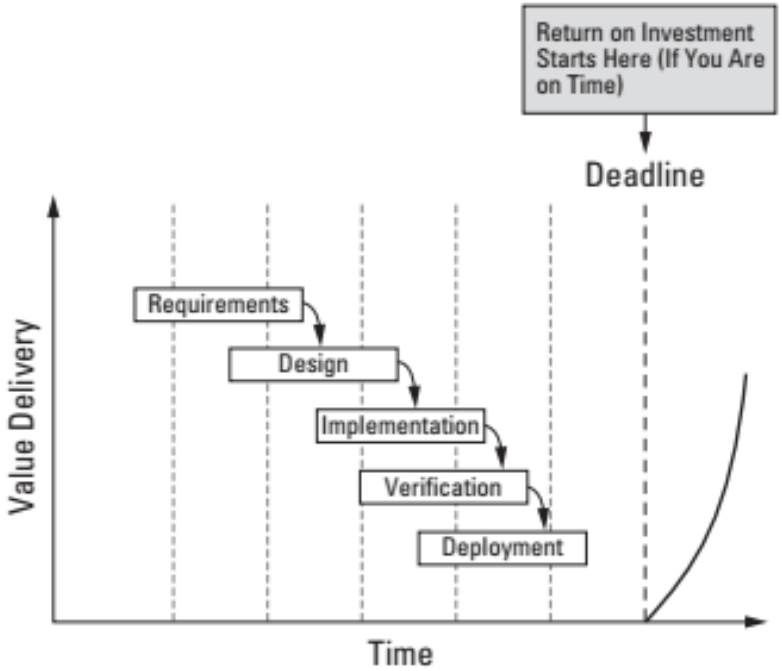
The net effect of this change is to eliminate the iron triangle that has kept us from achieving our quality and dependability objectives.

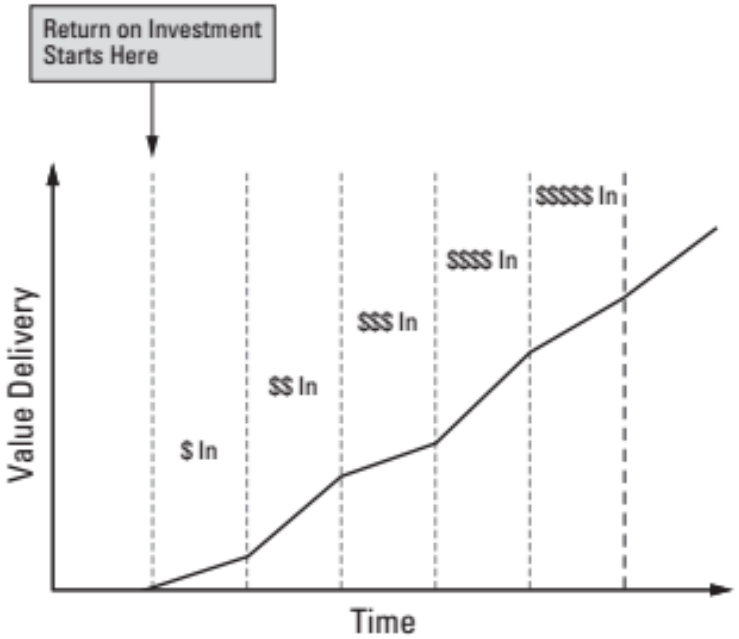


Goodbye Iron Triangle (2)

- Also, as we apply the appropriate agile technical practices, quality is also fixed. So, now we have a truly virtuous software cycle:
- *Fix quality—deliver a small increment in a timebox—repeat.*

