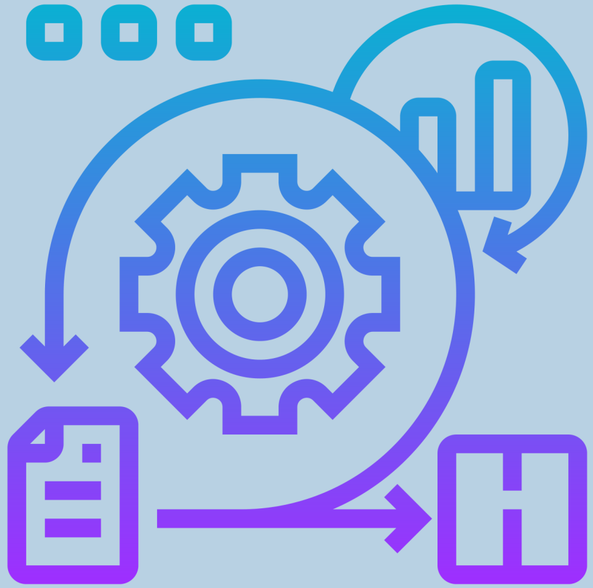
Chapter 3.
Agile Software Development

BSCS-3 Sana Izumi

Contents

- 3.1 Agile methods
- 3.2 Agile development techniques
- 3.3 Agile project management
- 3.4 Scaling agile methods



The Need for Agile in Today's Business Environment

- Businesses operate in a rapidly changing global environment.
  - Need to respond quickly to new opportunities, market changes, and competition.
  - Software is integral to business operations.
  - Rapid software development is crucial for business success.

The Need for Agile in Today's Business Environment

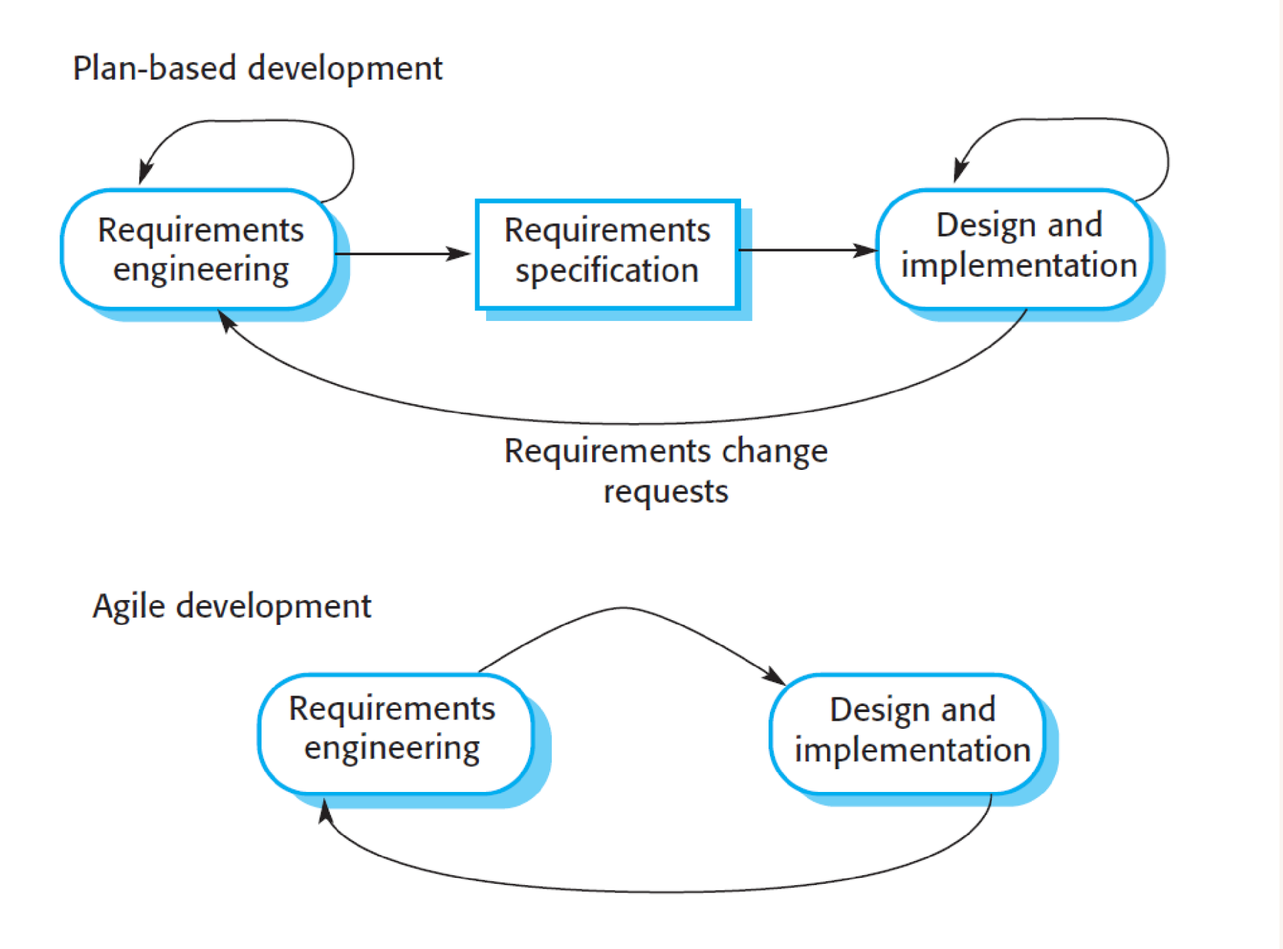
- Challenges:
  - Difficult to derive a complete set of stable software requirements.
  - Traditional plan-driven methods (e.g., waterfall) are lengthy and may not cater to changing requirements.
  - By the time software is delivered, business needs might have changed.

Rapic software development and Agile development

- Origins: Late 1990s with methods like Extreme Programming, Scrum, and DSDM (Dynamic systems development method).
- Key features:
  - Interleaved processes: No detailed system specification; minimal design documentation.
  - Incremental development: Involves end-users and stakeholders for feedback.
  - Extensive tool support: Automated testing, configuration management, UI production.

Rapic software development and Agile development

- Agile vs. Plan-Driven:
  - Agile: Design and implementation are central; informal communications; and iteration across activities.
  - Plan-Driven: Separate stages with outputs; formal documents for communication; iteration within activities.
  - In practice, a hybrid approach can be adopted, combining elements of both.



