Elements of Software Engineering and Information Systems

Lecture 3 – Agile Software Development

Prof. Ing. Lelio Campanile, PhD

Corso di Laurea Magistrale in Data Science
Dipartimento di Matematica e Fisica
Università degli Studi della Campania Luigi Vanvitelli

A.A. 2025/2026

Outline

Università degli Studi della Campania *Luigi Vanvitelli*

Topics covered

In this lecture

- Agile methods
- Agile development techniques
- Agile project management
- Scaling agile methods

Agile Methods

Rapid software development

- Rapid development and delivery is now often the most important requirement for software systems
  - Businesses operate in a fast-changing environment and it is practically impossible to produce a set of stable software requirements
  - Software has to evolve quickly to reflect changing business needs
- Plan-driven development is essential for some types of system but does not meet these business needs

Key Insight

Agile development methods emerged in the late 1990s whose aim was to **radically reduce the delivery time** for working software systems.

Università degli Studi della Campania *Luigi Vanvitelli*

Agile development

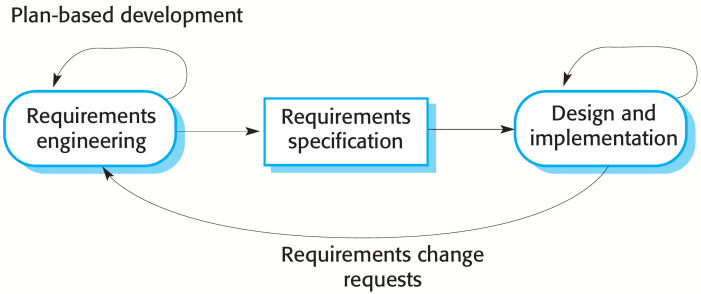
Definition

An approach where program specification, design and implementation are **inter-leaved**, the system is developed as a series of increments, and stakeholders are involved throughout.

- Frequent delivery of new versions for evaluation
- Extensive tool support (e.g. automated testing tools) used to support development
- Minimal documentation – focus on working code

Università degli Studi della Campania *Luigi Vanvitelli*

Plan-driven and agile development



Plan-driven and agile development

Plan-driven development

- Based around separate development stages with outputs planned in advance
- Not necessarily waterfall – plan-driven incremental development is possible
- Iteration occurs *within* activities

Agile development

- Specification, design, implementation and testing are inter-leaved
- Outputs are decided through negotiation during development

Università degli Studi della Campania *Luigi Vanvitelli*

Agile methods

- Dissatisfaction with the overheads involved in software design methods of the 1980s and 1990s led to the creation of agile methods. These methods:
  - Focus on the **code** rather than the design
  - Are based on an **iterative approach** to software development
  - Are intended to deliver working software quickly and evolve it quickly to meet changing requirements

Key Insight

The aim of agile methods is to reduce overheads in the software process and to be able to respond quickly to changing requirements **without excessive rework**.

Agile manifesto

We 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

Key Insight

While there is value in the items on the right, we value the items on the left **more**.

The principles of agile methods

Principle	Description
Customer involvement	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

Best suited for

Product development

Where a software company is developing a small or medium-sized product for sale. Virtually all software products and apps are now developed using an agile approach.

Custom system development

Within an organization, where there is a clear commitment from the customer to become involved in the development process and where there are few external rules and regulations that affect the software.

