History and Evolution of Requirements Paradigms 需求范式的历史与演变

Fahad Sabah

SRE_Fall2025_Beijing University of Technology, China
SRE_Fall2025_北京工业大学，中国

What You Will Learn (Learning Outcomes)

- ❑ **Explain the core principles, strengths, and weaknesses of key historical software development paradigms (Waterfall, Spiral, V-Model).**
- ❑ **Articulate the drivers behind the shift to Agile and DevOps and their impact on requirements management.**
- ❑ **Describe modern trends like AI-driven requirements and Model-Based Systems Engineering (MBSE).**
- ❑ **Analyze a real-world case study to understand the practical implications of a paradigm shift on requirements processes.**

你将学到什么（学习成果）

- ❑ **解释关键历史软件开发范式（瀑布、螺旋、V模型）的核心原则、优势和劣势。**
- ❑ **阐明向敏捷和DevOps转型的驱动力及其对需求管理的影响。**
- ❑ **描述现代趋势，如人工智能驱动的需求和基于模型的系统工程（MBSE）。**
- ❑ **分析一个真实案例研究，以理解范式转变对需求流程的实际影响。**

01

为什么选择SDLC

Why SDLC

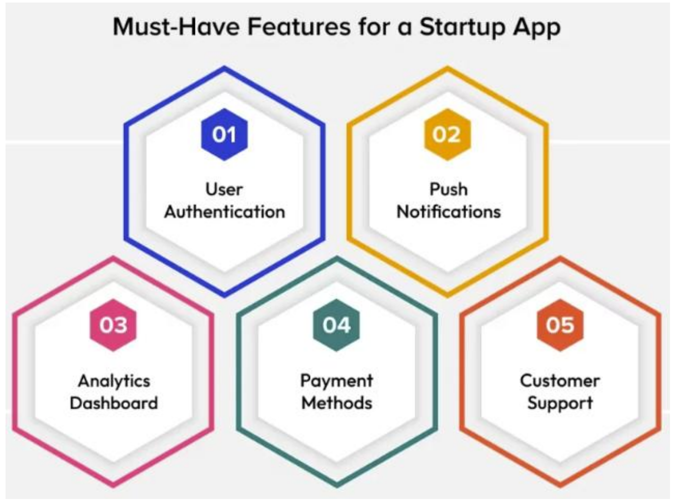
摩天大楼与启动应用的比较
Skyscraper vs. Startup App

01

Skyscraper Project

02

Startup App Project



03

Project Nature Dictates Approach

The nature of the project—whether it's stable and well-understood or dynamic and uncertain—dictates the development approach. The choice between plan-driven and adaptive methods is crucial for success.

什么是SDLC
What Is the SDLC

Definition of SDLC

The Software Development Life Cycle (SDLC) is a systematic process for building software that aims to produce **high-quality, cost-effective** software in the **shortest possible time**.

SDLC
阶段

SDLC
Stages

1. IDEATION	• Brainstorming ideas that solve a particular problem faced by target users
2. REQUIREMENTS	• Interfacing with the stakeholders and users to collect and document project requirements
3. DESIGN	• Creating the architecture of a software system and its elements
4. DEVELOPMENT	• Building the software using a programming language by the development team
5. TESTING	• Evaluating the quality of software with the aim of finding and fixing defects
6. DEPLOYMENT	• Preparing the software to run and operate in a specific environment
7. MAINTENANCE	• Updating and supporting the software after it has been delivered to the market.

1. IDEATION [1. 创意]

At this stage, the team has a lot to learn about the client's business and ideas.

Basic actions conducted during the ideation phase:

- ❑ review the documentation sent by the client (if it exists);
- ❑ research the domain and spend time getting familiar with it (especially if it's niche);
- ❑ conduct meetings with the stakeholders to discuss basic requirements, provide suggestions, and brainstorm ideas;

- ❑ **Make sure to consider the following:**
  - ❑ Whose opinion can influence the product (it can be someone from the client's side, specific departments like Marketing, or national or international regulations, laws, and institutions)?
  - ❑ What are the goals of these people, departments, or institutions?
  - ❑ Who is the product being made for?
  - ❑ Who will use the product?
  - ❑ Why is the product interesting and/or useful for these people?
  - ❑ Where do they live?
  - ❑ Do their daily routines need to be considered?
  - ❑ Does the client have any ideas about the monetization strategy for the product?

在第一阶段结束时，团队已经掌握了一些产品信息，并与客户找到了共同点。

- ❑ 项目总体思路;
- ❑ 高级思维导图;
- ❑ 利益相关者及其目标;
- ❑ 竞争 对手;
- ❑ 关键术语与定义

By the end of this first stage, the team already has some information about the product and found a common ground with the client.

- ❑ general idea of the project;
- ❑ high-level mind map;
- ❑ stakeholders and their goals;
- ❑ competitors;
- ❑ key terms and definition

2. REQUIREMENTS [要求]

This stage is still a part of the discovery phase. It gives a business analyst a chance to gather any specific requirements for the project and prepare the documentation needed for the next step.

Below are the main techniques used to elicit requirements:

- ❑ interviewing;
- ❑ holding workshops;
- ❑ brainstorming;
- ❑ observing.

By the end of this stage, a software requirements specification (SRS) is drafted and continuously reviewed by a quality assurance specialist to help all technical and non-technical stakeholders stay on the same page. The document describes functional and non-functional requirements and comes with a prototype.

3. DESIGN [设计]

This step is based on work done during the previous step. An architect prepares documentation pertaining to software architecture describing the architectural approach and data flow for the particular project, as well as technical decisions and third-party integrations.

During this phase, a test strategy is also defined for the further stages.

By the end of this stage, the **discovery phase** of the software development life cycle is considered to be finished and the project is ready for development.

4. DEVELOPMENT [发展]

Only at this step developers start implementing the project according to prepared documentation.

The quality of this step greatly depends on developers' experience and how well documents are detailed.

5. TESTING [测试]

This stage usually goes hand in hand with development. Testing provides quality assurance and makes sure that the project is implemented according to the requirements.

During testing, if any bug is found, the team returns to development again. It's common that a short cycle of these two steps will repeat until all bugs are removed.

There are several types of testing conducted at this stage:

- manual testing according to acceptance criteria;
- regression testing;
- load testing;
- unit-testing (optional);
- integrational (optional);
- auto testing (optional).

6. DEPLOYMENT [部署]

This is the final stage of the project. It implies deployment to the product server and/or digital distribution platforms.

7. MAINTENANCE [保养]

After product release, additional support may be needed. Additional documentation and support hours are discussed individually.