Plan-Driven Development

- Characteristics:
  - Large teams.
  - Development spans many years (e.g., aircraft control systems).
  - Emphasis on planning, design, documentation.
- Justified for: Coordinating multiple teams, critical systems, long-term maintenance.
- Overhead: Dominates the development process for small/medium systems.
- More focus on process than actual development/testing.
- Rework essential as requirements change.

Agile Methods

- Origin: Late 1990s, response to limitations of plan-driven methods.
- Focus: On the software, not excessive design/documentation.
- Benefits:
  - Rapid response to changing requirements.
  - Reduced bureaucracy and unnecessary documentation.
  - Deliver working software quickly for iterative feedback.

The 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:

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

That is, while there is value in the items on the right, we value the items on the left more.
(http://agilemanifesto.org/)

Common Principles of Agile Methods

Principle	Description
Customer involvement	Customers should be closely involved throughout the development process. Their role is provide and prioritize new system requirements and to evaluate the iterations of the system.
Embrace change	Expect the system requirements to change, and so design the system to accommodate these changes.
Incremental delivery	The software is developed in increments, with the customer specifying the requirements to be included in each increment.
Maintain simplicity	Focus on simplicity in both the software being developed and in the development process. Wherever possible, actively work to eliminate complexity from the system.
People, not process	The skills of the development team should be recognized and exploited. Team members should be left to develop their own ways of working without prescriptive processes.

- Incremental development and delivery.
- Principles based on the Agile Manifesto.
- Shared values across different agile processes.

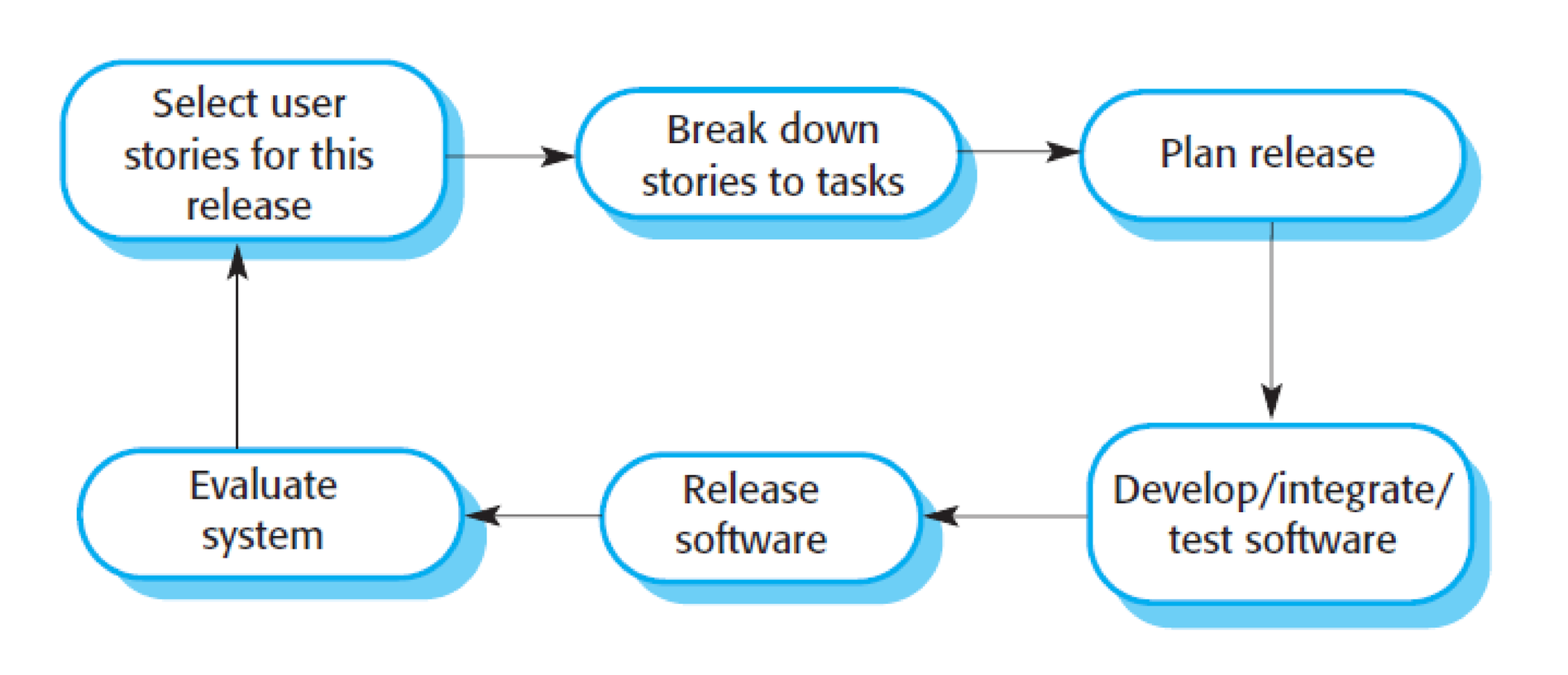
Success of Agile Methods

- Best Suited For:
  - Product development (software products, apps) for a small or medium-sized product.
  - Custom system development with committed customer involvement.
- Strengths:
  - Continuous communication between stakeholders.
  - Stand-alone systems with no need for parallel coordination.
  - Co-located teams benefiting from informal communication.

Extreme Programming (XP)

- Origin: Developed by Kent Beck in the 1990s.
- Concept: Pushing recognized good practices to "extreme" levels.
  - Multiple versions developed, integrated, and tested daily.
- Requirements: Expressed as user stories or scenarios.
- Short time gap between system releases.
- Direct implementation as a series of tasks.
- Pair programming: Two programmers work together.
- Test-first development: Tests developed before actual code.
- Continuous integration: All tests must pass when new code is added.

The XP release cycle



Key Agile Practices in XP (Based on Agile Manifesto)

1. Incremental Development:
   - Small, frequent system releases.
   - Requirements based on customer stories or scenarios.
2. Customer Involvement:
   - Continuous engagement of the customer in the team.
   - Customer defines acceptance tests.
3. People over Process:
   - Emphasis on pair programming.
   - Collective ownership of code.
   - Sustainable development without overworking.

