Agile software development

Mariano Ceccato

mariano.ceccato@univr.it

A bit of history

- 1980s - 1990s: higher quality software using
  - Careful project planning
  - Rigorous and controlled design and development (with tool support)
- A plan-driven development process is the correct one when a complete analysis of the system is essential
- Elaborated by a community usually involved in the development of large systems, with long life
  - Safety-critical control systems
  - Governmental and avionics/space systems
    - Modern avionics systems can take 10 years of development
  - Many distinct teams working at different parts of the system

End of 90s: new context

- Rapid development and delivery is often the most important requirement for software systems
  - Many companies would even accept a trade off between quality and fast delivery

- When applied to small/medium systems, overhead of a rigorous process may not payoff
  - Longer to decide how to develop than to develop
  - Need to reduce overheads in the software process (e.g., by limiting documentation)
  - Respond quickly to changing requirements without excessive rework

- When business operates in a continuously evolving environment
  - Impossible to obtain a complete and stable set of requirements
  - Requirements are fully clear only after the system is delivered
- This context requires a novel approach: **agile**.

Agile development

1. Program specification, design and implementation are interleaved

2. The system is developed as a series of versions or increments with stakeholders involved in version specification and evaluation
   - 2-4 weeks
   - Minimal documentation
   - Informal communication

3. Extensive tool support used to support development
   - Automated testing
   - Configuration management
   - Continuous integration
   - Automated user-interface production

Agile manifesto

Dove poniamo l'importanza		Dove era l'importanza
Individuals and interactions	over	Process and tools
Working software	over	Comprehensive documentation
Customer collaboration	over	Contract negotiation
Responding to change	over	Following a plan

$\xrightarrow{}$ Punto più importante

Source: www.agilemanifesto.org

Agile principles

• Customer involvement coinvolgimento del cliente	Customers should be closely involved throughout the development process. Their role is to provide and prioritize new system requirements and to evaluate the iterations of the system
• Incremental delivery	The software is developed in increments with the customer specifying the requirements to be included in each increment
• 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
• Embrace change	Expect the system requirements to change and so design the system to accommodate these changes
• 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

Agile method applicability

- Software company is developing a small or medium-sized product for sale
- Custom system development within an organization, with clear commitment from the customer to be involved in the development
- Few external rules and regulations that affect the software
- Agile development often in conjunction with agile management (scrum)

- **Extreme programming** → takes iterative development to the *extreme*
  - New versions may be built several times per day
  - Increments are delivered to customers every 2 weeks
  - All tests must be run for every build
  - The build is only accepted if tests run successfully

Extreme programming practices

- Incremental planning
- Small releases
- Simple design
- Test-first development
- Refactoring
- Pair programming
- Collective ownership $\longrightarrow$ responsabilità di un sorgente è di tutto il team
- Continuous integration
- Sustainable pace
- On-site customer

User stories

- Software requirements always change
- Requirements elicitation is integrated with development
- The customer works with the development team and discuss requirements with team members
- A user card briefly describes a story that encapsulates the customer needs

Mentcare: 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.

Task cards

- The team develop *task cards* based on story cards
  - Can be revise in case requirements changes
- The team estimate effort and resources for each task card
- Customer sort task cards based on priority
  - Objective: identify useful features to be developed in 2 weeks
- Completeness problem: difficult to assess if all the essential requirements are captured

Task 1: Change dose of prescribed drug

Task 2: Formulary selection

Task 3: Dose checking

Dose checking is a safety precaution to check that the doctor has not prescribed a dangerously small or large dose.

Using the formulary id for the generic drug name, look up the formulary and retrieve the recommended maximum and minimum dose.

Check the prescribed dose against the minimum and maximum. If outside the range, issue an error message saying that the dose is too high or too low. If within the range, enable the 'Confirm' button.

different priority

Test-driven development

1. Tests are written before the system code
   - Tests can be run while developing the code
   - Problems are discovered while developing
   - Development can not continue if tests do not run successfully

2. Incremental test development from scenarios
   - Relation between requirements and tests
   - Tests can be written only with clear understanding of the specification

3. User involvement in the test development and validation,
   - User helps in writing *acceptance tests* for stories

4. Automated testing frameworks
   - Previous and new tests are run automatically when new functionality is added → the new functionality has not introduced errors.