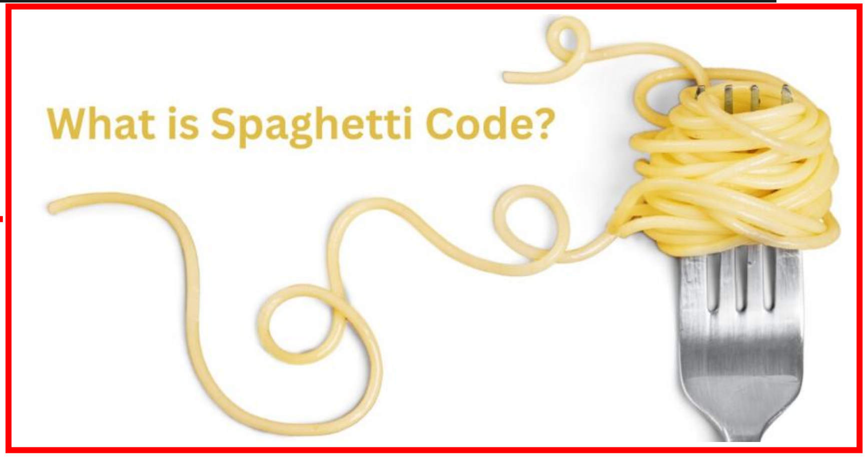
The Fundamental Problem [根本问题]

Software systems are complex, abstract, and expensive to build. Models help us tackle these challenges by providing **abstractions** that make complex systems understandable and manageable.

1. To Manage Complexity - The "Divide and Conquer" Approach
Problem: You can't build Amazon in one giant code file.
Example: E-commerce System Model

```
[Customer] → (Places) → [Order] → (Contains) → [Order Items] → (References) → [Product]
      ↓                     ↓
[Payment Method]         [Shipping Address]
```

What this model does:
•Breaks a massive system into understandable pieces
•Different teams can work on different parts simultaneously
•Prevents the "spaghetti code" nightmare



2. To Communicate Clearly - The "Shared Language" [清晰沟通—“共享语言”]

Problem: Everyone has a different mental picture of what "user login" means.

Without Model:

•Student A: "I'll just add email and password fields“

•Student B: "We need social media login buttons“

•Student C: "What about password reset?“

With Model (Wireframe):

```
+------------------------------+
|          [Logo]              |
|                              |
| Email: [_____________]       |
| Password: [____________]     |
|                              |
| [ Sign In ]    [ Forgot? ]  |
|                              |
| -- or sign in with --        |
| [Google] [Facebook] [GitHub] |
+------------------------------+
```

Result: Everyone builds the **same thing.**

3. 及早发现错误——“廉价橡皮擦与昂贵代码变更”原则

3. To Find Mistakes Early - The "Cheap Eraser vs. Expensive Code Change" Principle

Analogy: It's easier to erase a whiteboard than to tear down a built wall.

Example: Database Design Flaw

Initial (Flawed) Model:

```
STUDENTS Table:
- student_id
- name
- course_name  ← PROBLEM: What if a student takes multiple courses?
```

During modeling, we spot the issue and fix it:

```
STUDENTS Table:               COURSES Table:          ENROLLMENTS Table:
- student_id                  - course_id             - enrollment_id
- name                        - course_name           - student_id
- email                       - instructor            - course_id
                                                        - semester
```

Cost of fixing on paper: 2 minutes

Cost of fixing after coding: 2 days of rewriting + testing

4. 规划与估算—“建设蓝图”原则
4. To Plan and Estimate - The "Construction Blueprint" Principle

Problem: "How long will this project take?" Without models, it's just guessing.

With Architectural Model:

```
Frontend (React) → API Gateway → Backend Services → Database
        ↓                ↓               ↓                ↓
~2 weeks         ~1 week         ~3 weeks         ~1 week
```

Now you can realistically estimate: **7 weeks total** with parallel work.

5. 维护记录—“未来你的地图”
5. To Document for Maintenance - The "Map for Future You"

Problem: You return to your code 6 months later and think: "What was I thinking?!"

With Models:

- ❑ Sequence diagrams show how components interact
- ❑ Class diagrams show the structure
- ❑ State diagrams show behavior flows

Example: Login Sequence Documentation

```
User → Frontend: Enter credentials
Frontend → AuthService: Validate
AuthService → Database: Check user
AuthService → Frontend: Success/Fail
Frontend → User: Show appropriate message
```

主要要点

1. 模型是思考工具—不仅仅是文档
2. 早期建模省去深夜调试—在修复成本低的时候发现问题
3. 模型促进团队合作—没有模型，你就在构建不同的系统
4. 合适的模型取决于你的受众—开发者用技术图，客户用简单图
5. 模型会演变—它们是随着理解而变化的活文档

Key Takeaways

1. **Models are thinking tools** - not just documentation
2. **Early modeling saves late-night debugging** - find problems when they're cheap to fix
3. **Models enable teamwork** - without them, you're building different systems
4. **The right model depends on your audience** - technical diagrams for developers, simple diagrams for clients
5. **Models evolve** - they're living documents that change with your understanding

在软件开发过程中可以使用几种SDLC模型:

- ❑ 瀑布模型;
- ❑ 迭代模型;
- ❑ V型;
- ❑ 螺旋模型;
- ❑ 敏捷模型。

每个模型都有一套特定的步骤和特殊流程。团队通过部署不同模型，可以体验到不同的工作效率。最常见的是迭代模型。

Several SDLC models can be used during the software development process:

- ❑ waterfall model;
- ❑ iterative model;
- ❑ V-model;
- ❑ spiral model;
- ❑ Agile model(s).

Each model has a specific set of steps and special processes. The team can experience different kinds of work efficiency by deploying different models. The most common is the iterative model.

瀑布模型介绍
Waterfall Model Introduction

Waterfall Origin [瀑布起源]

瀑布模型由温斯顿·罗伊斯于1970年提出，源自制造业和建筑业。这是一个线性、顺序的流程，每个阶段都必须完成并签字后才能进入下一阶段。

Waterfall Model Overview

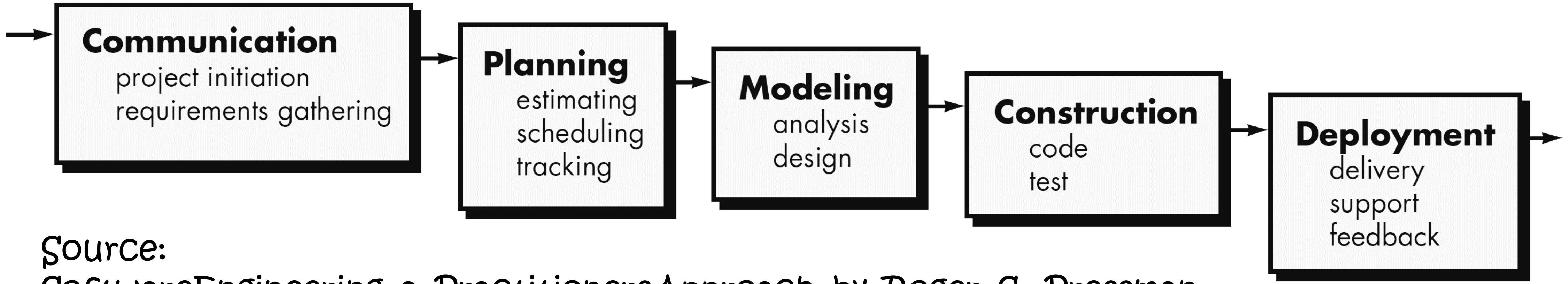
The Waterfall Model, introduced by Winston Royce in 1970, is derived from manufacturing and construction industries. It is a linear, sequential process where each phase must be completed and signed off before moving to the next.