Key Agile Practices in XP (Based on Agile Manifesto)

4. Embracing Change:
    - Regular system releases.
    - Test-first approach.
    - Refactoring and continuous integration.
5. Maintaining Simplicity:
    - Constant refactoring for code quality.
    - Simple designs without unnecessary anticipation of future changes.

Key Agile Practices in XP (Based on Agile Manifesto)

Principle or practice	Description
Collective ownership	The pairs of developers work on all areas of the system, so that no islands of expertise develop and all the developers take responsibility for all of the code. Anyone can change anything.
Continuous integration	As soon as the work on a task is complete, it is integrated into the whole system. After any such integration, all the unit tests in the system must pass.
Incremental planning	Requirements are recorded on "story cards," and the stories to be included in a release are determined by the time available and their relative priority. The developers break these stories into development "tasks." See Figures 3.5 and 3.6.
On-site customer	A representative of the end-user of the system (the Customer) should be available full time for the use of the XP team. In an extreme programming process, the customer is a member of the development team and is responsible for bringing system requirements to the team for implementation.
Pair programming	Developers work in pairs, checking each other's work and providing the support to always do a good job.

Key Agile Practices in XP (Based on Agile Manifesto)

Refactoring	All developers are expected to refactor the code continuously as soon as potential code improvements are found. This keeps the code simple and maintainable.
Simple design	Enough design is carried out to meet the current requirements and no more.
Small releases	The minimal useful set of functionality that provides business value is developed first. Releases of the system are frequent and incrementally add functionality to the first release.
Sustainable pace	Large amounts of overtime are not considered acceptable, as the net effect is often to reduce code quality and medium-term productivity.
Test first development	An automated unit test framework is used to write tests for a new piece of functionality before that functionality itself is implemented.

Challenges and Adaptations of XP

- Application of original XP proved challenging.
- Difficult to integrate with traditional business management and culture.
- Companies adapt by:
  - Selecting XP practices suitable for their operations.
  - Incorporating XP practices into their own processes or combining with other agile methods like Scrum.

3.2.1 User Stories

- Definition: User stories are scenarios of use experienced by a system user.
- Purpose: Handle changing requirements by integrating requirements elicitation with development.
- Collaboration: System customer works closely with the development team to discuss and develop story cards encapsulating customer needs.

Creating a User Story

Prescribing medication

Kate is a doctor who wishes to prescribe medication for a patient attending a clinic. The patient record is already displayed on her computer so she clicks on the medication field and can select 'current medication', 'new medication' or 'formulary'.

If she selects 'current medication', the system asks her to check the dose; If she wants to change the dose, she enters the new dose then confirms the prescription.

If she chooses 'new medication', the system assumes that she knows which medication to prescribe. She types the first few letters of the drug name. The system displays a list of possible drugs starting with these letters. She chooses the required medication and the system responds by asking her to check that the medication selected is correct. She enters the dose then confirms the prescription.

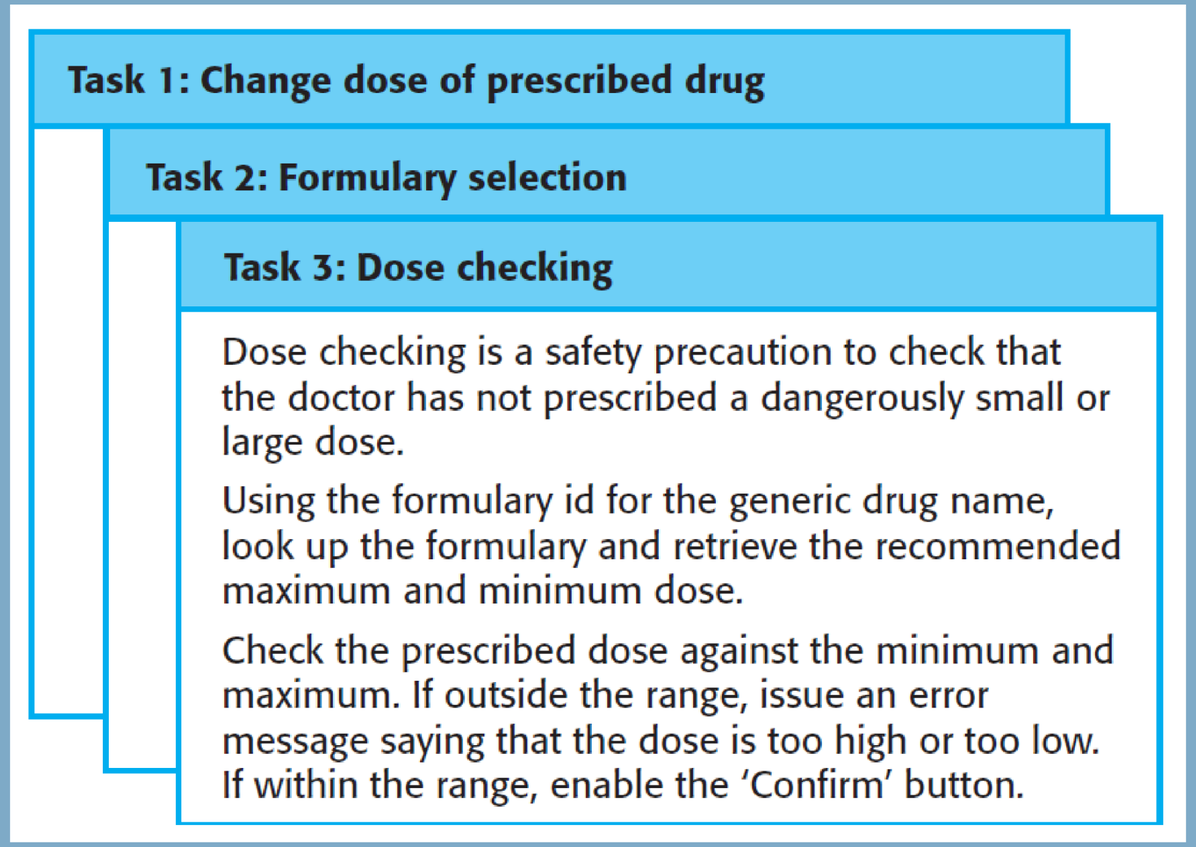
If she chooses 'formulary', the system displays a search box for the approved formulary. She can then search for the drug required. She selects a drug and is asked to check that the medication is correct. She enters the dose then confirms the prescription.