Test example: Mentcare

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.

Refactoring

- With incremental development, local changes tend to degrade the software structure
  - Future changes become harder and harder
  - Overall structure degrades as code is added
  - E.g., code duplication, inappropriate reuse of code
- When the team notices the possibility to improve the code, the code is improved, even if there is no immediate need for improvement
- **Refactoring** improves the software structure and readability
  - Reorganization of a class hierarchy
  - Removing duplicate code
  - Replacement of similar code sections with calls to methods
  - Tidying up and renaming of attributes and methods.

Pair programming

- Two developers sit the same computer
- Pairs are created dynamically, all team members work with each other

- Advantages:
  - Collective ownership and responsibility for the system
  - Informal code review
    - More simple and cheap than formal review
  - Refactoring is encouraged
    - Perceived as *inefficiency* on single programming
    - Immediate benefit for the team (collective ownership)
  - Sharing of knowledge
  - Not necessarily inefficient
    - Some evidence that a pair working together is more efficient than 2 programmers working separately

Agile project management

Agile project management

- Software managers have the responsibility that software is delivered on time and within the planned budget

- Plan-driven: Managers monitor that the project is on time with respect to the plan (time, budget, people)
- Agile project management: adapted to incremental development and the practices used in agile methods

Scrum

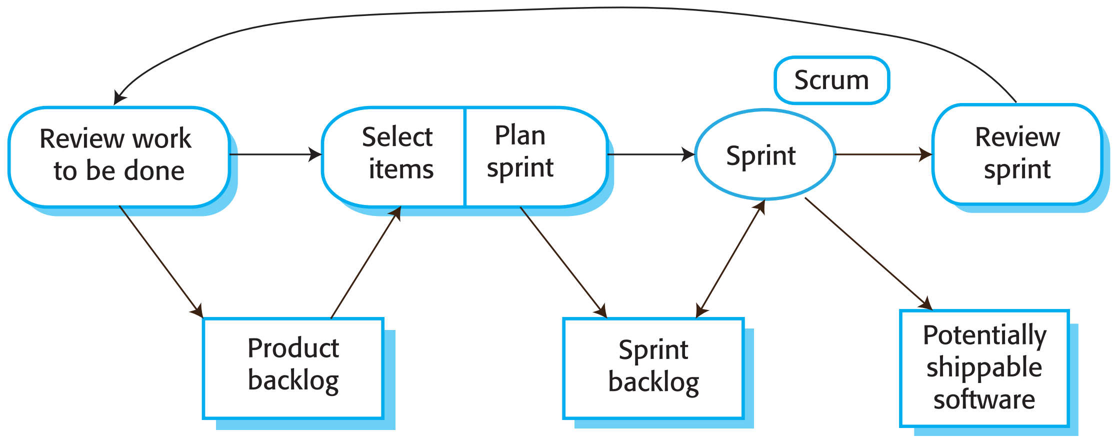
- Managing iterative development rather than specific agile practices
- Framework for organizing agile projects and provide external visibility of what is going on

- Phases:
  1. **Initial phase**: outline planning, to establish the general objectives for the project and design the software architecture
  2. Series of **sprint cycles**: each cycle develops an increment of the system.
  3. **Project closure**: phase wraps up the project, completes required documentation (e.g., user manuals) and assesses the lessons learned