The waterfall model [瀑布起源]

- ❑ Classic Lifecycle
- ❑ Systematic Sequential Approach*

- ❑ **Core Philosophy:** "Measure twice, cut once." Fully understand, document, and sign-off on all requirements before any design or coding begins. Development flows sequentially downhill, like a waterfall, through distinct phases.

- ❑ **The Process:**



Source:
SoftwareEngineering_a_PractitionersApproach_by_Roger_S._Pressman

*NOTE: Although the original waterfall model proposed by Winston Royce [Roy70] made provision for "feedback loops," the vast majority of organizations that apply this process model treat it as if it were strictly linear.

V模型（验证与核实模型）
The V-Model (Validation and Verification Model)

Core Philosophy: Extends Waterfall by emphasizing that testing should be planned in parallel with the corresponding development phase. It creates a strong, explicit link between requirements and testing.

The Process: The left side of the "V" is the decomposition of requirements and creation of system designs. The right side is the integration and testing phases.

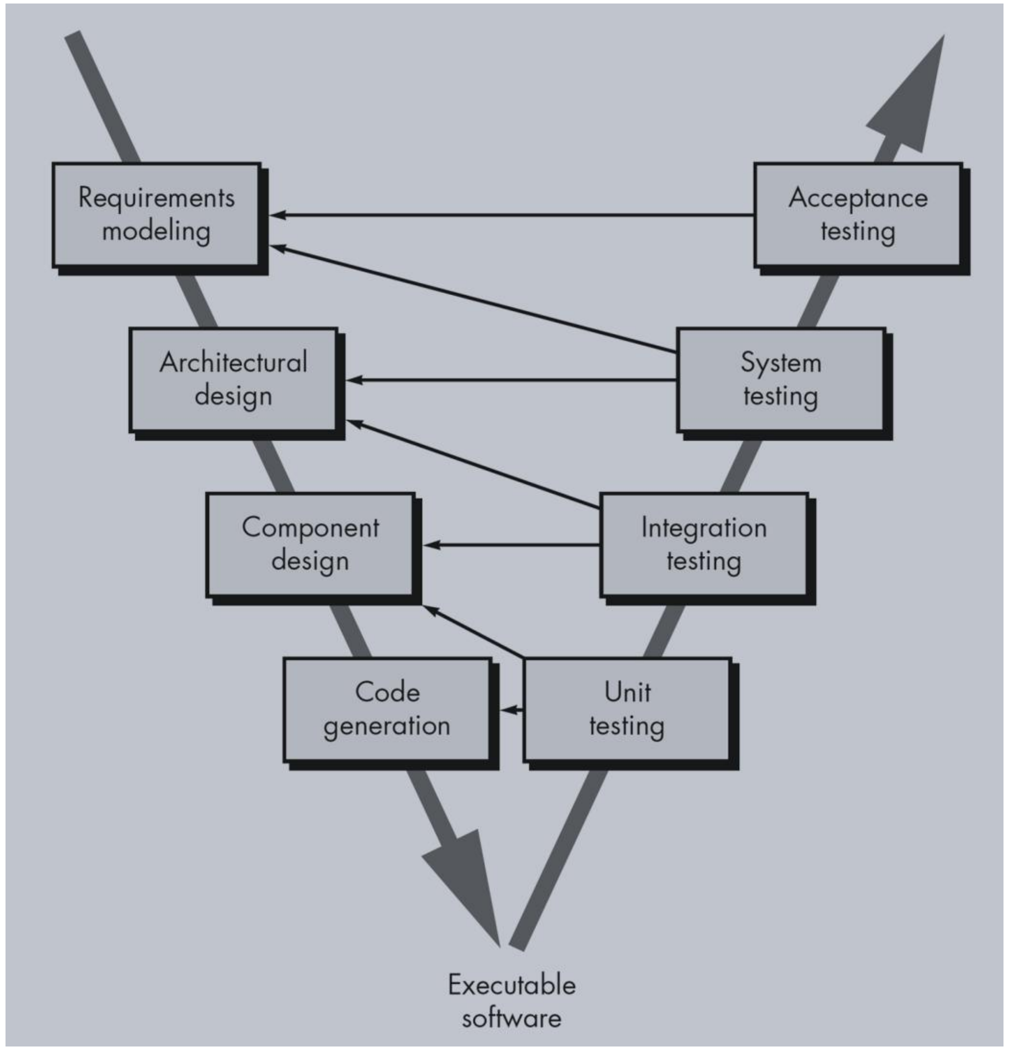
- ❑ Business Requirements <--> Acceptance Testing (Did we build the right system for the user?)
- ❑ System Requirements <--> System Testing (Does the whole system work as specified?)
- ❑ Architectural Design <--> Integration Testing (Do the modules work together?)
- ❑ Module Design <--> Unit Testing (Does each unit work correctly?)