The system always checks that the dose is within the approved range. If it isn't, Kate is asked to change the dose.

After Kate has confirmed the prescription, it will be displayed for checking. She either clicks 'OK' or 'Change'. If she clicks 'OK', the prescription is recorded on the audit database. If she clicks on 'Change', she reenters the 'Prescribing medication' process.

Creating a User Story

- Process:
  - Develop story cards that describe customer needs.
  - Development team breaks stories into tasks (Figure 3.6).
  - Estimate effort and resources for each task.
  - Discuss with the customer to refine requirements.



Examples of task cards for prescribing medication

Planning and Iteration with User Stories

- Use in Planning: User stories guide system iterations.
- Prioritization: Customer prioritizes stories for immediate business support.
    - Goal: Deliver useful functionality in about two weeks (next system release).
- Handling Changes: New story cards are developed for changes in delivered systems.
    - Customer decides priority of changes vs. new functionality.

3.2.2 Refactoring

- Refactoring means that the programming team look for possible improvements to the software and implements them immediately. It involves continuously improving and optimizing code structure without changing its external behavior.
- Origin: Introduced as a key practice in Extreme Programming (XP).
- Purpose: Counteract the natural degradation of software structure during incremental development.

Traditional vs. XP Approach

- Traditional Software Engineering:
  - Design for anticipated future changes.
  - Often results in wasted effort due to unpredictability of actual changes.
- Extreme Programming (XP):
  - Avoids designing for anticipated change.
  - Focuses on immediate needs and constant refactoring.

Why Refactor?

- Address Structural Deterioration: Incremental changes can degrade software structure.
- Improve Code Readability: Makes it easier for developers to understand and modify the code.
- Remove Redundancies: Eliminate duplicate code and streamline operations.
- Enhance Flexibility: Ensure software remains adaptable to new requirements.

Examples of Refactoring

- Reorganizing class hierarchies to eliminate duplicate code.
- Renaming attributes and methods for clarity.
- Replacing similar code sections with method calls from a library.
- Utilizing development environments with built-in refactoring tools.

3.2.3 Test-First Development

- Introduced as a key practice in Extreme Programming (XP).
- A foundational aspect of XP and test-driven development (TDD).
- Running tests during code development helps discover issues early.

Key Features of Testing in XP

1. Test-First Development: Write tests before code.
2. Incremental Test Development: Based on user stories and scenarios.
3. User Involvement: Users help in test development and validation.
4. Automated Testing Frameworks: Essential for efficient and consistent testing.

Benefits of Test-First Development

- Defines interface and behavior specification implicitly.
- Reduces misunderstandings in requirements and interfaces.
- Ensures clear relationship between requirements and corresponding code.
- Avoids "test-lag" and ensures synchronization between development and testing.

Test case of Test-First Development

Relationship to User Stories:

- User stories broken down into task cards.
- Each task generates one or more unit tests.

Test 4: Dose checking

Input:
1. A number in mg representing a single dose of the drug.
2. A number representing the number of single doses per day.

Tests:
1. Test for inputs where the single dose is correct but the frequency is too high.
2. Test for inputs where the single dose is too high and too low.
3. Test for inputs where the single dose * frequency is too high and too low.
4. Test for inputs where single dose * frequency is in the permitted range.

Output:
OK or error message indicating that the dose is outside the safe range.

Test Automation and Test Challenges

- Essential for executing tests efficiently.
- Tests are stand-alone, simulate input submission, and check output specification. Automated test framework (e.g., Junit for Java) facilitates test execution.
- Benefit: Immediate detection of problems introduced by new code.

Test Coverage Challenges

- Programmer Bias
- Complex Tests
- Test Completeness/Coverage

3.2.4 Pair Programming

- Two programmers work together at the same computer to develop software.
- Introduced as a key practice in Extreme Programming (XP).
- Dynamics: Pairs are created dynamically, ensuring all team members collaborate over time.

Advantages:

- Collective Ownership & Responsibility
- Informal Review Process
- Encourages Refactoring

Productivity of Pair Programming

- Student Studies (Williams et al. 2000):
  - Productivity with pair programming comparable to two individuals.
  - Fewer false starts, less rework, and fewer errors leading to reduced testing time.
- Experienced Programmer Studies (Arisholm et al. 2007):
  - Significant loss of productivity compared to solo programming.
  - Some quality benefits, but not enough to offset the overhead.

Agile Project Management

- Agile project management addresses the need for managers to stay informed about project progress and ensure timely, budget-friendly software delivery.
- Traditional plan-driven approaches provide stability but lack visibility.

Agile vs Plan-Driven

- Early agile methods emphasized informal planning and self-organizing teams, challenging the need for visibility.
- Suitable for small companies but not larger organizations.

Scrum: Agile Project Framework

- Scrum, a framework for agile project organization, provides external visibility.
- Introduces unique terminology, e.g., ScrumMaster replaces project manager.

Scrum: Agile Project Framework

Scrum term	Definition
Development team	A self-organizing group of software developers, which should be no more than seven people. They are responsible for developing the software and other essential project documents.
Potentially shippable product increment	The software increment that is delivered from a sprint. The idea is that this should be "potentially shippable," which means that it is in a finished state and no further work, such as testing, is needed to incorporate it into the final product. In practice, this is not always achievable.
Product backlog	This is a list of "to do" items that the Scrum team must tackle. They may be feature definitions for the software, software requirements, user stories, or descriptions of supplementary tasks that are needed, such as architecture definition or user documentation.
Product owner	An individual (or possibly a small group) whose job is to identify product features or requirements, prioritize these for development, and continuously review the product backlog to ensure that the project continues to meet critical business needs. The Product Owner can be a customer but might also be a product manager in a software company or other stakeholder representative.

Scrum: Agile Project Framework

Product owner	An individual (or possibly a small group) whose job is to identify product features or requirements, prioritize these for development, and continuously review the product backlog to ensure that the project continues to meet critical business needs. The Product Owner can be a customer but might also be a product manager in a software company or other stakeholder representative.
Scrum	A daily meeting of the Scrum team that reviews progress and prioritizes work to be done that day. Ideally, this should be a short face-to-face meeting that includes the whole team.
ScrumMaster	The ScrumMaster is responsible for ensuring that the Scrum process is followed and guides the team in the effective use of Scrum. He or she is responsible for interfacing with the rest of the company and for ensuring that the Scrum team is not diverted by outside interference. The Scrum developers are adamant that the ScrumMaster should not be thought of as a project manager. Others, however, may not always find it easy to see the difference.
Sprint	A development iteration. Sprints are usually 2 to 4 weeks long.
Velocity	An estimate of how much product backlog effort a team can cover in a single sprint. Understanding a team's velocity helps them estimate what can be covered in a sprint and provides a basis for measuring improving performance.