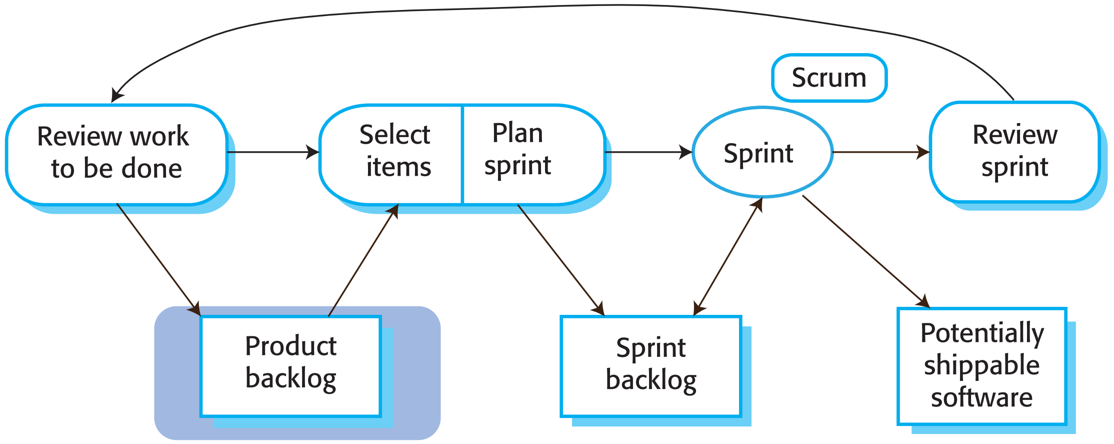
Scrum terminology

- Development team
  - Max 7 people
- Potentially shippable product increment
  - Delivered from a sprint, in a finished form
- Product backlog
  - Todo list
- Product owner
  - Identify product features, prioritize them, review product backlog
- Scrum
  - Daily meeting to review progress and prioritizes work
- ScrumMaster
  - Ensures that the Scrum process is followed
- Sprint
  - 2-4 weeks long
- Velocity
  - How much product backlog effort a team covers in a sprint

Scrum sprint cycle



Scrum sprint cycle

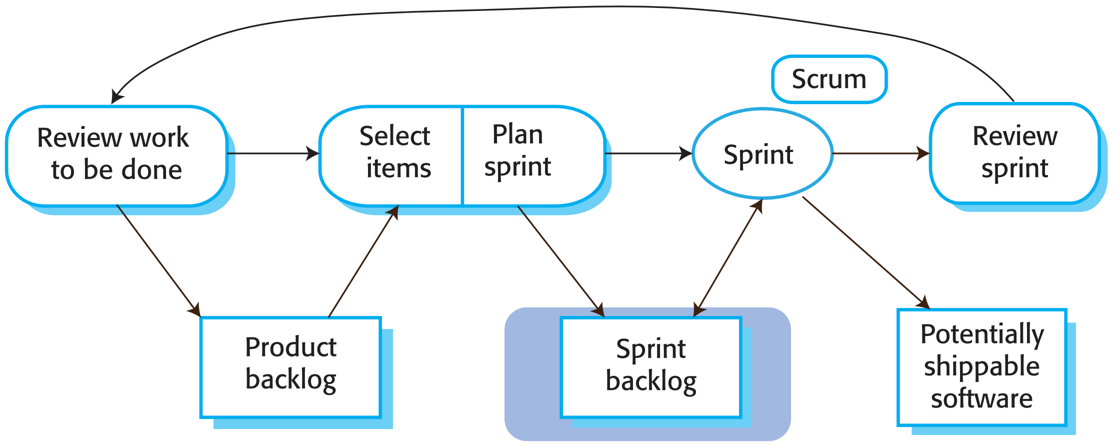


- Planning starts from the the product backlog
  - Product features, requirements, and engineering improvement
  - Initial product backlog may be derived from a requirements document, user stories, or other description of the software to be developed.
- The product owner selects the functionalities to be developed during the sprint

Scrum task board

Product backlog	Sprint backlog	In Execution	Completed	Release
Change dose of prescribed drug				
Formulary selection				
Dose checking				

