Agile software development

Vũ Thị Hồng Nhạn

(vthnhan@vnu.edu.vn)

Dept. of Software Engineering, FIT, UET

Vietnam National Univ., Hanoi

Contents

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

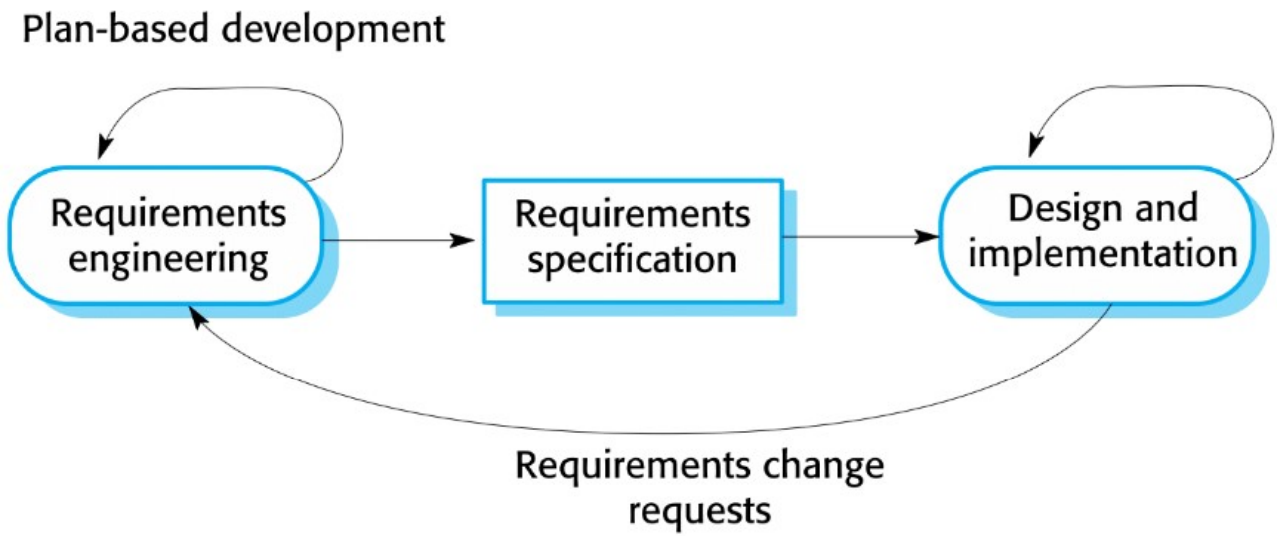
Rapid software development

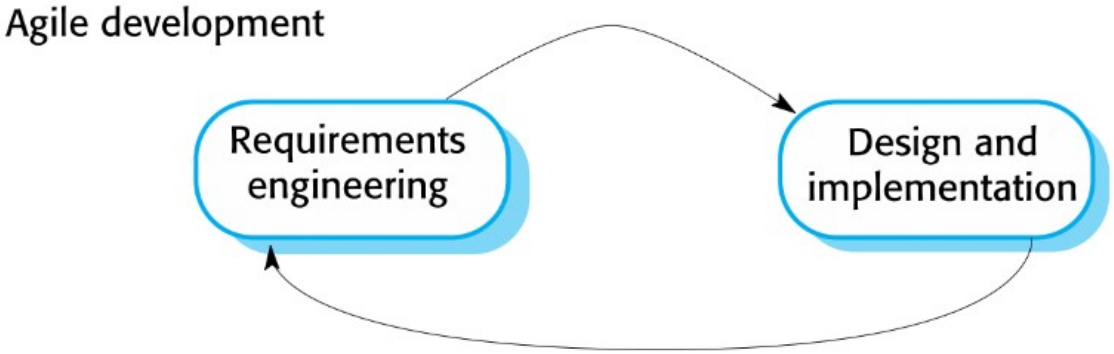
- Rapid development and delivery is now often **the most important requirement** for software systems
  - Business operate in a fast-changing environment and it is practically impossible to produce a set of **stable** software requirements
  - Software has to evolve quickly to reflect changing business needs
- Plan-driven development is essential for some types of system but does not meet the changing business needs
- **Agile development methods** emerged in the late 1990s with the aim is to deliver software systems **rapidly**