Scrum: Agile Project Framework

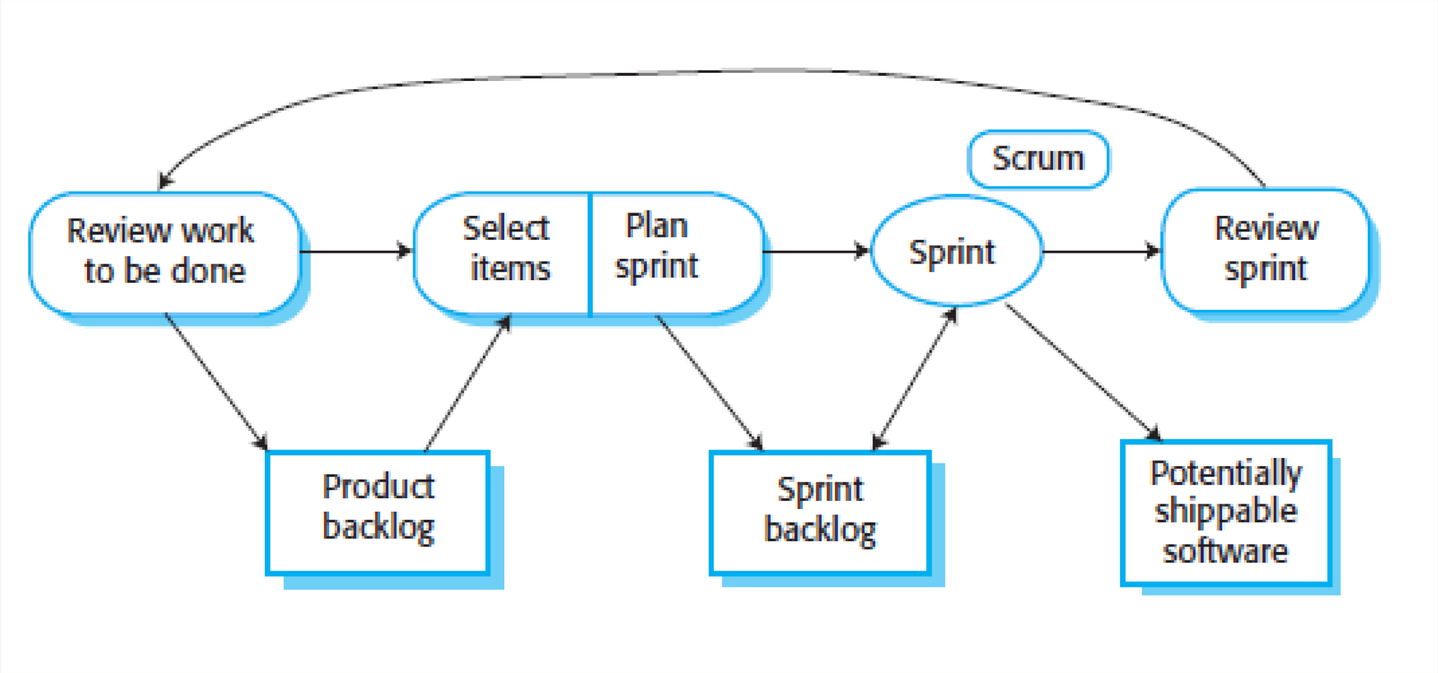
- Scrum, a framework for agile project organization, provides external visibility.
- Introduces unique terminology, e.g., ScrumMaster replaces project manager.

Characteristics of Scrum

- Follows agile manifesto principles.
- Emphasizes framework for organization rather than prescribing specific development practices.
- Easily integrates with existing company practices.
- Widely adopted in mainstream software development.

Scrum Sprint Cycle

- Product backlog is the input.
- Each iteration produces a product increment.
- Sprint duration typically 2 to 4 weeks.
- No extension of sprints for unfinished work; items return to backlog.



Product Backlog

- Contains items like product features, requirements, and engineering improvements.
- Varies in detail; Product Owner ensures appropriate specification.
- Examples: user stories or tasks like refactoring.

Sprint Planning

- Product Owner prioritizes backlog items for the upcoming sprint.
- Team selects highest-priority items they can complete.
- Estimates based on velocity from previous sprints.
- Creates a sprint backlog.

Daily Scrum Meetings

- Short daily meetings to review progress, re-prioritize, and address issues.
- Team members share progress and plan for the day.
- No top-down direction; self-organization encouraged.

Scrum Board

- A shared resource with sprint backlog, work status, staff availability, etc.
- Facilitates visibility into team activities and remaining work.

Sprint Review Meeting

- Involves the whole team for process improvement and product evaluation.
- Input for the next sprint's product backlog.

ScrumMaster Role

- Not a formal project manager but often assumes the role in traditional management structures.
- Reports on progress, involved in long-term planning, and project budgeting.
- Handles project administration and liaises with HR and procurement.

Scrum Success Stories

- User-liked aspects of Scrum:
- Manageable product chunks.
- Progress unaffected by unstable requirements.
- Enhanced team visibility, communication, and morale.
- On-time delivery with customer feedback.
- Trust-building and positive project culture.