解码V图
Decoding the V-Diagram

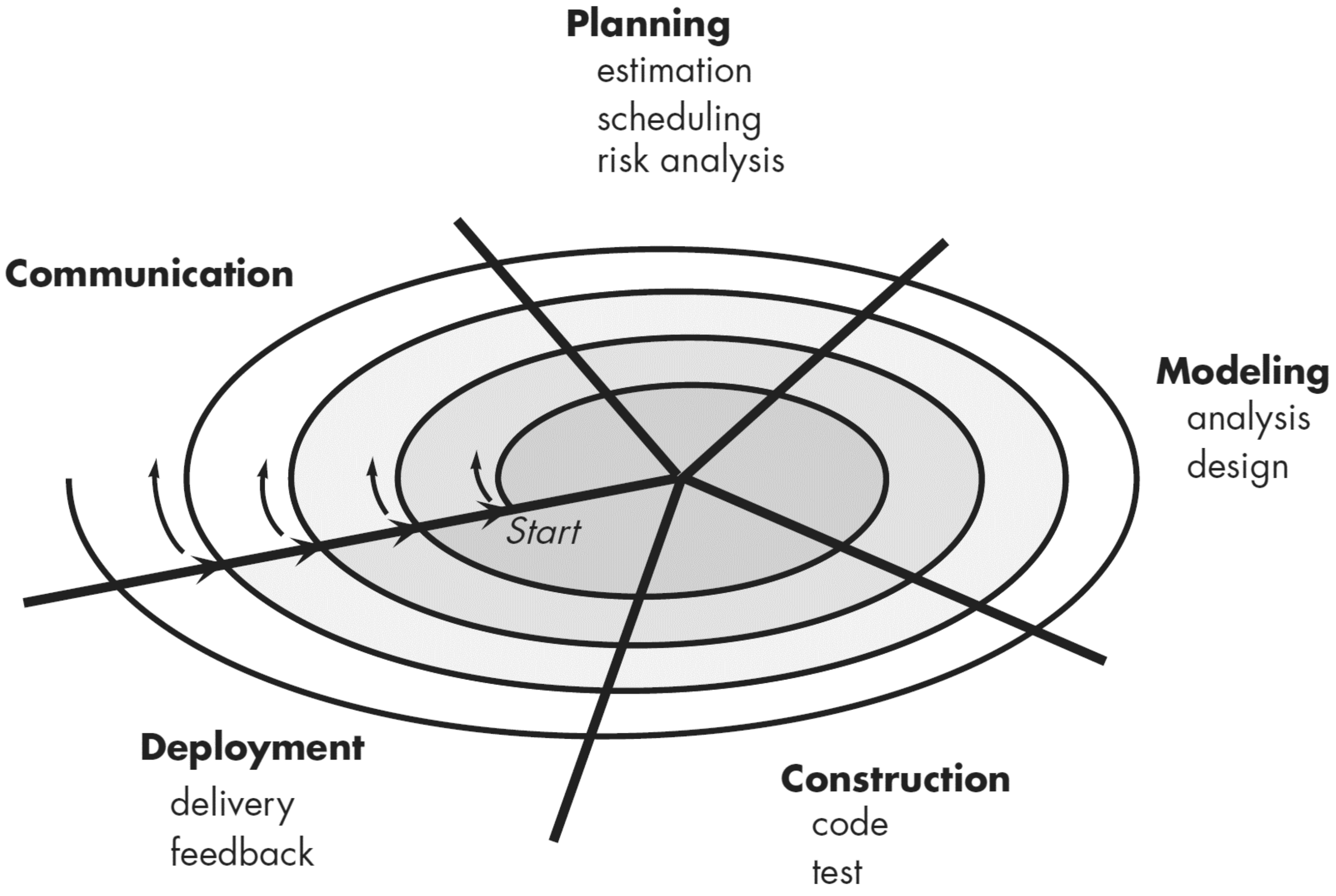
V图结构
并行测试规划

V-Diagram Structure

Parallel Testing Planning



Source:
SoftwareEngineering_a_PractitionersApproach_by_Roger_S._Pressman



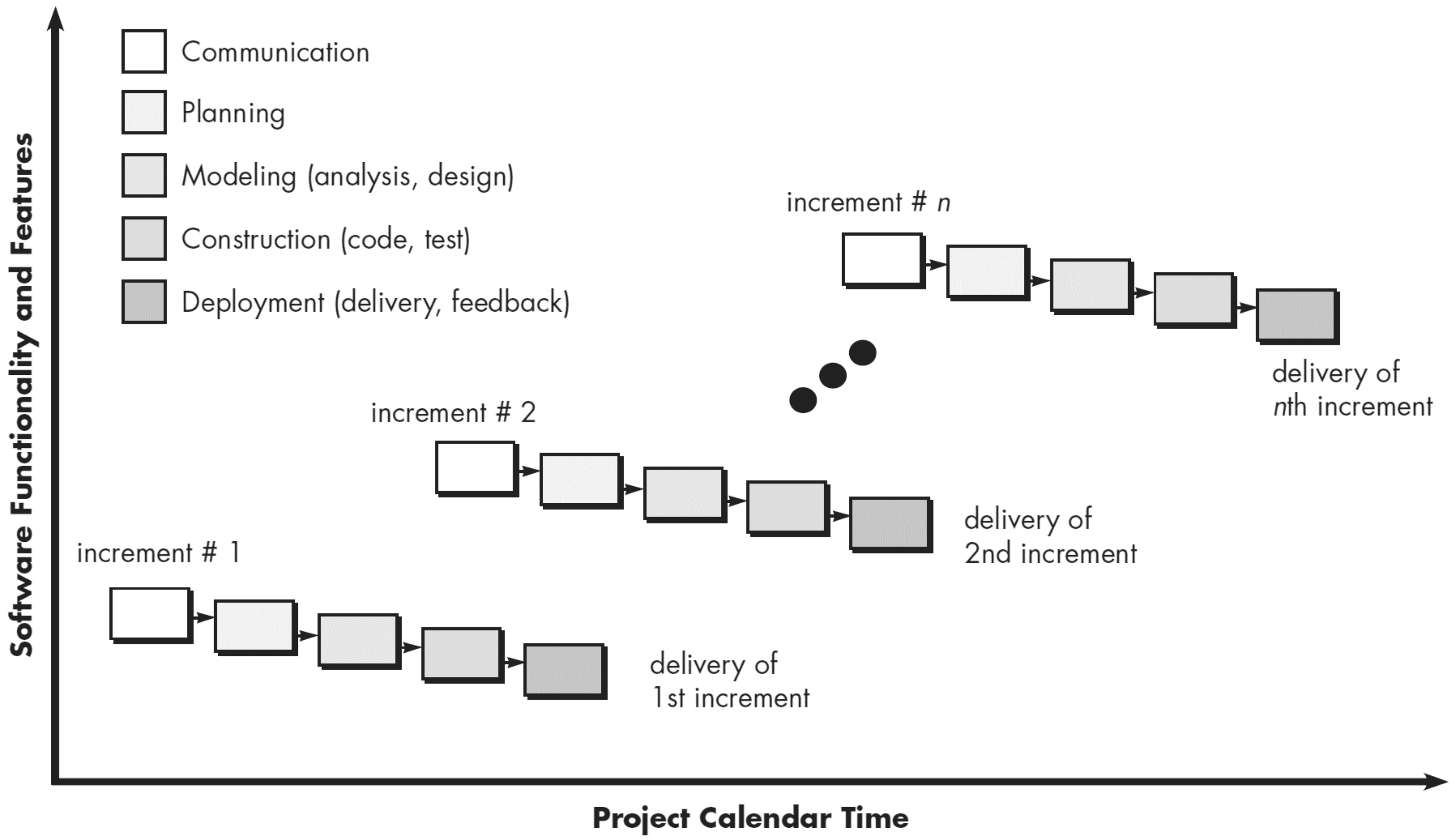
Source: SoftwareEngineering_a_PractitionersApproach_by_Roger_S._Pressman

The Spiral Model [螺旋模型]

Core Philosophy: "Address the biggest risks first." Development occurs in a series of iterative cycles (spirals), each designed to identify and mitigate the most significant risks remaining in the project.