Scrum sprint cycle

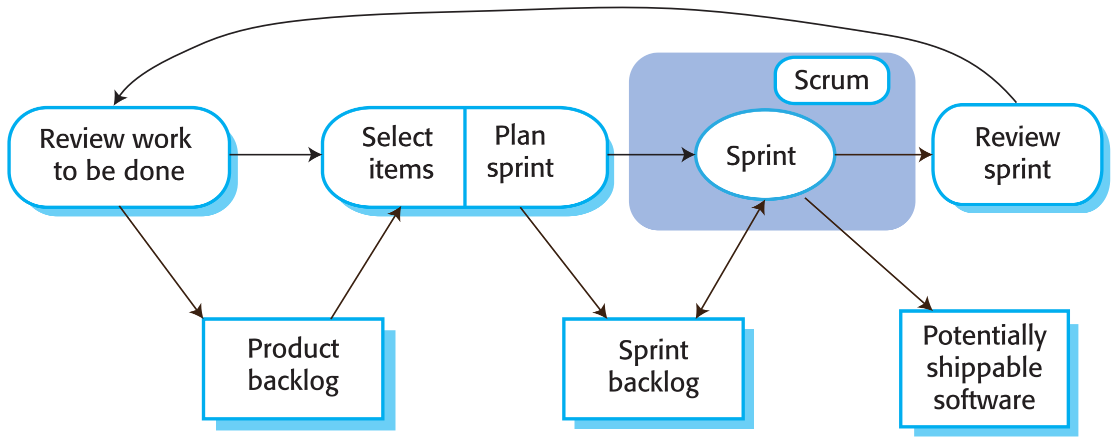


- Selecting which of the highest priority items they believe can be completed
- Estimate the time required to complete these items
  - Using velocity of previous sprints

Scrum task board

Product backlog	Sprint backlog	In Execution	Completed	Release
	Change dose of prescribed drug			
	Formulary selection			
Dose checking				

Scrum sprint cycle



- Fixed length, 2-4 weeks
  - Can not be extended
  - Team is isolated from the customer, all communications channelled through the *Scrum master*
- Short daily meetings (Scrums) to review progress and (possibly) re-prioritize work. Each team member
  - Shares information
  - Describes progresses since the last meeting
  - Brings up problems
  - States what is planned for the day
- Scrum board with note and information about the sprint backlog

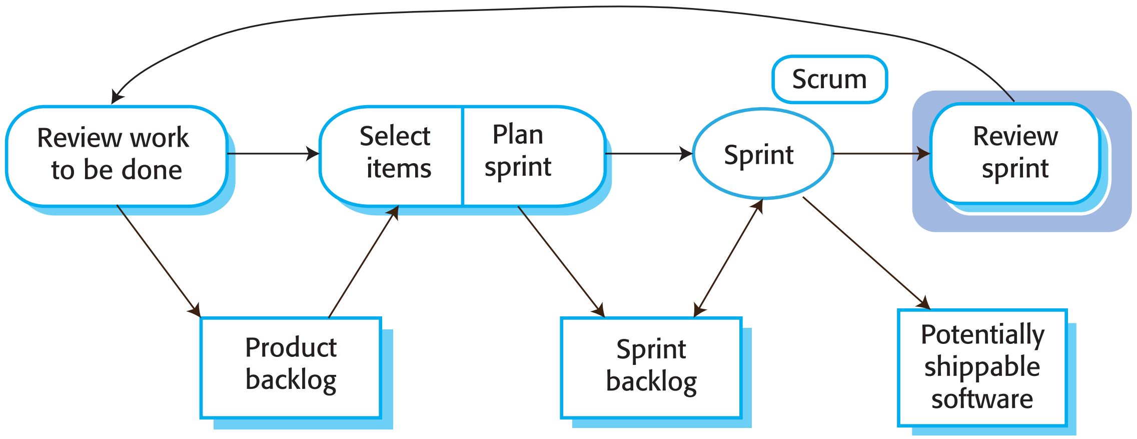
Scrum task board

Product backlog	Sprint backlog	In Execution	Completed	Release
Dose checking	Formulary selection	Change dose of prescribed drug		

Scrum task board

Product backlog	Sprint backlog	In Execution	Completed	Release
Dose checking	Formulary selection		Change dose of prescribed drug	