Agile Development Techniques

Extreme programming

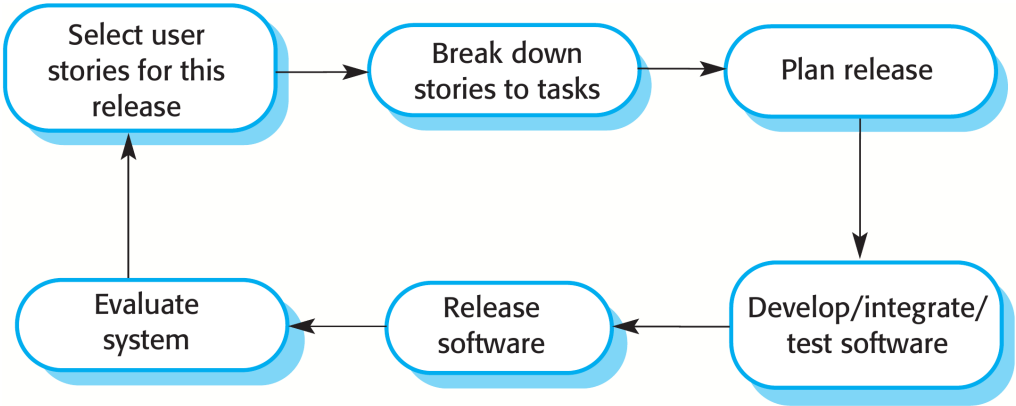
Definition

A very influential agile method, developed in the late 1990s, that introduced a range of agile development techniques.

XP takes an 'extreme' approach to iterative development:

- 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 and the build is only accepted if tests run **successfully**

The extreme programming release cycle



Extreme programming practices (a)

Principle / practice	Description
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'.
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.
Simple design	Enough design is carried out to meet the current requirements and no more.
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.
Refactoring	All developers are expected to refactor the code continuously as soon as possible code improvements are found. This keeps the code simple and maintainable.

Extreme programming practices (b)

Principle / practice	Description
Pair programming	Developers work in pairs, checking each other's work and providing the support to always do a good job.
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.
Sustainable pace	Large amounts of overtime are not considered acceptable as the net effect is often to reduce code quality and medium-term productivity.
On-site customer	A representative of the end-user of the system should be available full time for the use of the XP team. The customer is a member of the development team and is responsible for bringing system requirements to the team for implementation.

XP and agile principles

Mapping XP to agile principles

- **Incremental development** is supported through small, frequent system releases
- **Customer involvement** means full-time customer engagement with the team
- **People not process** through pair programming, collective ownership and a process that avoids long working hours
- **Change** supported through regular system releases
- **Maintaining simplicity** through constant refactoring of code

Influential XP practices

- Extreme programming has a **technical focus** and is not easy to integrate with management practice in most organizations
- Consequently, while agile development uses practices from XP, the method as originally defined is **not widely used**

Key practices

- User stories for specification
- Refactoring
- Test-first development
- Pair programming

User stories for requirements

Definition

In XP, user requirements are expressed as **user stories** or scenarios, written on cards and broken down into implementation tasks.

- A customer or user is part of the XP team and is responsible for making decisions on requirements
- These tasks are the basis of schedule and cost estimates
- The customer chooses the stories for inclusion in the next release based on their priorities and the schedule estimates

Università degli Studi della Campania *Luigi Vanvitelli*

A ‘prescribing medication’ story

Prescribing medication

The record of the patient must be open for input. Click on the medication field and select either ‘current medication’, ‘new medication’ or ‘formulary’.

If you select ‘current medication’, you will be asked to check the dose; If you wish to change the dose, enter the new dose then confirm the prescription.

If you choose, ‘new medication’, the system assumes that you know which medication you wish to prescribe. Type the first few letters of the drug name. You will then see a list of possible drugs starting with these letters. Choose the required medication. You will then be asked to check that the medication you have selected is correct. Enter the dose then confirm the prescription.

If you choose ‘formulary’, you will be presented with a search box for the approved formulary. Search for the drug required then select it. You will then be asked to check that the medication you have selected is correct. Enter the dose then confirm the prescription.

In all cases, the system will check that the dose is within the approved range and will ask you to change it if it is outside the range of recommended doses.

After you have confirmed the prescription, it will be displayed for checking. Either click ‘OK’ or ‘Change’. If you click ‘OK’, your prescription will be recorded on the audit database. If you click ‘Change’, you reenter the ‘Prescribing medication’ process.