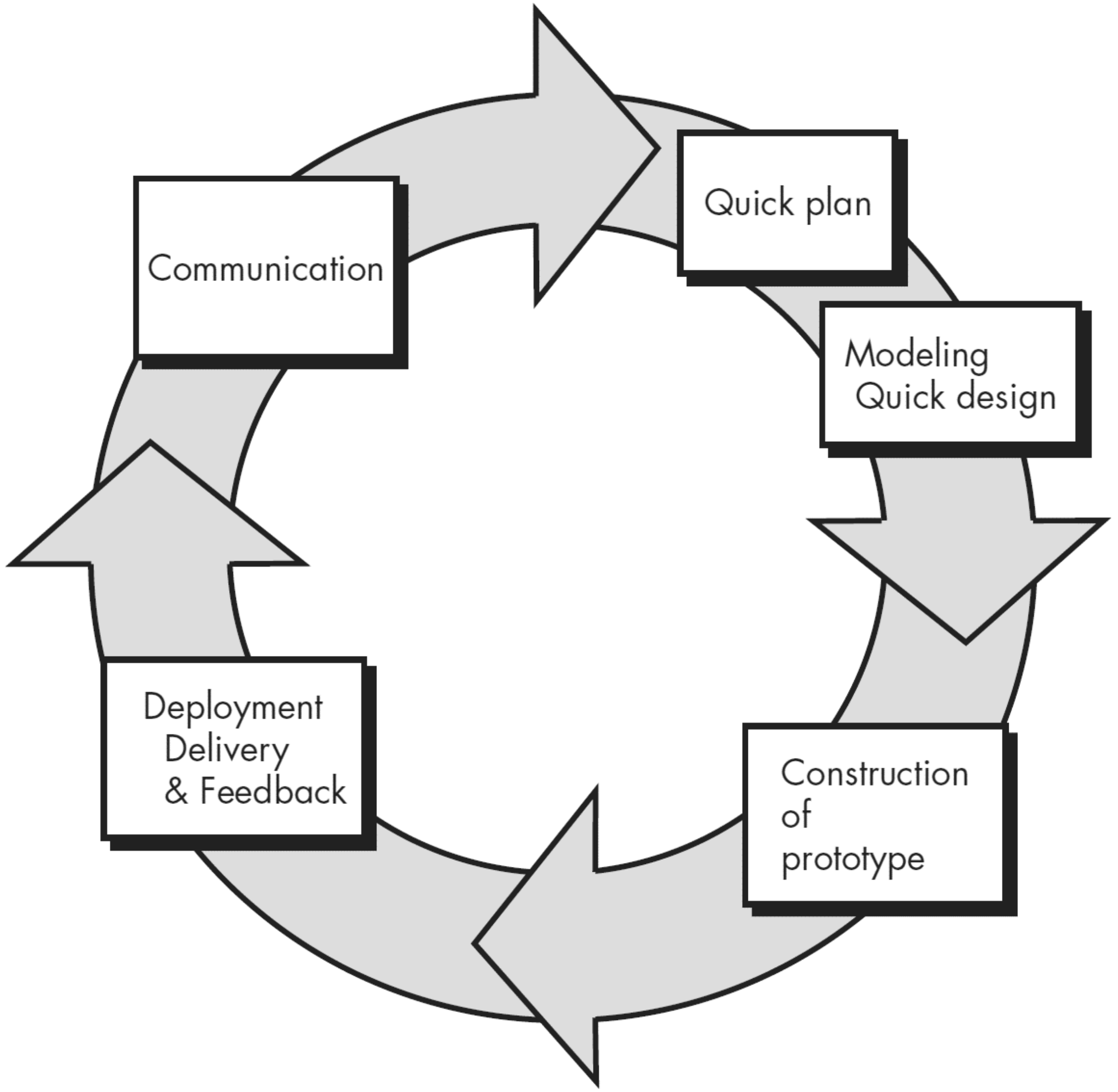
The Process (Each Loop has 4 Steps):

- ❑ Identify Objectives & Constraints: What do we want to achieve in this cycle?
- ❑ Identify & Resolve Risks: What could prevent us? (e.g., technical risk, usability risk). Use prototyping, research, or simulation to resolve it.
- ❑ Develop & Test: Build and test the deliverables for this cycle.
- ❑ Plan the Next Cycle: Review and plan for the next iteration.



Communication
Planning
Modeling (analysis, design)
Construction (code, test)
Deployment (delivery, feedback)
increment # n
delivery of nth increment
increment # 2
delivery of 2nd increment
increment # 1
delivery of 1st increment
Software Functionality and Features
Project Calendar Time

Source:
SoftwareEngineering_a_PractitionersApproach_by_Roger_S._Pressman



Source: SoftwareEngineering_a_PractitionersApproach_by_Roger_S._Pressman

90年代的软件危机
Software Crisis of the 90s

01

Industry Frustration

By the 1990s, the software industry faced a crisis. Projects were consistently late, over budget, and failed to meet user needs.

02

Us vs. Them Dynamic

A divide emerged between business stakeholders and developers. Business felt developers were slow and unresponsive, while developers felt business requirements were unclear and constantly changing.

03

Document-Heavy Models

Heavy, document-centric models like Waterfall and V-Model enforced this separation, leading to slow communication and formal change processes.