Scrum sprint cycle



- Process improvement
  - Team reviews the way they worked and reflects on how things could have been done better
  - Input on the product and its product state for the product backlog review (for the next sprint)

Scrum task board

Product backlog	Sprint backlog	In Execution	Completed	Release
				Change dose of prescribed drug
				Formulary selection
Dose checking				

Scrum benefits

- The product is broken down into a set of manageable and understandable chunks
- Unstable requirements do not delay development
- The whole team have visibility of everything and consequently team communication is improved
- Customers see on-time delivery of increments and gain feedback on how the product works
- Trust between customers and developers is established and a positive culture is created in which everyone expects the project to succeed

Distributed scrum

- The ScrumMaster should be located with the development team so that he or she is aware of everyday problems
- The Product Owner should visit the developers and try to establish a good relationship with them. It is essential that they trust each other
- Real-time communications between team members for informal communication, particularly instant messaging and video calls
- Continuous integration, so that all team members can be aware of the state of the product at any time
- A common development environment for all teams
- Videoconferencing between the product owner and the development team

Scaling agile methods

Scaling agile methods

- Agile methods proved to be successful for
  - Small and medium projects
  - Developed by a small co-located team
- Success coming also from improved communications, possible when everyone is working together

- Scaling up involves changing agile methods to cope with
  - Larger software projects
  - Projects that last longer
  - Multiple development teams
  - Distributed teams (working in different locations)

Scaling up/out

- **Scaling up**: developing large software systems that cannot be developed by a small team.
- **Scaling out**: introduced across a large organization with many years of software development experience.

- When scaling agile methods it is important to maintain agile fundamentals:
  - Flexible planning
  - Frequent system releases
  - Continuous integration
  - Test-driven development
  - Good team communications

Agile + maintenance

- Maintaining existing software is often more expensive than developing new software
  - Agile methods should support development and maintenance

- Two key issues:
  - Systems developed using an agile approach might be difficult to maintain
    - Emphasis on minimizing formal documentation
    - When the original team member leave, knowledge is lost
      - Issue on long-lasting projects
  - Customers should keep involved in the maintenance process.

Agile and plan-driven

- Most projects include elements of plan-driven and agile processes
- Decision based on on these considerations:
  - Is it important to have a very detailed specification and design before moving to implementation?

    → plan-driven

  - Is an incremental delivery strategy realistic? (you deliver the software to customers and get rapid feedback from them)

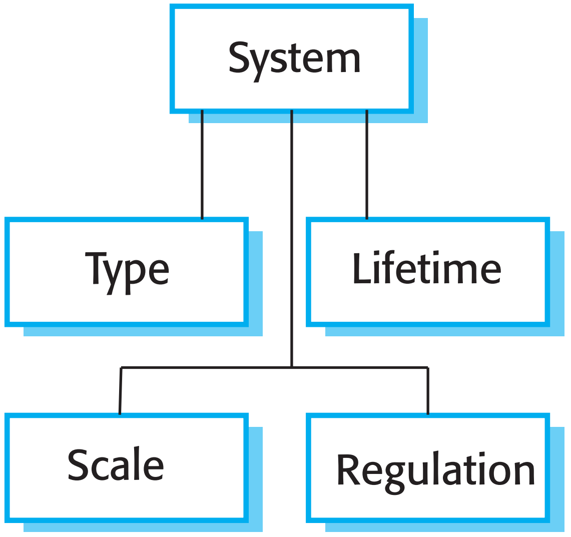
    → agile

  - How large is the system that is being developed?
    - The system can be developed with a small co-located team who can communicate informally.