Università degli Studi della Campania *Luigi Vanvitelli*

Examples of task cards for prescribing medication

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, lookup 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.

Refactoring

- Conventional wisdom in software engineering is to *design for change* – it is worth spending time and effort anticipating changes as this reduces costs later in the life cycle
- XP, however, maintains that this is not worthwhile as changes **cannot be reliably anticipated**

Key Insight

XP proposes **constant code improvement (refactoring)** to make changes easier when they have to be implemented.

Università degli Studi della Campania *Luigi Vanvitelli*

Refactoring

- Programming team look for possible software improvements and make these improvements even where there is no immediate need for them
- This **improves the understandability** of the software and so reduces the need for documentation
- Changes are easier to make because the code is well-structured and clear

Caveat

Some changes require **architecture refactoring** and this is much more expensive.

Examples of refactoring

Common refactoring activities

- Re-organization of a class hierarchy to remove duplicate code
- Tidying up and renaming attributes and methods to make them easier to understand
- The replacement of inline code with calls to methods that have been included in a program library

Test-first development

- Testing is **central to XP** and XP has developed an approach where the program is tested after every change has been made

XP testing features

- Test-first development
- Incremental test development from scenarios
- User involvement in test development and validation
- Automated test harnesses are used to run all component tests each time a new release is built

Università degli Studi della Campania *Luigi Vanvitelli*

Test-driven development

Key Principle

Writing tests **before code** clarifies the requirements to be implemented.

- Tests are written as *programs* rather than data so that they can be executed automatically. The test includes a check that it has executed correctly.
  - Usually relies on a testing framework such as **JUnit**
- All previous and new tests are run automatically when new functionality is added, thus checking that the new functionality has not introduced errors

Università degli Studi della Campania *Luigi Vanvitelli*

Customer involvement for testing

- The role of the customer in the testing process is to help develop **acceptance tests** for the stories to be implemented in the next release
- The customer writes tests as development proceeds – all new code is validated to ensure it is what the customer needs

Problem

People adopting the customer role have limited time available and may feel that providing the requirements was enough of a contribution, so may be **reluctant to get involved in the testing process**.

Test case description for dose checking

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

Executable test components

Test automation means that tests are written as executable components *before* the task is implemented. These testing components should be stand-alone, should simulate the submission of input to be tested and should check that the result meets the output specification. An automated test framework (e.g. JUnit) makes it easy to write executable tests and submit a set of tests for execution.

Benefit

As testing is automated, there is always a set of tests that can be **quickly and easily executed**. Whenever any functionality is added, problems that the new code has introduced can be caught immediately.

Università degli Studi della Campania *Luigi Vanvitelli*

Problems with test-first development

Problems

- Programmers prefer programming to testing and sometimes they take **short cuts** when writing tests (e.g. incomplete tests that do not check for all possible exceptions)
- Some tests can be very difficult to write **incrementally** (e.g. in a complex user interface, it is often difficult to write unit tests for the 'display logic' and workflow between screens)
- It is difficult to judge the **completeness** of a set of tests. Although you may have a lot of system tests, your test set may not provide complete coverage

Università degli Studi della Campania *Luigi Vanvitelli*

Pair programming

Definition

Pair programming involves programmers working **in pairs**, developing code together at the same workstation.

- Helps develop **common ownership** of code and spreads knowledge across the team
- Serves as an **informal review** process as each line of code is looked at by more than one person
- Encourages **refactoring** as the whole team can benefit from improving the system code

Università degli Studi della Campania *Luigi Vanvitelli*

Pair programming

- Pairs are created **dynamically** so that all team members work with each other during the development process
- The sharing of knowledge that happens during pair programming is very important as it **reduces the overall risks** to a project when team members leave

Key Insight

Pair programming is not necessarily inefficient and there is some evidence that suggests that a pair working together is **more efficient than 2 programmers working separately**.