Agile development

- As mentioned, in **the Agile process** specification, design, implementation and validation are **inter-leaved**
- The system is developed as **a series of versions** or increments with **stakeholders** involved in both specification and evaluation
- In this development process, a new version is frequently delivered for evaluation
- Test-first development: it's the development and use of automated testing tools to support the development of software
- And the final objective of agile development is to achieve "Minimal documentation" – simply focus on working code

Plan-driven & agile development





Plan-driven & agile development

- Plan-driven development
  - A plan-driven approach to SE is based around separate development stages with **the outputs** to be produced at each of these stages planned **in advance**
  - Not necessarily waterfall model: plan-driven, incremental development is possible
  - **Iteration** occurs within activities
- Agile development
  - Specification, design, implementation, and testing are interleaved, and the **output** from the development process are decided through a process of negotiation during the software development process

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
  - focuses 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 without excessive rework

Agile manifesto

- A better way of developing software by doing it and helping others do it
- Through this work, value can be gained include….
  - Individuals and interactions **over** processes and tools .
  - Working software **over** comprehensive documentation.
  - Customer collaboration **over** contract negotiation.
  - Responding to change **over** following a plan

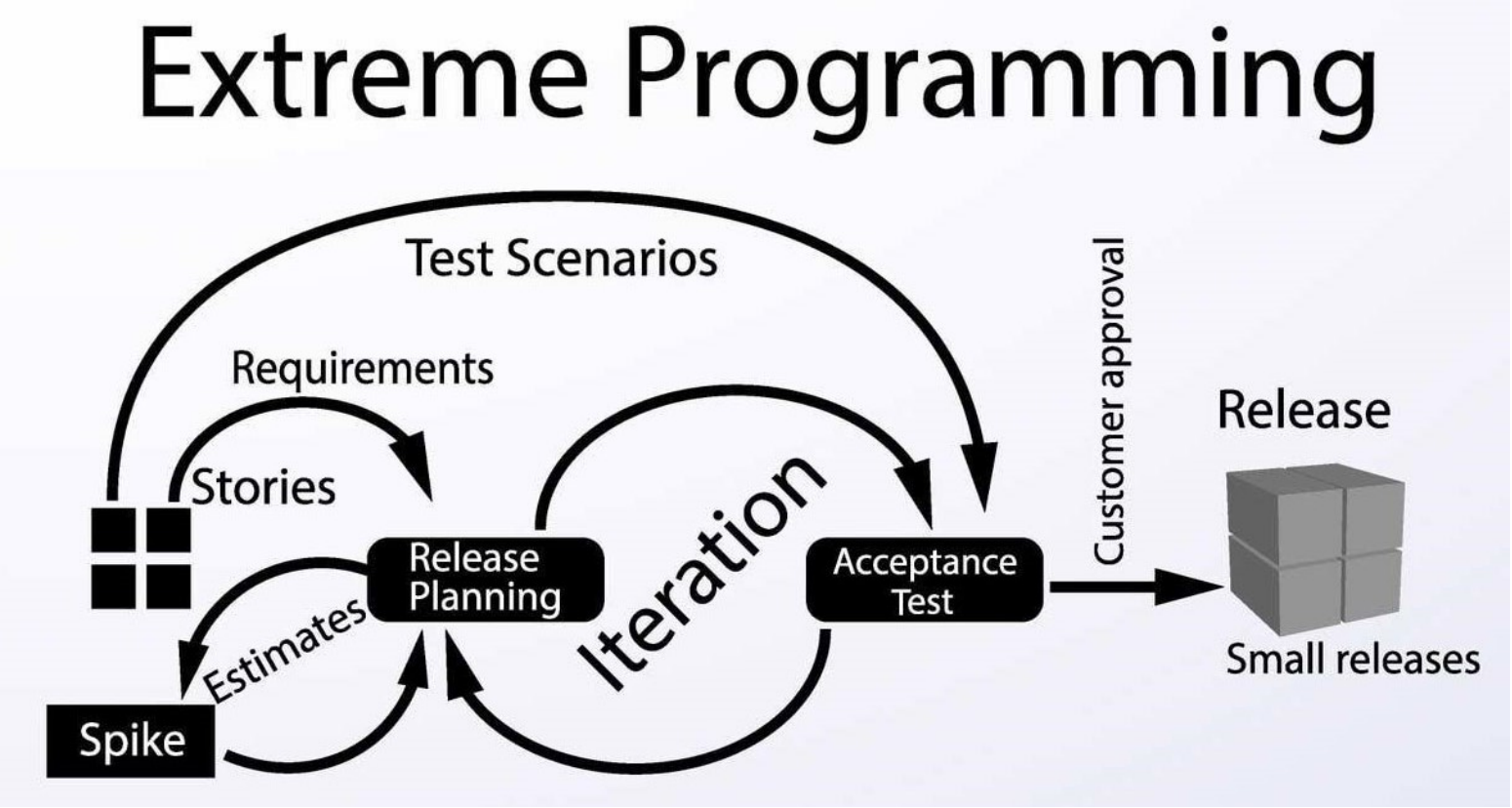
The principles of agile methods

Principle	Description
Customer involvement	Customers should be closely involved throughout the development process. Their role is to provide and prioritize new system requirements to evaluate the iteration of the system

Agile method applicability

- Plane-based **approach is suitable to developing** a product with fully known requirements at the beginning of the development process.
- Agile method is suitable to developing **a small or medium-sized product** for sale
  - Virtually **all apps** are now developed using an agile approach
- Agile method is applied to **Customer system development** within an organization, where there's 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