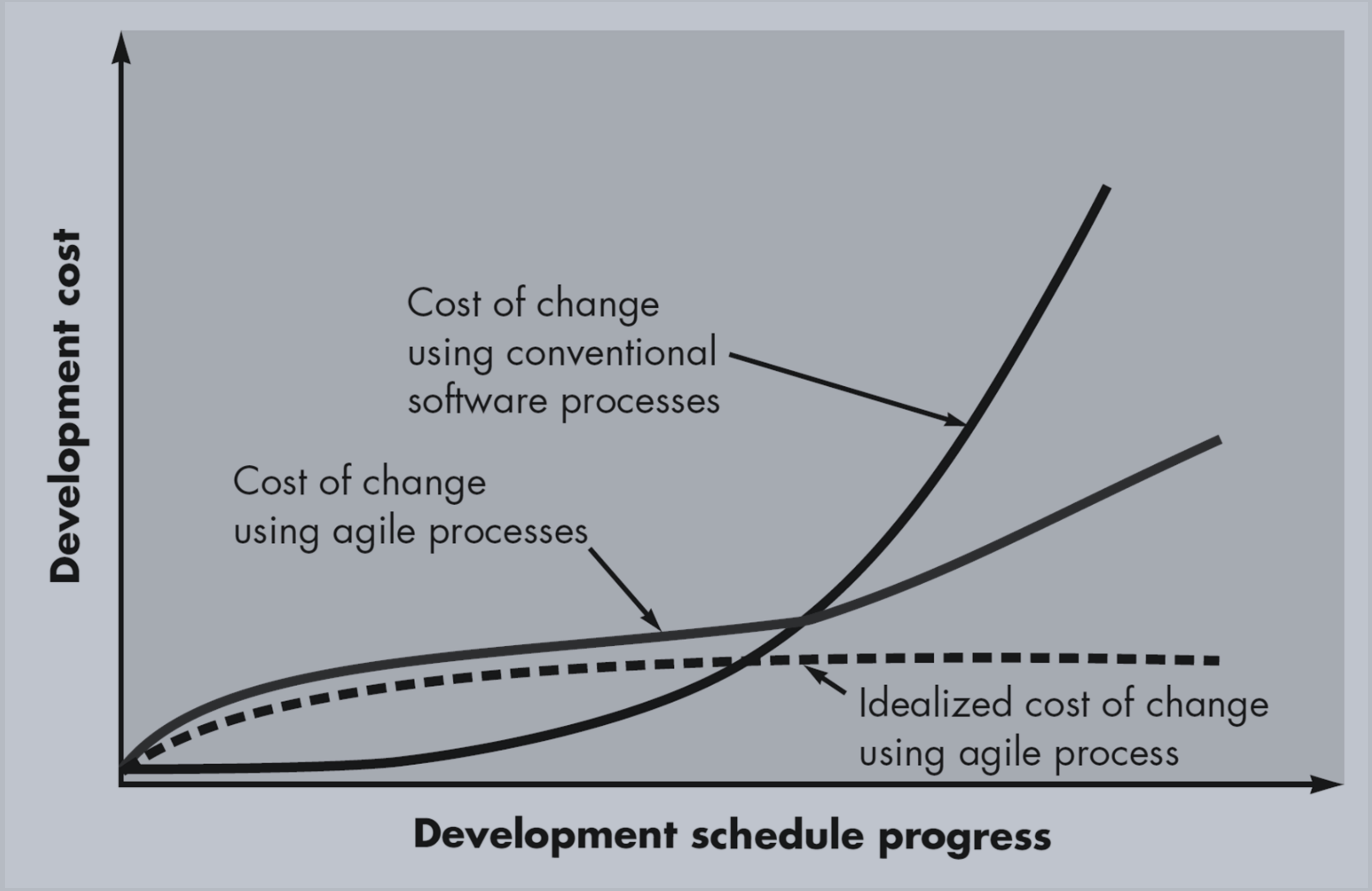
2001 Revolution [2001年革命]

Agile Manifesto Birth

In 2001, a group of leading practitioners met, to find a better way. The result was the Agile Manifesto, a statement of values for software development.

Philosophical Shift

The Agile Manifesto is not a methodology but a philosophy. It emphasizes individuals and interactions, working software, customer collaboration, and responding to change.



文档中可用的代码
Working Code Over Docs

Working Software Priority

Agile values working software over comprehensive documentation. The true measure of progress is delivering value to the user through functional software.

敏捷与迭代模型（Scrum，XP）
Agile & Iterative Models (Scrum, XP)

Core Philosophy: "Responding to change over following a plan." Deliver small, working pieces of software frequently (in iterations/sprints) and adapt the plan based on continuous feedback.

The Process (using Scrum as an example):

- ❑ Work is prioritized in a Product Backlog (a living list of requirements, often as User Stories).
- ❑ Short, fixed-length iterations (Sprints, e.g., 2 weeks) where the team builds a "Done" increment of software.
- ❑ Daily stand-up meetings for synchronization.
- ❑ At the end of the sprint, the increment is demonstrated to the Product Owner (customer proxy) for feedback, which is used to adjust the backlog.

Advantages & Disadvantages [优缺点]

Pros:

- ❑ Highly adaptable to change,
- ❑ delivers value early and often, close customer collaboration,
- ❑ improves team morale.

Cons:

- ❑ Can be difficult to predict long-term release dates,
- ❑ requires a high degree of customer involvement,
- ❑ can lead to technical debt if not managed carefully.

Real-World Example: A Mobile App Startup (e.g., a new social media app)

Why it fit: The market is uncertain. You don't know what features users will love. You release a "Minimum Viable Product (MVP)" with just a core feature (Sprint 1). You get user feedback: "We love this, but we wish we could share photos." You prioritize that for the next sprint. A competitor launches a new feature? You can adapt your backlog and respond quickly. The plan evolves with the market.

敏捷原则一览
Principles of Agile at a Glance

01

Customer Satisfaction Through Early & Continuous Delivery

- ❑ Deliver working software in small, valuable increments
- ❑ Get real feedback early and often
- ❑ Prove progress with tangible results

Example: Instead of building an entire e-commerce site in 6 months, deliver a basic product catalog in 2 weeks, shopping cart in next 2 weeks, and payment processing after that.

01

通过早期和持续交付实现客户满意

以小而有价值的增量交付可用的软件
尽早且频繁地获得真实反馈
用切实的成果证明进步

举个例子：不是在6个月内建好整个电商网站，而是在2周内交付基础产品目录，接下来2周完成购物车，之后再进行支付处理。

敏捷原则一览
Principles of Agile at a Glance

02

Welcome Changing Requirements

- ❑ Change is inevitable - embrace it!
- ❑ Late changes can provide competitive edge
- ❑ Build systems that can adapt to new requirements

Example: A banking app initially designed for basic transactions pivots to include cryptocurrency features when market demand emerges.

02

欢迎不断变化的要求

- ❑ 变化是不可避免的—拥抱它!
- ❑ 临时变更能带来竞争优势
- ❑ 构建能够适应新需求的系统

举例：一款最初为基础交易设计的银行应用，随着市场需求出现，应用转向加入加密货币功能。

敏捷原则一览
Principles of Agile at a Glance

03

Frequent Delivery of Working Software

- ❑ Short development cycles (sprints/iterations)
- ❑ Regular, predictable delivery rhythm
- ❑ Reduces risk and provides constant validation

Example: Two-week sprints where each sprint delivers a potentially shippable product increment.

03

频繁交付工作软件

- ❑ 短开发周期（冲刺/迭代）
- ❑ 规律且可预测的分娩节奏
- ❑ 降低风险并持续提供验证

示例：两周冲刺，每个冲刺交付一个潜在可发货的产品增量。

敏捷原则一览
Principles of Agile at a Glance

04

Business & Development Collaboration

- ❑ Break down silos between technical and business teams
- ❑ Constant communication prevents misunderstandings
- ❑ Shared ownership of outcomes

Example: Product owner sits with development team, participating in daily stand-ups and available for immediate clarification.