Agile Project Management

Agile project management

- The principal responsibility of software project managers is to manage the project so that the software is delivered **on time** and **within budget**
- The standard approach to project management is **plan-driven**. Managers draw up a plan for the project showing what should be delivered, when and who will work on it

Key Insight

Agile project management requires a **different approach**, adapted to incremental development and the practices used in agile methods.

Scrum

Definition

Scrum is an agile method that focuses on **managing iterative development** rather than specific agile practices.

Three phases in Scrum

Initial phase

An outline planning phase where you establish the general objectives and design the software architecture.

Sprint cycles

A series of cycles where each cycle develops an increment of the system.

Project closure

Wraps up the project, completes required documentation such as system help frames and user manuals, and assesses the lessons learned.

Scrum terminology (a)

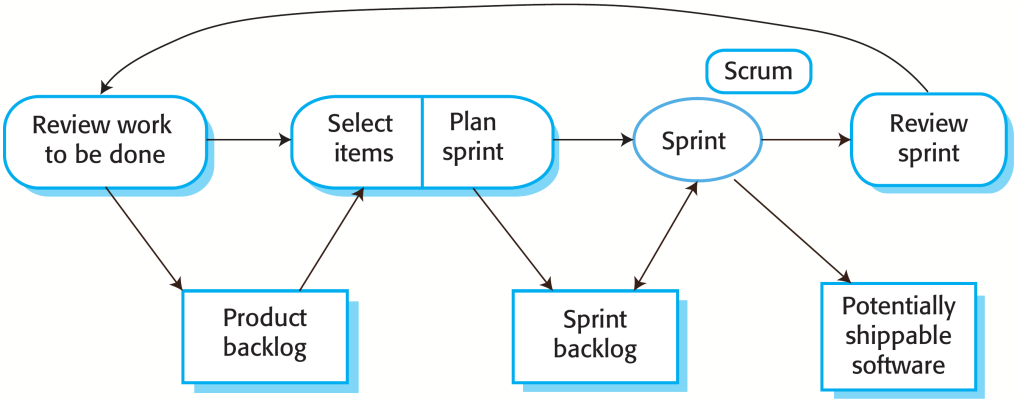
Scrum term	Definition
Development team	A self-organizing group of software developers, which should be no more than 7 people. They are responsible for developing the software and other essential project documents.
Potentially shippable product increment	The software increment that is delivered from a sprint. It should be 'potentially shippable', meaning it is in a finished state and no further work is needed to incorporate it into the final product.
Product backlog	A list of 'to do' items which the Scrum team must tackle. They may be feature definitions, software requirements, user stories or descriptions of supplementary tasks.
Product owner	An individual whose job is to identify product features or requirements, prioritize these for development and continuously review the product backlog to ensure the project continues to meet critical business needs.

Scrum terminology (b)

Scrum term	Definition
Daily 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.
Scrum Master	Responsible for ensuring that the Scrum process is followed and guides the team in the effective use of Scrum. Responsible for interfacing with the rest of the company and for ensuring that the Scrum team is not diverted by outside interference.
Sprint	A development iteration. Sprints are usually 2–4 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 performance.

Università degli Studi della Campania *Luigi Vanvitelli*

Scrum sprint cycle



The Scrum sprint cycle

Sprint planning

- Sprints are **fixed length**, normally 2–4 weeks
- The starting point for planning is the **product backlog**, the list of work to be done on the project
- The **selection phase** involves all of the project team who work with the customer to select the features and functionality to be developed during the sprint

The sprint cycle

- Once features are agreed, the team organize themselves to develop the software
- During this stage the team is **isolated from the customer** and the organization, with all communications channelled through the Scrum master

Key Insight

The role of the Scrum master is to **protect the development team from external distractions**. At the end of the sprint, the work done is reviewed and presented to stakeholders.