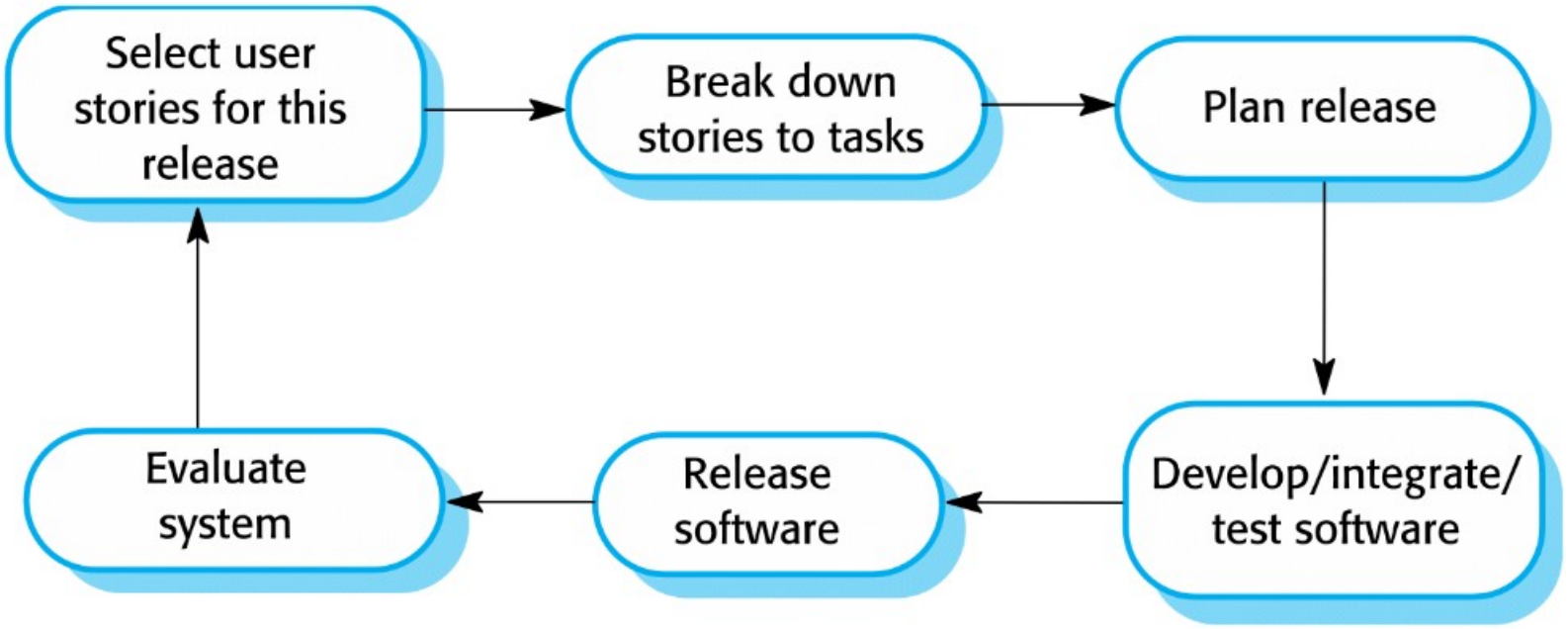
Agile development Methodologies: XP



XP (Extreme programming)

- Extreme programming (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 XP release cycle



XP practices

Principle or 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”

XP practices

Principle or practice	Description
Pair programming	Developers work in pairs, checking each other's work and providing the support to always do a good job

XP & agile principles

- Incremental development is supported through small, frequent system releases
- Customer involvement means full-time customer engagement with the team
- **People** not process are supported through pair programming, collective ownership and a process that avoids long working hours