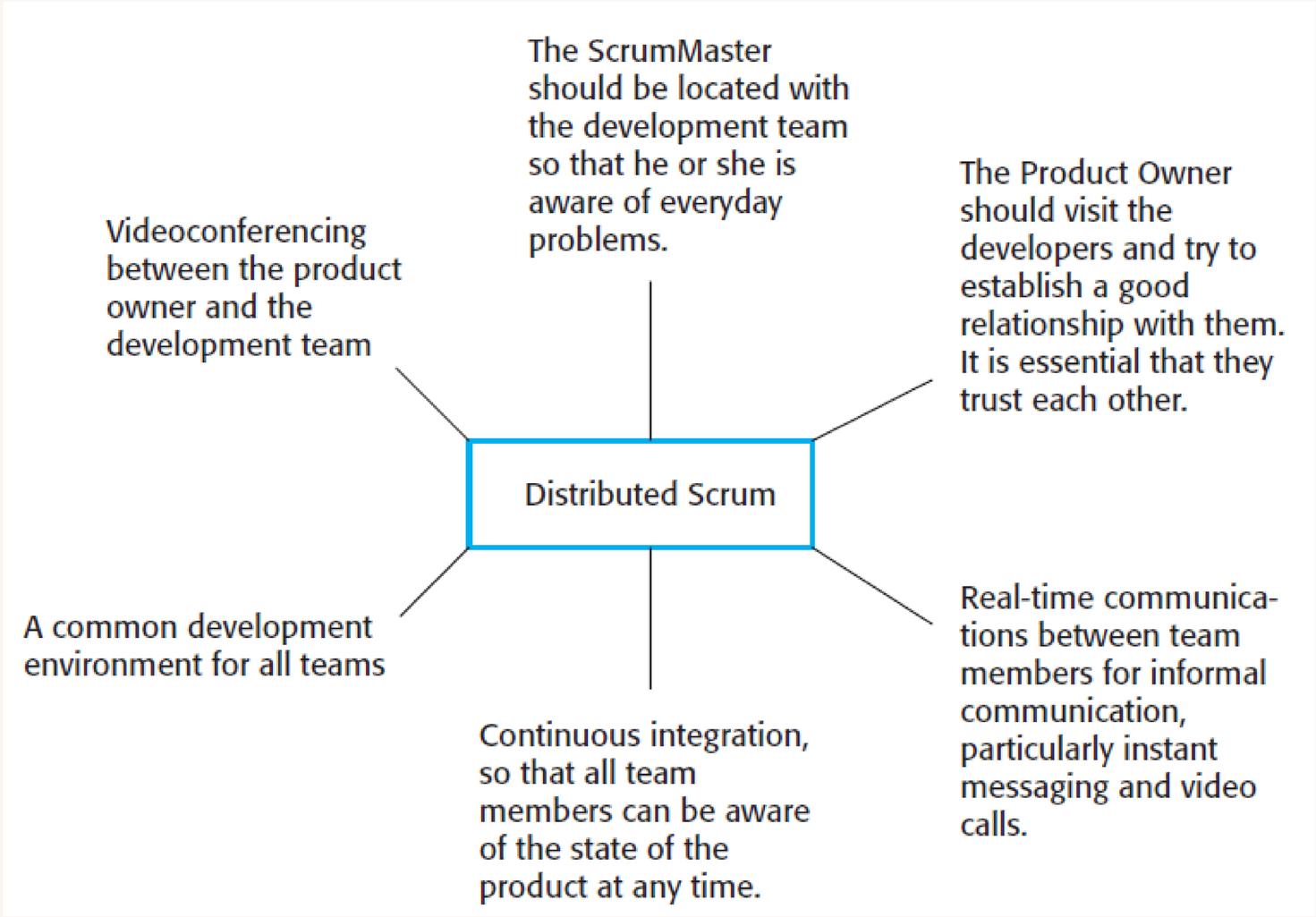
Scrum in Distributed Development

- Originally designed for co-located teams.
- Adaptations for distributed teams with members worldwide.
- Enables cost savings, access to specialized skills, and 24-hour development.

Distributed Scrum Requirements

- Considerations for Distributed Scrum (Deemer 2011)

Distributed Scrum



Scaling Agile

- Agile methods initially designed for small teams and projects.
- Need for faster, customer-centric software applies to larger systems and companies.
- Scaling agile involves two main aspects:
- Scaling up for larger systems.
- Scaling out for broader adoption in large organizations.

Challenges and Benefits

- Anecdotal evidence suggests significant productivity and defect reduction with agile methods.
- Ambler's perspective: Expect 15% productivity improvement over 3 years and fewer defects.

Practical Problems with Agile Methods

- Agile success in software products but challenges in other contexts.
- Challenges for large, long-lifetime systems:
- Incompatibility with formal contract definition.
- Less suitable for software maintenance.
- Limited for distributed development.

Practical Problems with Agile Methods

Principle	Practice
Customer involvement	This depends on having a customer who is willing and able to spend time with the development team and who can represent all system stakeholders. Often, customer representatives have other demands on their time and cannot play a full part in the software development. Where there are external stakeholders, such as regulators, it is difficult to represent their views to the agile team.
Embrace change	Prioritizing changes can be extremely difficult, especially in systems for which there are many stakeholders. Typically, each stakeholder gives different priorities to different changes.
Incremental delivery	Rapid iterations and short-term planning for development does not always fit in with the longer-term planning cycles of business planning and marketing. Marketing managers may need to know product features several months in advance to prepare an effective marketing campaign.
Maintain simplicity	Under pressure from delivery schedules, team members may not have time to carry out desirable system simplifications.
People, not process	Individual team members may not have suitable personalities for the intense involvement that is typical of agile methods and therefore may not interact well with other team members.

3.4 Scaling Agile methods

Contractual Issues

- Agile contracts based on time rather than fixed requirements.
- Potential disputes over extra time and resources.
- Agile's strength: adaptable development but uncertainty in contracts.

Agile in Software Maintenance

- Lack of product documentation challenges maintenance.
- Informal requirements collection may hinder future maintenance.
- Development team continuity matters.

3.4 Scaling Agile methods

Keeping Customers Involved

- Customer involvement during maintenance is less continuous.
- Alternative mechanisms like change proposals may be necessary.
- Challenge: Maintaining customer interest.

Maintaining Development Team Continuity

- Agile relies on implicit team knowledge.
- Agile team disruption affects understanding and productivity.
- Developers may prefer new development tasks over maintenance.