Teamwork in Scrum

Daily meetings

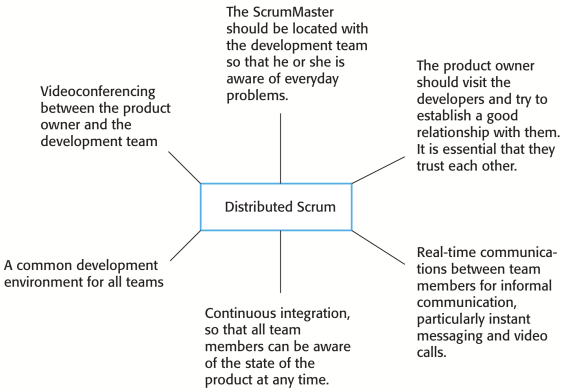
- The 'Scrum master' is a **facilitator** who arranges daily meetings, tracks the backlog, records decisions, measures progress and communicates with customers and management
- The whole team attends short daily meetings (**Scrums**) where all team members share information, describe their progress, problems and plans for the following day
  - This means that everyone on the team knows what is going on and, if problems arise, can re-plan short-term work to cope with them

Scrum benefits

Benefits

- The product is broken down into a set of **manageable and understandable** chunks
- Unstable requirements do not hold up progress
- 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

Distributed Scrum



Scaling Agile Methods

Scaling agile methods

- Agile methods have proved to be successful for **small and medium sized projects** that can be developed by a small co-located team
- It is sometimes argued that the success of these methods comes because of improved communications which is possible when everyone is working together

Challenge

Scaling up agile methods involves changing these to cope with **larger, longer projects** where there are multiple development teams, perhaps working in different locations.

Università degli Studi della Campania *Luigi Vanvitelli*

Scaling out and scaling up

Scaling up

Using agile methods for developing **large software systems** that cannot be developed by a small team.

Scaling out

How agile methods can be introduced across a **large organization** with many years of software development experience.

Key Insight

When scaling agile methods it is important to maintain agile fundamentals: **flexible planning, frequent system releases, continuous integration, test-driven development** and good team communications.

Università degli Studi della Campania *Luigi Vanvitelli*

Practical problems with agile methods

Problems

- The **informality** of agile development is incompatible with the legal approach to contract definition that is commonly used in large companies
- Agile methods are most appropriate for **new software development** rather than software maintenance. Yet the majority of software costs in large companies come from maintaining their existing software systems
- Agile methods are designed for small **co-located teams** yet much software development now involves worldwide distributed teams

Contractual issues

- Most software contracts for custom systems are based around a **specification**, which sets out what has to be implemented by the developer for the customer
- However, this precludes interleaving specification and development as is the norm in agile development

Problem

A contract that pays for developer time rather than functionality is required. However, this is seen as **high risk** by many legal departments because what has to be delivered cannot be guaranteed.

Agile methods and software maintenance

- Most organizations spend more on **maintaining existing software** than they do on new software development

Two key issues

- Are systems developed using an agile approach **maintainable**, given the emphasis on minimizing formal documentation?
- Can agile methods be used effectively for **evolving a system** in response to customer change requests?

Problems may arise if the original development team cannot be maintained.

Università degli Studi della Campania *Luigi Vanvitelli*

Agile maintenance

Key problems

- Lack of product **documentation**
- Keeping customers **involved** in the development process
- Maintaining the **continuity** of the development team

Agile development relies on the development team knowing and understanding what has to be done. For **long-lifetime systems**, this is a real problem as the original developers will not always work on the system.

Agile and plan-driven methods

Deciding on the balance

Most projects include elements of **both** plan-driven and agile processes. Deciding on the balance depends on:

- Is it important to have a very **detailed specification and design** before moving to implementation?
- Is an **incremental delivery strategy**, where you deliver the software to customers and get rapid feedback from them, realistic?
- **How large** is the system? Agile methods are most effective with a small co-located team who can communicate informally. For large systems a plan-driven approach may have to be used.