      →agile
    - Difficult for large systems that require larger development teams

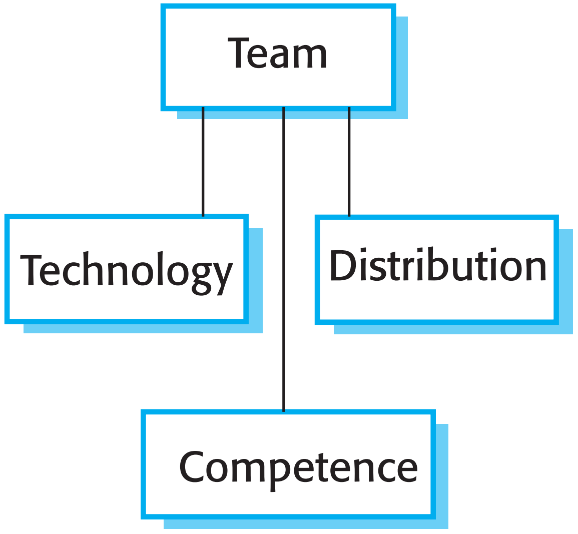
      → plan-driven

System issues



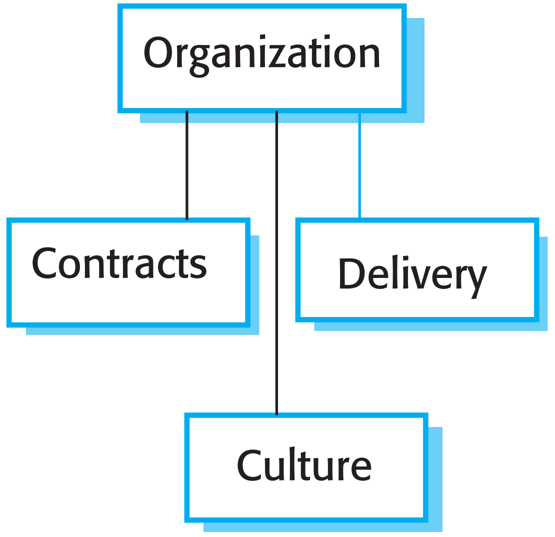
- How large is the system being developed?
  - Agile methods are most effective a relatively small co-located team who can communicate informally
- What type of system is being developed?
  - Systems that require a lot of analysis before implementation need a fairly detailed design to carry out this analysis
- What is the expected system lifetime?
  - Long-lifetime systems require documentation to communicate the intentions of the system developers to the support team
- Is the system subject to external regulation?
  - If a system is regulated you will probably be required to produce detailed documentation as part of the system safety case

People and team issues



- What support technologies are available?
  - IDE support for visualization and program analysis is essential if design documentation is not available
- How good are the designers and programmers in the development team?
  - It is sometimes argued that agile methods require higher *skill levels* than plan-based approaches in which programmers simply translate a detailed design into code
- How is the development team organized?
  - Design documents may be required if the team is distributed.

Organizational issues



- Is it a standard organizational practice to develop a detailed system specification (possibly part of the contract)?

- Can informal agile development fit into the organizational culture of detailed documentation?

- Will customer representatives be available to provide feedback of system increments?