04

商业与发展合作

- ❑ 打破技术团队与业务团队之间的壁垒
- ❑ 持续沟通可以防止误解
- ❑ 结果的共享所有权

示例：产品负责人与开发团队坐在一起，参加每日站会，随时提供澄清。

敏捷原则一览
Principles of Agile at a Glance

05

Motivated Individuals & Trust

- ❑ Hire good people and trust their expertise
- ❑ Provide resources and remove obstacles
- ❑ Micromanagement kills creativity and productivity

Example: Instead of detailed task assignments, give team the goal and let them self-organize to achieve it.

05

激励的个人与信任

- ❑ 雇佣优秀人才并信任他们的专业
- ❑ 提供资源并消除障碍
- ❑ 微观管理扼杀了创造力和生产力

举例：与其给团队详细的任务分配，不如给团队设定目标，让他们自我组织以实现目标。

敏捷原则一览
Principles of Agile at a Glance

06

Face-to-Face Communication

- ❑ Rich communication with immediate feedback
- ❑ Reduces misunderstandings from written communication
- ❑ Builds team cohesion and trust

Example: Co-located teams or daily video calls for remote teams instead of relying solely on email and documentation.

06

面对面交流

- ❑ 丰富的沟通与即时反馈
- ❑ 减少书面沟通中的误解
- ❑ 增强团队凝聚力和信任

例如：远程团队与团队共址或每日视频通话，而非仅依赖电子邮件和文档。

敏捷原则一览
Principles of Agile at a Glance

07

Working Software as Progress Measure

- ❑ Not documentation, not plans, not meetings
- ❑ Only delivered features count as real progress
- ❑ Tangible evidence of forward movement

Example: At sprint review, demonstrate actual working features rather than presenting status reports or Gantt charts.

07

Working Software as Progress Measure

- ❑ Not documentation, not plans, not meetings
- ❑ Only delivered features count as real progress
- ❑ Tangible evidence of forward movement

Example: At sprint review, demonstrate actual working features rather than presenting status reports or Gantt charts.

敏捷原则一览
Principles of Agile at a Glance

08

Sustainable Development Pace

- ❑ No death marches or burnout cycles
- ❑ Consistent, predictable velocity
- ❑ Quality of life enables quality of work

Example: 40-hour work weeks as norm, avoiding last-minute crunches that lead to bugs and employee turnover.

08

Sustainable Development Pace

- ❑ No death marches or burnout cycles
- ❑ Consistent, predictable velocity
- ❑ Quality of life enables quality of work

Example: 40-hour work weeks as norm, avoiding last-minute crunches that lead to bugs and employee turnover.

敏捷原则一览
Principles of Agile at a Glance

09

Continuous Technical Excellence

- ❑ Quality enables speed in the long run
- ❑ Refactoring and clean code aren't optional
- ❑ Technical debt slows you down eventually

Example: Dedicate time each sprint for refactoring, code reviews, and improving test coverage.

09

Continuous Technical Excellence

- ❑ Quality enables speed in the long run
- ❑ Refactoring and clean code aren't optional
- ❑ Technical debt slows you down eventually

Example: Dedicate time each sprint for refactoring, code reviews, and improving test coverage.

敏捷原则一览
Principles of Agile at a Glance

10

Simplicity & Maximizing Work Not Done

- ❑ Build only what's necessary
- ❑ Eliminate waste and unnecessary features
- ❑ Simple solutions are easier to change

Example: Build a Minimal Viable Product (MVP) first, then add features based on real user feedback rather than assumptions.

10

Simplicity & Maximizing Work Not Done

- ❑ Build only what's necessary
- ❑ Eliminate waste and unnecessary features
- ❑ Simple solutions are easier to change

Example: Build a Minimal Viable Product (MVP) first, then add features based on real user feedback rather than assumptions.

敏捷原则一览
Principles of Agile at a Glance

11

Self-Organizing Teams

- Teams know best how to do their work
- Collective ownership of solutions
- Innovation comes from empowered teams

Example: Team decides how to implement features, which technologies to use, and how to solve technical challenges.

11

Self-Organizing Teams

- Teams know best how to do their work
- Collective ownership of solutions
- Innovation comes from empowered teams

Example: Team decides how to implement features, which technologies to use, and how to solve technical challenges.

敏捷原则一览
Principles of Agile at a Glance

12

Regular Reflection & Adaptation

- ❑ Continuous improvement built into the process
- ❑ Learn from successes and failures
- ❑ Adapt processes to fit team context

Example: Sprint demonstrations where team discusses what went well, what didn't, and creates action items for improvement.

12

Regular Reflection & Adaptation

- ❑ Continuous improvement built into the process
- ❑ Learn from successes and failures
- ❑ Adapt processes to fit team context

Example: Sprint demonstrations where team discusses what went well, what didn't, and creates action items for improvement.

The Agile Spirit

Principles Over Prescription

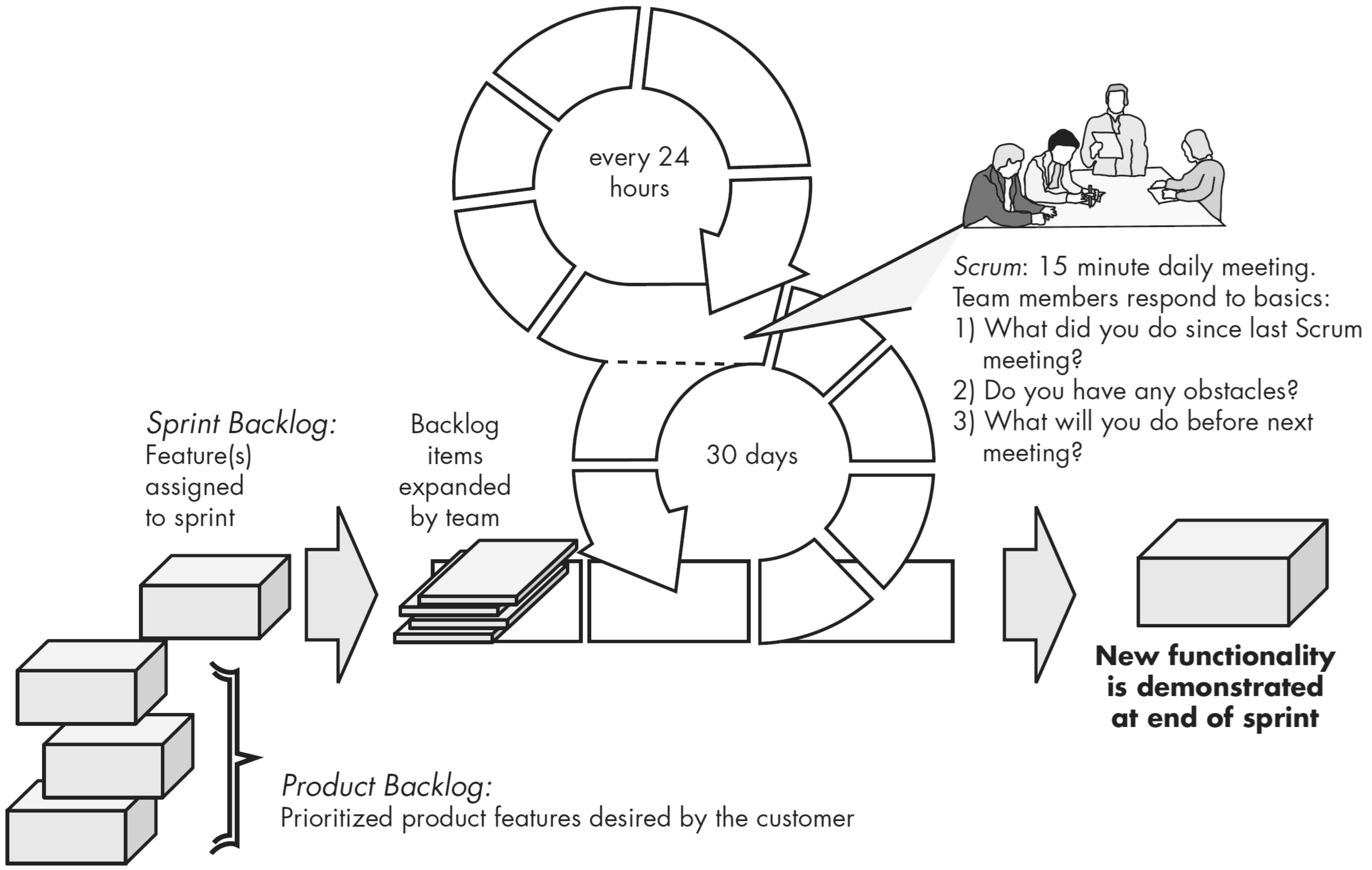
- Not all agile methods apply principles equally
- Some emphasize certain principles more than others
- Common thread: People-centric, adaptive approach
- Flexibility in application, consistency in values