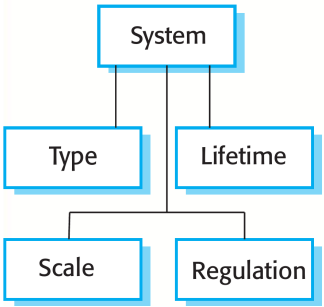
Agile principles and organizational practice (1/2)

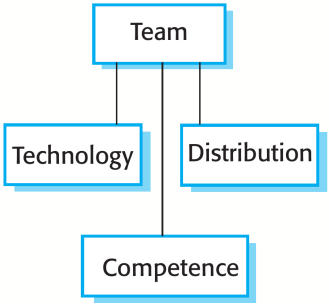
Principle	Practice
Customer involvement	This depends on having a customer who is willing and able to spend time with the development team and who can represent all system stakeholders. Customer representatives often have other demands on their time. Where there are external stakeholders such as regulators, 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 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.

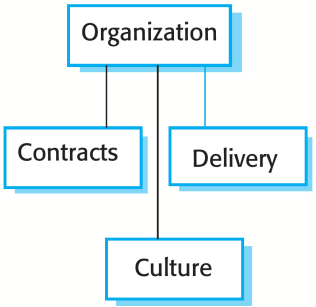
Agile principles and organizational practice (2/2)

Principle	Practice
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 and plan-based factors







System issues

Questions to consider

How large is the system?

Agile methods are most effective with 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.

What is the expected system lifetime?

Long-lifetime systems require documentation to communicate the intentions of the developers to the support team.

Is the system subject to external regulation?

If so, you will probably be required to produce detailed documentation as part of the system safety case.

People and teams

Questions to consider

How good are the designers and programmers?

Agile methods sometimes require higher skill levels than plan-based approaches.

How is the development team organized?

Design documents may be required if the team is distributed.

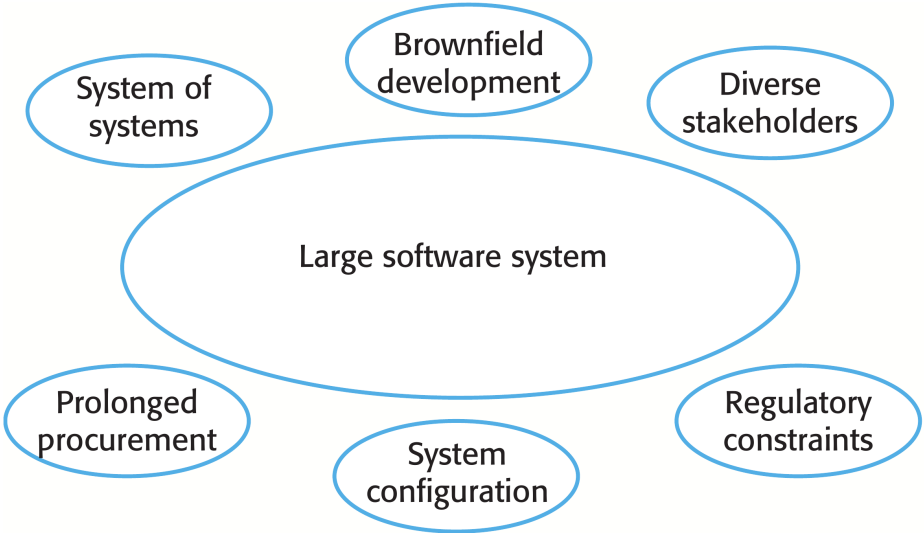
What support technologies are available?

IDE support for visualisation and program analysis is essential if design documentation is not available.

Organizational issues

- Traditional engineering organizations have a culture of **plan-based development**, as this is the norm in engineering
- Is it standard organizational practice to develop a detailed system specification?
- Will customer representatives be available to provide feedback on system increments?
- Can informal agile development fit into the organizational culture of detailed documentation?