Agile Optimizes ROI Through Incremental Value Delivery





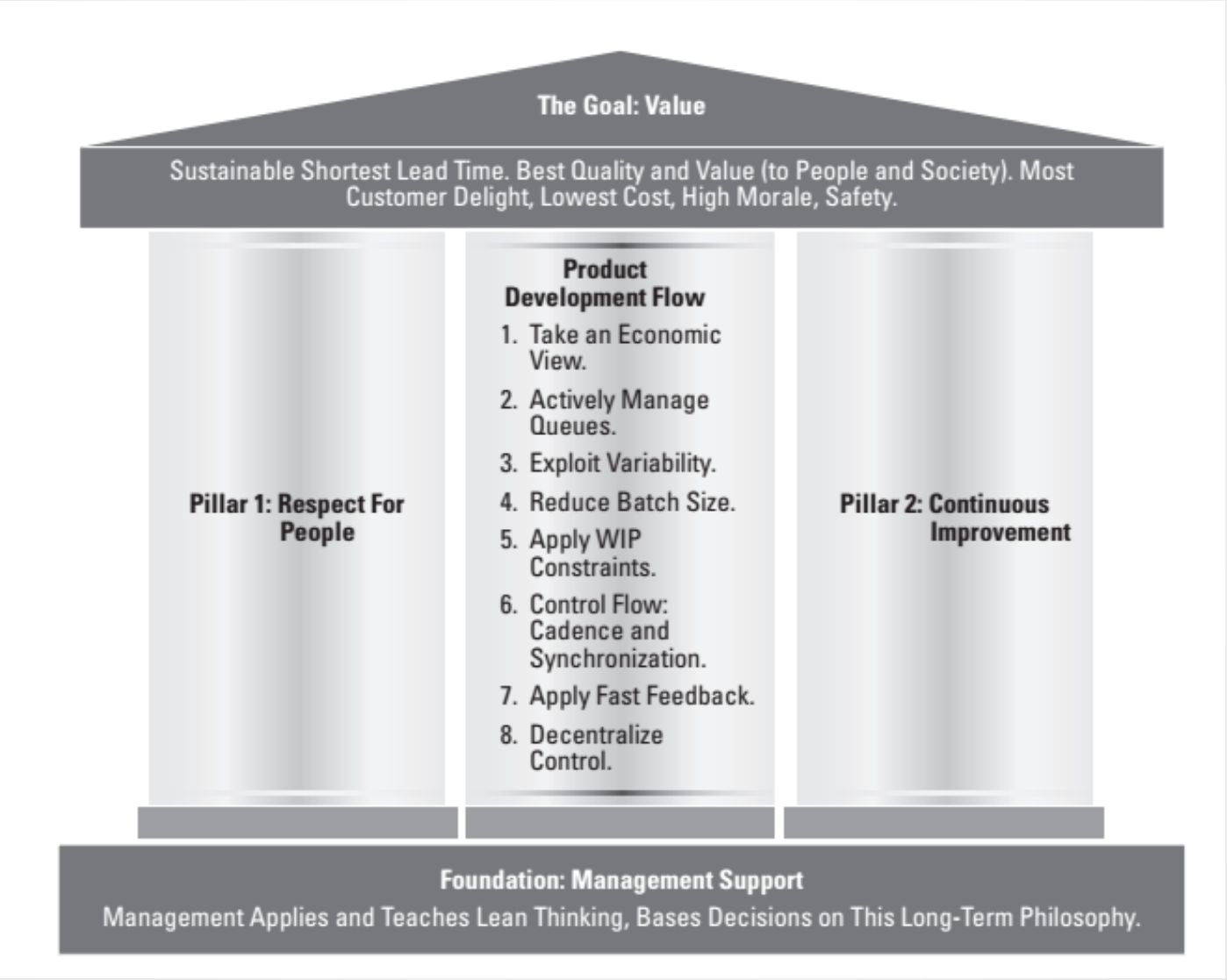
Introduction to LEAN software

- In a parallel universe, the roots of the lean software movement were evolving primarily from the successes of Toyota and the **Toyota Production System (TPS)**, an alternative to the mass production systems in use at the time.
- TPS is a set of lean, economically-based manufacturing philosophies, principles, and practices used to vault Toyota to become the world's leading car manufacturer by 2007.
- Indeed, it is easy to view XP, Scrum, and others as "software instances of lean," and as such, lean provides an even broader framework for **improving the economics of new product development** in those enterprises dependent on software.

The House of Lean Software

- Larman and Vodde have described a framework for lean software thinking that translates many of the core principles and practices into a manageable software context.
- In so doing, they also reintroduced a “house of lean thinking” graphic, inspired by earlier houses of lean from Toyota and others.

The House of Lean Software



Kanban:
Another Software Method Emerges

Kanban (the word means “signal” in Japanese) is the label for a way of scheduling and managing software work that is seeing increasing use in the agile community.

- Visualizes some unit of value.
- Manages the flow of these units of value, through the use of WIP limits.
- Deals with these units of value through the whole system, from when they enter a team’s control until they leave it.
- The pull system of Kanban tends to quickly expose impediments, blocking issues, and bottlenecks in the flow (which may result from either a capacity constraint or no instant availability of a resource).

Summary

1. In this introductory chapter, we provided a **brief history of requirements methods** as they have evolved over the past 20 to 30 years.

2. We did so for two reasons: to provide context for advancing methods based on lessons learned in the past, and since all these methods are still at work in the industry today, to help us understand the existing practices before we attempt to improve them.

3. We introduced **agile development methods** that are being successfully applied at the team and enterprise levels. We noted how these methods are being further advanced, and further scaled, through the application of **lean** and **flow principles**.