- **Change's** supported through regular system release
- Maintaining **simplicity** through constant refractoring 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 wildly used
- So **the Key practices of XP** method include
  - User stories for specification
  - Refactoring
  - Test-first development
  - Pair programming

User stories for requirements

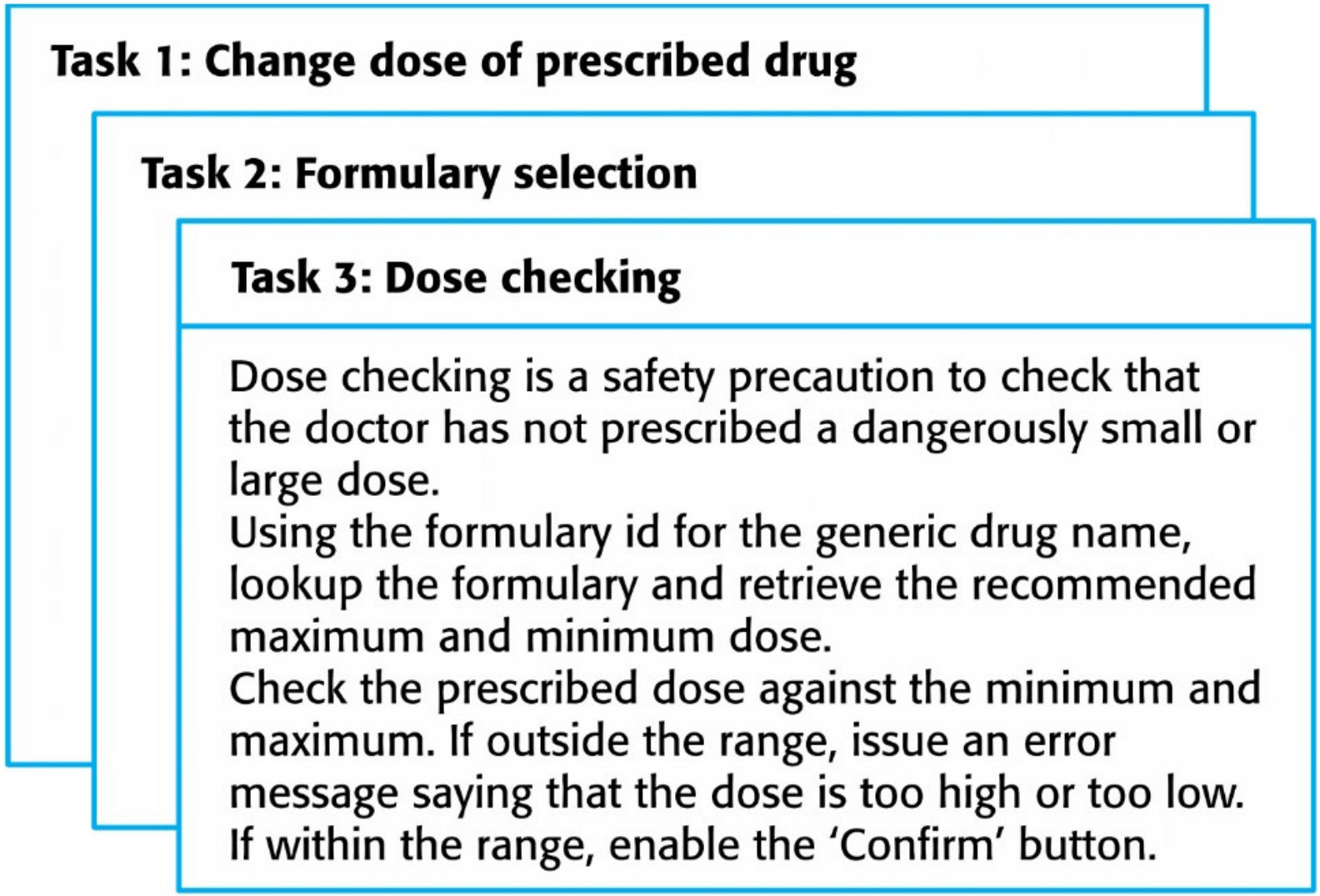
- User requirements are expressed as **user stories** or **scenarios**
- They are written on cards and the development team break them down into **implementation tasks.**
  - These tasks are the basis of schedule and cost estimate
- The customer chooses the stories for inclusion in the next release based on their priorities and the schedule estimates