Factors in large systems



Agile methods for large systems

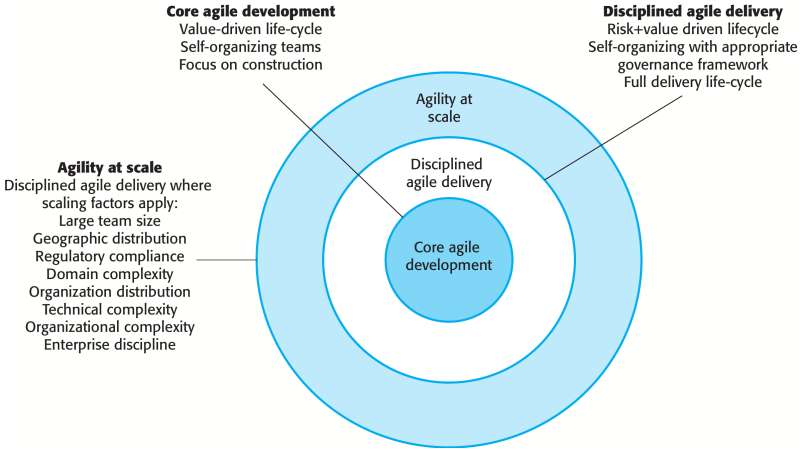
- Large systems are usually **collections of separate, communicating systems**, where separate teams develop each system, frequently working in different places and time zones
- Large systems are '**brownfield systems**' – they include and interact with a number of existing systems. Many requirements are concerned with this interaction
- Where several systems are integrated to create a system, a significant fraction of the development is concerned with system **configuration** rather than original code development

Large system development

Challenges

- Large systems and their development processes are often constrained by **external rules and regulations** limiting the way that they can be developed
- Large systems have a **long procurement and development time**. It is difficult to maintain coherent teams who know about the system over that period
- Large systems usually have a **diverse set of stakeholders**. It is practically impossible to involve all of these different stakeholders in the development process

IBM's agility at scale model



Scaling up to large systems

Limitations at scale

- A completely incremental approach to requirements engineering is **impossible**
- There cannot be a **single product owner** or customer representative
- For large systems development, it is not possible to focus only on the code
- **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

Università degli Studi della Campania *Luigi Vanvitelli*

Multi-team Scrum

Coordination mechanisms

Role replication

Each team has a Product Owner for their work component and a ScrumMaster

Product architects

Each team chooses a product architect and these architects collaborate to design and evolve the overall system architecture

Release alignment

The dates of product releases from each team are aligned so that 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

Agile methods across organizations

Barriers

- Project managers who do not have experience of agile methods may be **reluctant to accept the risk**
- Large organizations often have **quality procedures and standards** that are likely to be incompatible with agile methods
- Agile methods seem to work best when team members have a **relatively high skill level**. Within large organizations, there are likely to be a wide range of skills
- There may be **cultural resistance** to agile methods, especially in organizations with a long history of conventional systems engineering processes

Università degli Studi della Campania *Luigi Vanvitelli*

Key points

Summary

- Agile methods are incremental development methods that focus on **rapid software development**, frequent releases, reducing process overheads by minimizing documentation and producing high-quality code
- Agile development practices include:
  - User stories for system specification
  - Frequent releases of the software
  - Continuous software improvement
  - Test-first development
  - Customer participation in the development team

Key points (cont.)

Summary

- **Scrum** is an agile method that provides a project management framework
  - It is centred round a set of **sprints**, which are fixed time periods when a system increment is developed
- Many practical development methods are a mixture of **plan-based and agile** development
- Scaling agile methods for large systems is **difficult**
  - Large systems need up-front design and some documentation, and organizational practice may conflict with the informality of agile approaches

Università degli Studi della Campania *Luigi Vanvitelli*

Grazie per l'attenzione

Prof. Ing. Lelio Campanile, PhD

Assistant Professor

Dipartimento di Matematica e Fisica

Elements of Software Engineering and Information Systems

Corso di Laurea Magistrale in Data Science

Università degli Studi della Campania Luigi Vanvitelli