Agile methods for large systems

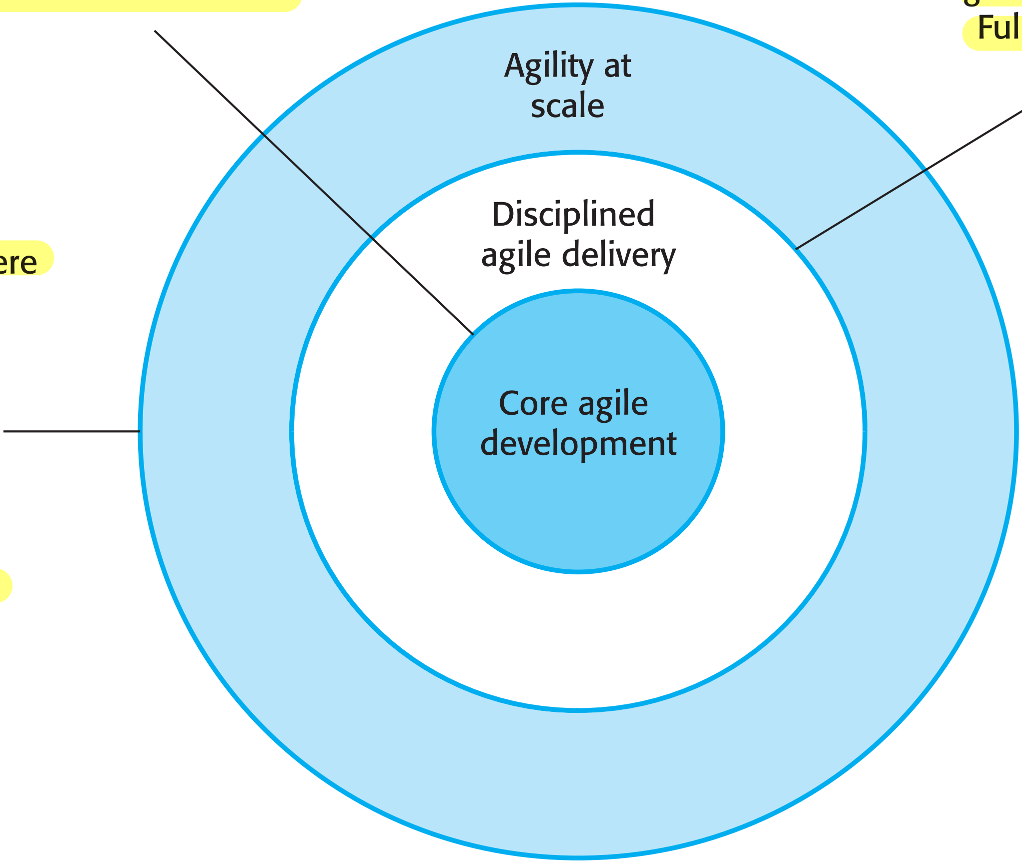
- Large-scale software systems are much more complex and difficult to understand and manage than small-scale systems
  - System of systems: developed by separated teams
  - Brownfield development: interact with a number of existing systems
  - System configuration: rather than development
  - Regulatory constraints: external to the organization
  - Prolonged procurement: key-developer might leave
  - Diverse stakeholders: impossible to involve them all

ASM: Agility at scale model

Core agile development
Value-driven life-cycle
Self-organizing teams
Focus on construction

Disciplined agile delivery
Risk+value driven lifecycle
Self-organizing with appropriate governance framework
Full delivery life-cycle

Agility at scale
Disciplined agile delivery where scaling factors apply:
Large team size
Geographic distribution
Regulatory compliance
Domain complexity
Organization distribution
Technical complexity
Organizational complexity
Enterprise discipline



IBM Agility at Scale 2010

Scaling up to large systems

- A completely incremental approach to requirements engineering is impossible
    - A preliminary requirement analysis is needed
- There cannot be a single product owner or customer representative
    - Different people are interested in different parts of the system
- For large systems development, it is not possible to focus only on the code of the system.
    - Critical parts, architecture
- Cross-team communication mechanisms have to be designed and used
- Continuous integration is practically impossible. However, it is essential to maintain frequent system builds and regular releases of the system

Multi-team scrum

- Role replication
  - Each team has a Product Owner and a ScrumMaster

- Product architects
  - Each team has a *product architect*
  - All product architects collaborate to design and evolve the overall system architecture

- Release alignment
  - The dates of product releases for all teams are aligned
  - A demonstrable and complete system is produced

- Scrum of Scrums
  - There is a daily Scrum of Scrums where representatives from each team meet to discuss progress and plan work to be done.

Resistance to agile methods

- Project managers who have <u>no experience</u> of agile methods may be reluctant to accept the risk of a new approach

- Large organizations often have <u>quality procedures and standards</u> that all projects are expected to follow and, due to their bureaucratic nature, these are likely to be incompatible with agile methods

- Agile methods seem to work best when team members have a relatively <u>high skill level</u>. However, within large organizations, there are likely to be a wide range of skills and abilities.

- There may be <u>cultural resistance</u> to agile methods, especially in those organizations that have a long history of using conventional systems engineering processes.

Tomorrow: practical group lab

- Please bring:
  - A newspaper
  - A Pen