A 'prescribing medication' story

- 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'll 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 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
- 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

Examples of task cards for prescribing medication

The user story is broken down into the tasks written on the cards



Refactoring

- **As you know, Conventional wisdom** in SE is to design for change. It's worth spending time and effort anticipating **changes** as **this reduces costs later** in the life cycle
- **However, XP** asserts that this is not useful (worthwhile) as changes cannot be reliably anticipated
- Rather, it proposes **constant code improvement** (refactoring) to make changes easier when they have to be implemented

Refactoring...

- To that end, **Programming team** tries to look for possible software improvement and make these improvements **even** there's 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
- However, **some changes** require architecture refactoring and this is much more expensive

Example of refactoring

- Re-organize a class hierarchy to remove duplicate code
- Tidy up and rename 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-driven development

- **Testing** is central to XP and XP has developed an approach where the **program** is tested after every change has been made
- Features of XP testing are
  - Test-first development
  - Incremental test development for scenarios
  - User involvement in test development and validation
  - Automated test harnesses are used to run all component tests each time that a new release is built

Test-driven development

- 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*
  - Usually relies on a testing framework such as Junit
- **All p**revious and new tests are run automatically when new functionality is added, **thus** checking that the new functionality has not introduced errors

Customer involvement

- The role of the customer in the testing process is to help develop **acceptance tests** for **the stories** that are to be implemented in the next release of the system
- **The customer** who is part of the team writes tests as the development proceeds
  - All new code is therefore validated to ensure that it is what the customer needs
- However, people adopting the customer role have limited time available and so cannot work full-time with the development team
  - They may feel that providing the requirements was enough of a contribution and 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