Agile and Plan-Driven Integration

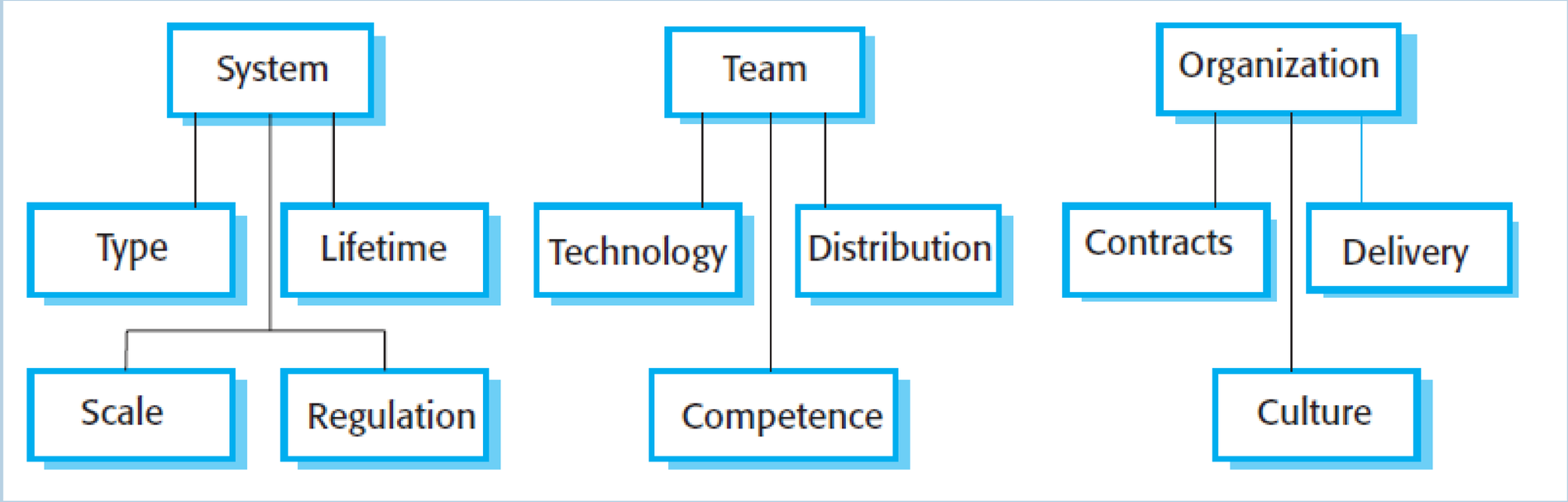
- Need to integrate agile and plan-driven approaches in large organizations.
- Finding the right balance: Technical, human, and organizational considerations.
- Key factors include system size, type, lifetime, and external regulation.

Agile and Plan-Driven Integration

Principle	Practice
Customer involvement	This depends on having a customer who is willing and able to spend time with the development team and who can represent all system stakeholders. Often, customer representatives have other demands on their time and cannot play a full part in the software development. Where there are external stakeholders, such as regulators, it is difficult to represent their views to the agile team.
Embrace change	Prioritizing changes can be extremely difficult, especially in systems for which there are many stakeholders. Typically, each stakeholder gives different priorities to different changes.
Incremental delivery	Rapid iterations and short-term planning for development does not always fit in with the longer-term planning cycles of business planning and marketing. Marketing managers may need to know product features several months in advance to prepare an effective marketing campaign.
Maintain simplicity	Under pressure from delivery schedules, team members may not have time to carry out desirable system simplifications.
People, not process	Individual team members may not have suitable personalities for the intense involvement that is typical of agile methods and therefore may not interact well with other team members.

Agile principles and organizational practice

Agile and Plan-Driven Integration



Factors influencing the choice of plan-based or agile development

System Characteristics and Agile Suitability

- Considerations for choosing between agile and plan-driven:
  - System size and co-location.
  - Analysis requirements and detailed design.
  - Expected system lifetime.
  - External regulatory requirements.

Development Team Considerations

- Skill levels in the development team.
  - How good are the designers and programmers in the development team?
- Team organization and distribution.
  - How is the development team organized?
- Importance of available development technologies.
  - What support technologies are available?

Organizational and Cultural Factors

- Impact of organization culture and management preferences.
- Compatibility with established quality procedures and standards.
- Challenges in change management and testing procedures.