Different implementations

- **Scrum:** Strong on principles 3, 7, 12
- **XP:** Strong on principles 4, 6, 9
- **Kanban:** Strong on principles 1, 8, 10

Takeaways

Scrum Framework


every 24 hours
Scrum: 15 minute daily meeting.
Team members respond to basics:
1) What did you do since last Scrum meeting?
2) Do you have any obstacles?
3) What will you do before next meeting?
Sprint Backlog:
Feature(s) assigned to sprint
Backlog items expanded by team
30 days
New functionality is demonstrated at end of sprint
Product Backlog:
Prioritized product features desired by the customer

DevOps

Core Philosophy: "Breaking down the wall between Development and Operations." It's a cultural and technical practice that emphasizes automation, collaboration, and continuous delivery.

The Process: Enabled by a CI/CD (Continuous Integration/Continuous Deployment) Pipeline.

Continuous Integration: Developers merge code changes frequently to a central repo, triggering automated builds and tests.

Continuous Delivery/Deployment: Code that passes CI is automatically deployed to staging or production environments.

Impact on Requirements: Requirements are no longer just about features. They must include Operational Requirements (or Non-Functional Requirements) like:

"The system must deploy automatically with a single click."
"The system must scale to 1 million users."
"The system must have 99.99% uptime."

现实世界的例子：Netflix

为什么适合：Netflix 需要每天部署数百次代码。手动流程是不可能的。他们的整个平台都是建立在DevOps原则之上。如果批准了新的推荐算法需求，开发者的代码变更可以被集成、测试并部署到全球CDN。关于可扩展性和弹性的运营需求从一开始就融入了每个功能中。

Real-World Example: Netflix

Why it fit: Netflix needs to deploy code hundreds of times per day. A manual process is impossible. Their entire platform is built on DevOps principles. If a requirement for a new recommendation algorithm is approved, a developer's code change can be integrated, tested, and deployed to a global CDN automatically. Operational requirements around scalability and resilience are baked into every feature from the start.

人工智能驱动的需求工程

核心理念：利用人工智能（尤其是自然语言处理——NLP）自动化RE中繁琐的部分，发掘隐藏的洞见。

流程与应用：

自动质量检查：AI工具扫描需求文档或用户故事，标记“用户友好”、“快速”或“稳健”等模糊词汇，建议更精确的替代方案。

自动化可追溯性：AI分析代码提交和测试结果，自动建议指向原始需求的链接，节省数百小时的人工工作。

需求分类：AI会自动将收到的需求标记为“功能性”、“安全性”、“性能”等。

AI-Driven Requirements Engineering

Core Philosophy: Use Artificial Intelligence (especially Natural Language Processing - NLP) to automate the tedious parts of RE and uncover hidden insights.

The Process & Applications:

Automated Quality Checking: An AI tool scans your requirements document or user stories and flags vague words like "user-friendly," "fast," or "robust," suggesting more precise alternatives.

Automated Traceability: The AI analyzes code commits and test results, automatically suggesting links back to the original requirements, saving hundreds of hours of manual work.

Requirement Classification: The AI automatically tags incoming requirements as "Functional," "Security," "Performance," etc.

现实世界的例子：一家大型汽车公司（例如，信息娱乐系统）

为什么被使用：他们有来自不同利益相关者的成千上万个需求。AI工具可以在几分钟内处理所有这些问题，识别矛盾（例如，“屏幕应始终开启”与“屏幕应在2分钟不活跃后关闭”），并确保使用一致的术语。这避免了在编写代码前出现昂贵的错误。

Real-World Example: A Large Automotive Company (e.g., for an infotainment system)

Why it's used: They have thousands of requirements from different stakeholders. An AI tool can process all of them in minutes, identifying contradictions (e.g., "The screen shall be always on" vs. "The screen shall turn off after 2 minutes of inactivity") and ensuring they use consistent terminology. This prevents costly errors before any code is written.

Model-Based Systems Engineering (MBSE)

Core Philosophy: Move from "document-based" to "model-based." Instead of thousands of pages of text, a single, precise, and executable model becomes the central source of truth for the entire system.

The Process: Engineers create formal, visual models (using languages like SysML) that represent the system's structure, behavior, and requirements. These models can be simulated and analyzed before any physical prototype is built.

Pros: Eliminates ambiguity, enables early validation through simulation, automatic generation of documents, manages extreme complexity.

Cons: Steep learning curve, requires cultural shift and new tools.

Real-World Example: NASA's Orion Spacecraft

Why it's used: The Orion spacecraft is an incredibly complex system of systems (life support, propulsion, navigation, etc.). Using MBSE, engineers create a unified model. They can simulate what happens to power requirements if the life support system runs at a higher load, or check if a software command from the navigation module correctly interfaces with the propulsion system. The model catches integration errors on a computer that would be catastrophic and expensive to find during physical testing.