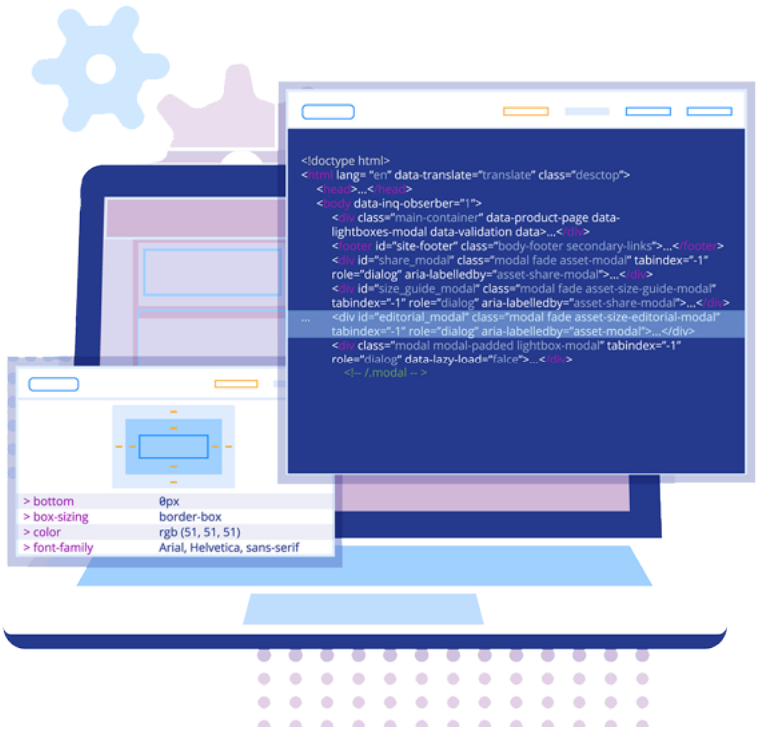
- **Test automation** means that **tests** are written as executable components before the task is implemented
  - These testing components should be stand-alone, **should simulate** the submission of input to be tested and should check that the result meets the output specification
  - An automated test framework (e.g., Junit) is a system that makes it easy to write executable tests and submit a set of tests for executions
- As testing is automated, there is always a set of tests that can be quickly and easily executed
  - Whenever any functionality is added to the system, **the tests** can be run and problems that the new code has introduced can be caught immediately

Problems with test-first development

- **Programmers** prefer programming to testing and sometimes they take shortcuts when writing test
  - E.g., they may write incomplete tests that don't check for all possible exceptions that may occur
- **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 code that implements 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

Pair programming

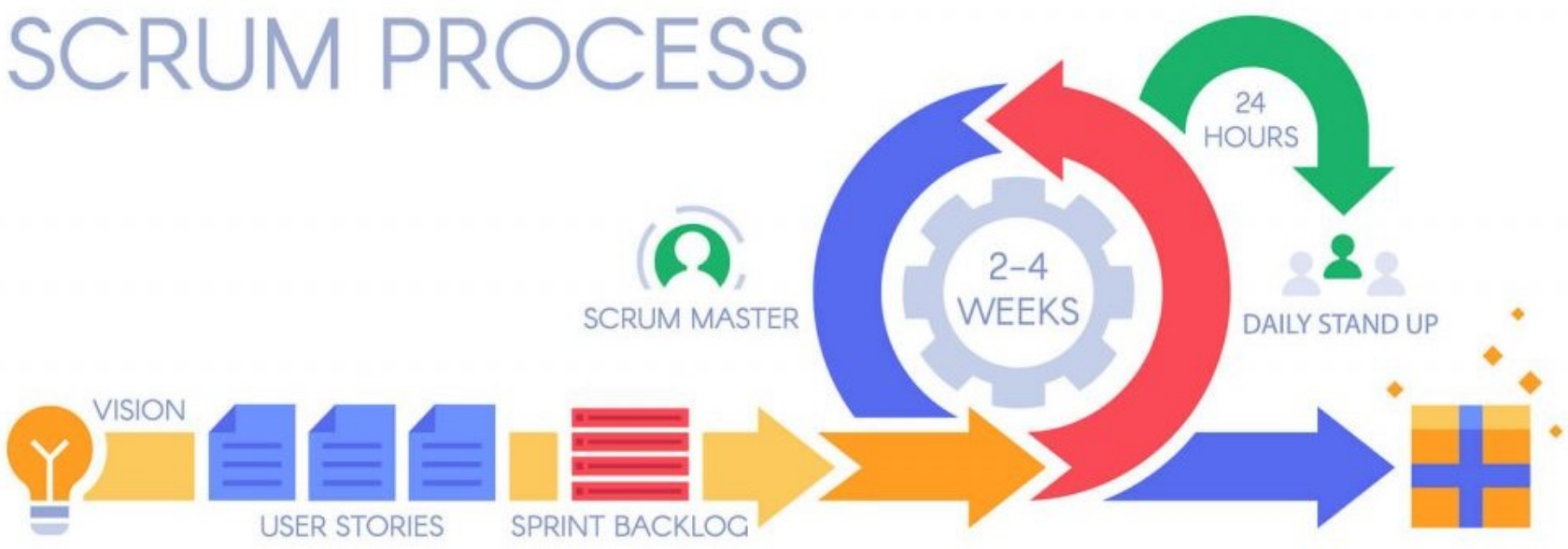
- Pair programming involves programmers working in pairs, developing code together
- This helps develop common ownership of code and spreads knowledge across the team
- It serves as an informal review process as each line of code is looked at by more than 1 person
- It encourages refactoring as the whole team can benefit from improving the system code