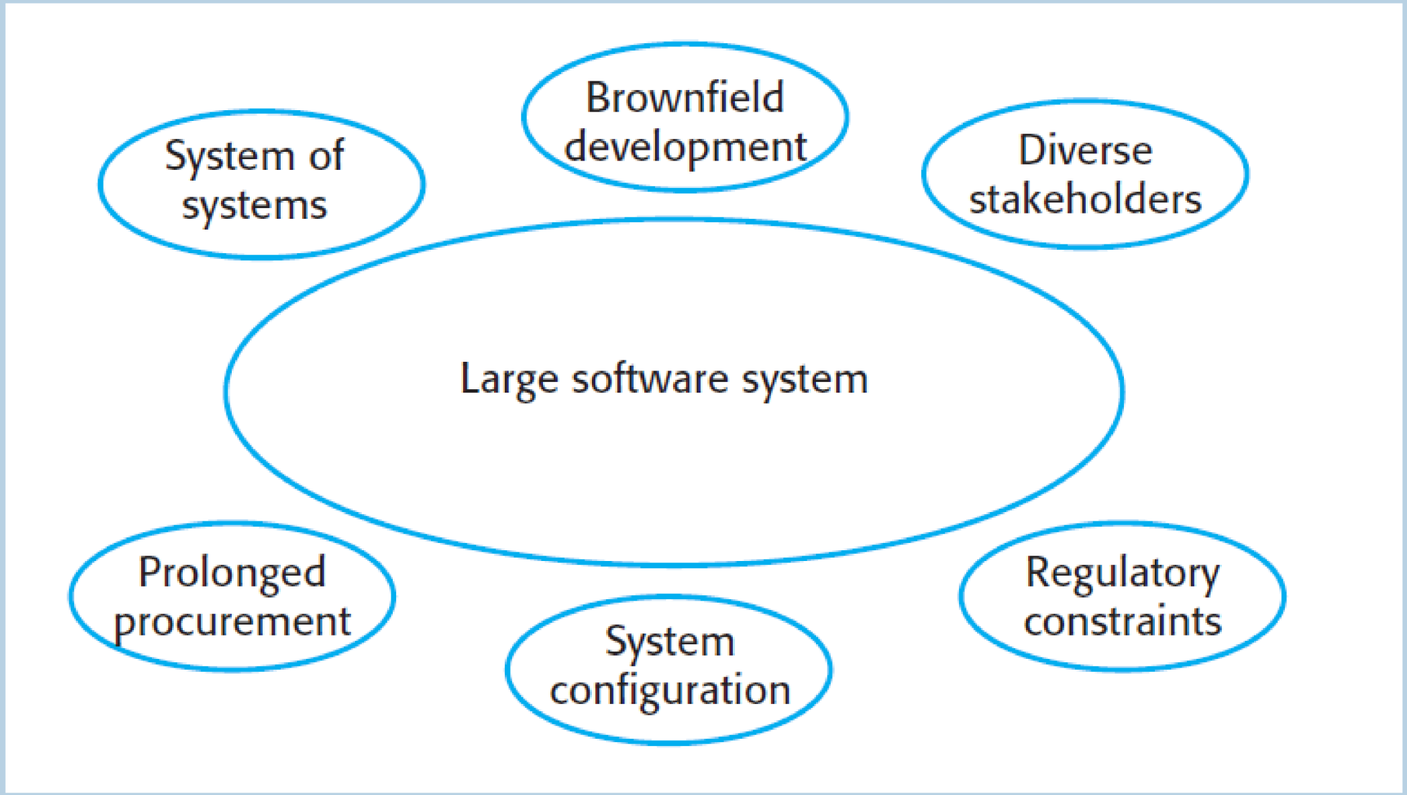
Pragmatic Approach to Method Selection

- Focus on delivering executable software that meets needs.
- Choose methods based on effectiveness, not labels (agile or plan-driven).

Agile Methods for Large Systems

- Scaling agile methods for large-scale software development.
- Complexity factors: Systems of systems, brownfield systems, system configuration, regulations, system lifetime, diverse stakeholders.

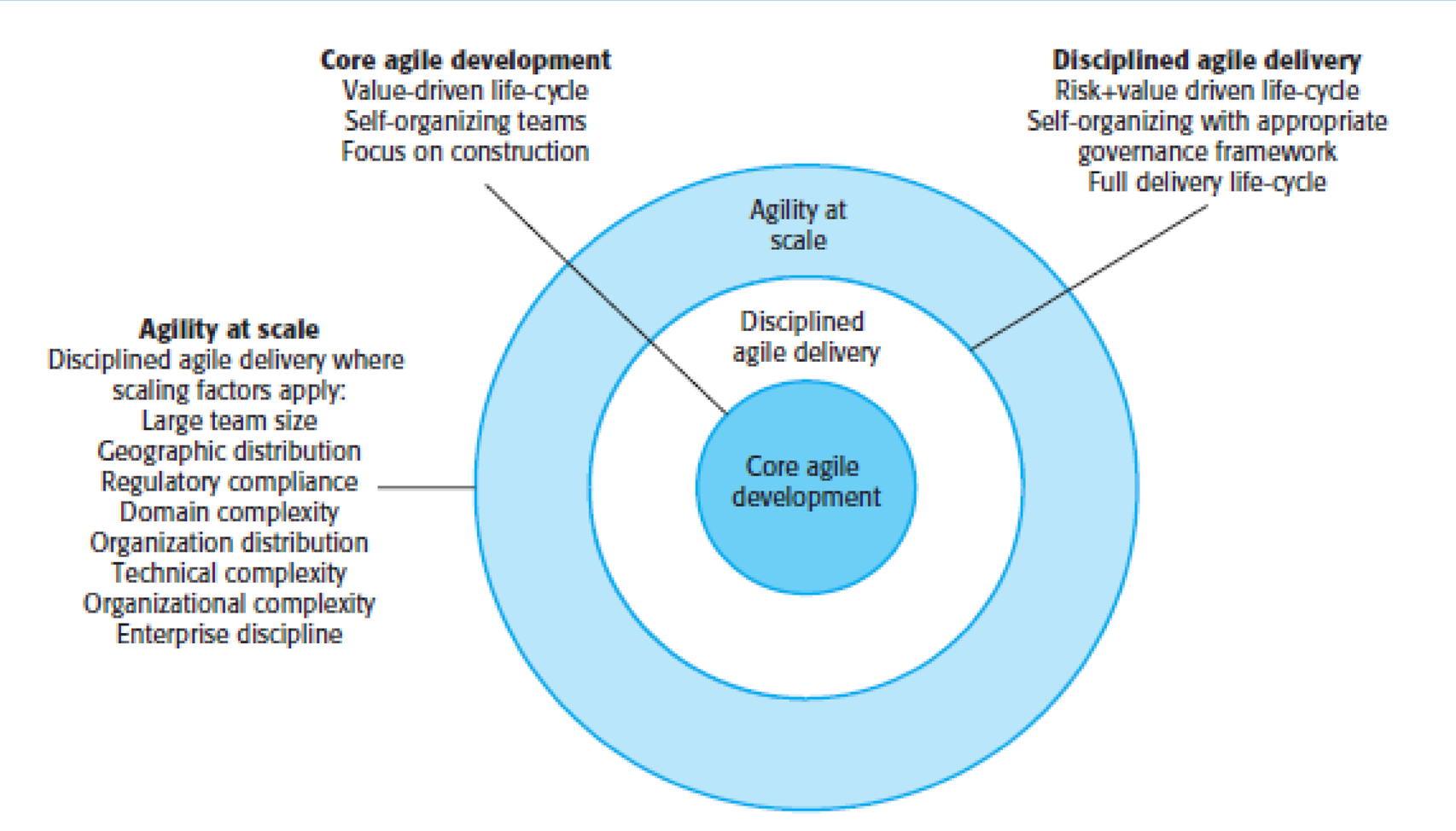
Agile Methods for Large Systems



Scaled Agile Framework (SAFe)

- Dean Leffingwell's SAFe framework for large-scale, multi-team software development.
- Role replication, product architects, release alignment, Scrum of Scrums.
- Adaptation of practices to the project context.

Scaled Agile Framework (SAFe)



Agile Methods Across Organizations

- Project managers who do not have experience of agile methods may be reluctant to accept the risk of a new approach.
- Large organizations often have quality procedures and standards that all projects are expected to follow and, because of their bureaucratic nature, these are likely to be incompatible with agile methods.
- Agile methods seem to work best when team members have a relatively high skill level. However, within large organizations, there are likely to be a wide range of skills and abilities.
- There may be cultural resistance to agile methods, especially in those organizations that have a long history of using conventional systems engineering processes.

Key Takeaways

- Agile methods offer flexibility and customer-centric development.
- Adaptation of agile methods based on project context is crucial.
- Scaling agile involves integrating plan-driven practices.
- Successful introduction of agile requires cultural change and gradual adoption