Pair programming...

- In pair programming, **programmers** sit together at the same computer to develop the software
- **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**
- Pair programming is pretty efficient 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 the planned budget for the project
- The standard approach to project management is **plan-driven**
  - **Managers** draw up a plan for the project showing what should be delivered, when it should be delivered and who will work on the development of the project deliverables
- Agile project management requires a different approach, which is adapted to incremental development and the practices used in agile methods

Scrum

- Scrum is an agile method that focuses on **managing iterative development** rather than specific agile practices
- There are 3 phases in Scrum
  - The initial phase is an outline planning phase where you establish the general objectives for the project and design the software architecture
  - This is followed by a series of sprint cycles, where each cycle develops an increment of the system
  - The project closure phase wraps up the project, completes required documentation such as system help frames and user manuals and assesses the lessons learned from the project

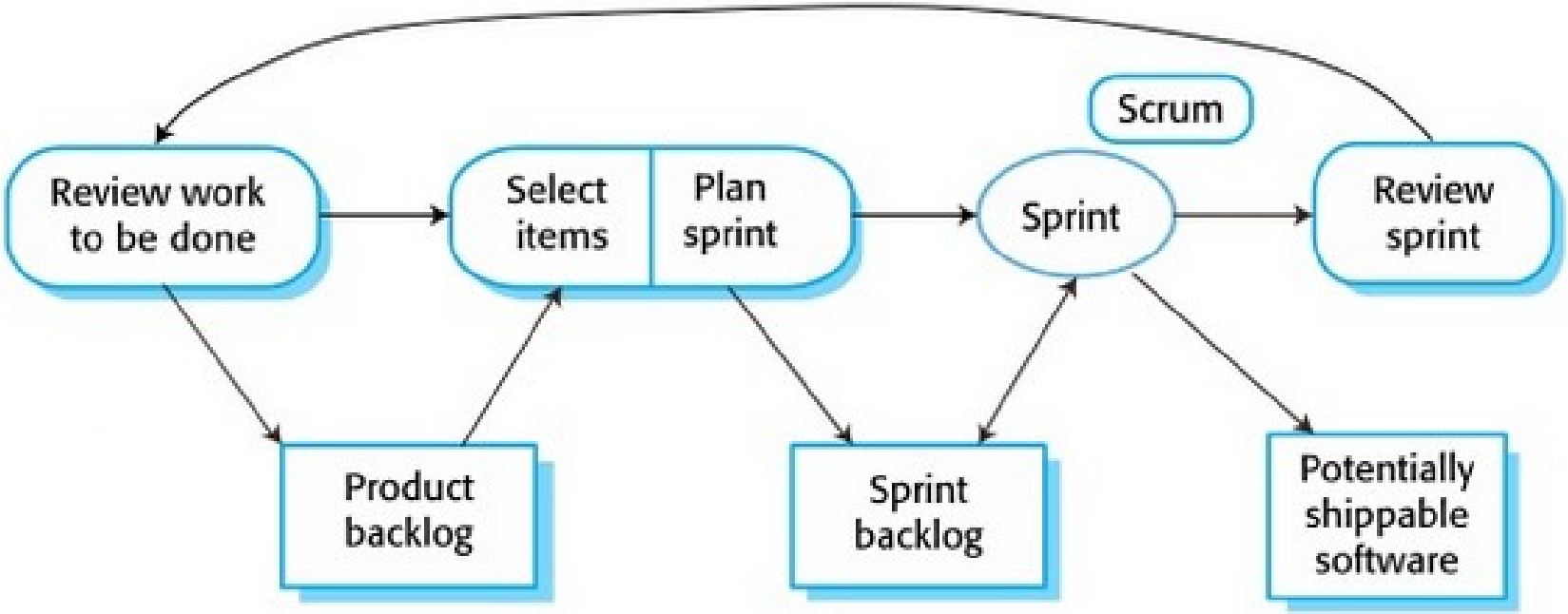
Scrum terminology

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. The idea is that this should be 'potentially shippable' which means that it is a finished state and no further work, such as testing, is needed to incorporate it into the final product. In practice, this's not always achievable
Product backlog	This is a list of 'to do' items which 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 shareholder representative

Scrum terminology...

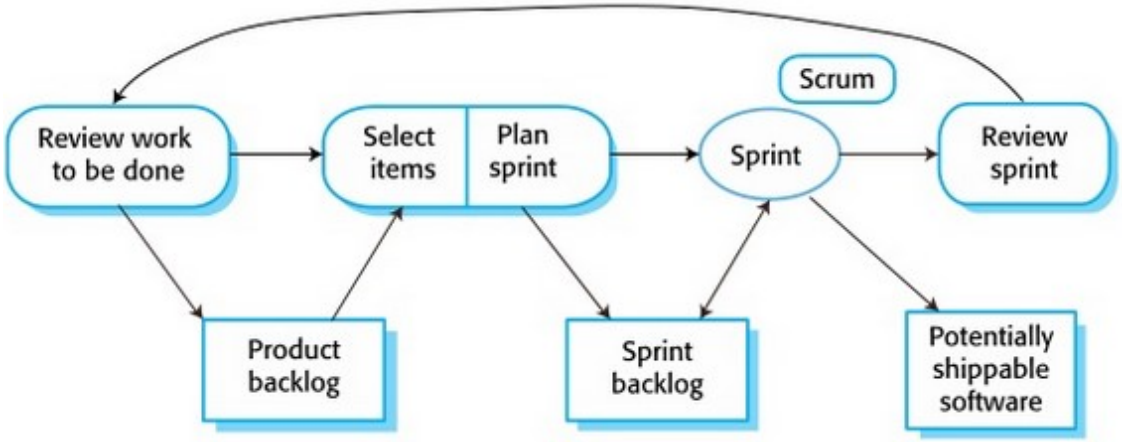
Scrum term	Definition
Scrum	A daily meeting of the Scrum team that reviews progress and prioritizes work to be done on that day. Ideally, this should be a short face-to-face meeting that includes the whole team
ScrumMaster	is responsible for ensuring that the Scrum process is followed and guides the team in the effective use of Scrum. S/he 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.
Sprint	A development iteration. Sprints are usually 2-4 weeks long
Velocity	An estimate of how much product backlog effort that a team can cover in a single unit. Understanding a team's velocity helps them estimate what can be covered in a sprint and provides a basis for measuring improvement performance

Scrum sprint cycle



Scrum sprint cycle

- Sprints are fixed length, normally 2-4 weeks
- The starting point for planning is **the product backlog**, which is the list of works 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 from the product backlog to be developed during **the sprint**



The sprint cycle

- Once selected items 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 channeled through the so-called **'Scum master'**
  - 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. The next sprint cycle then begins

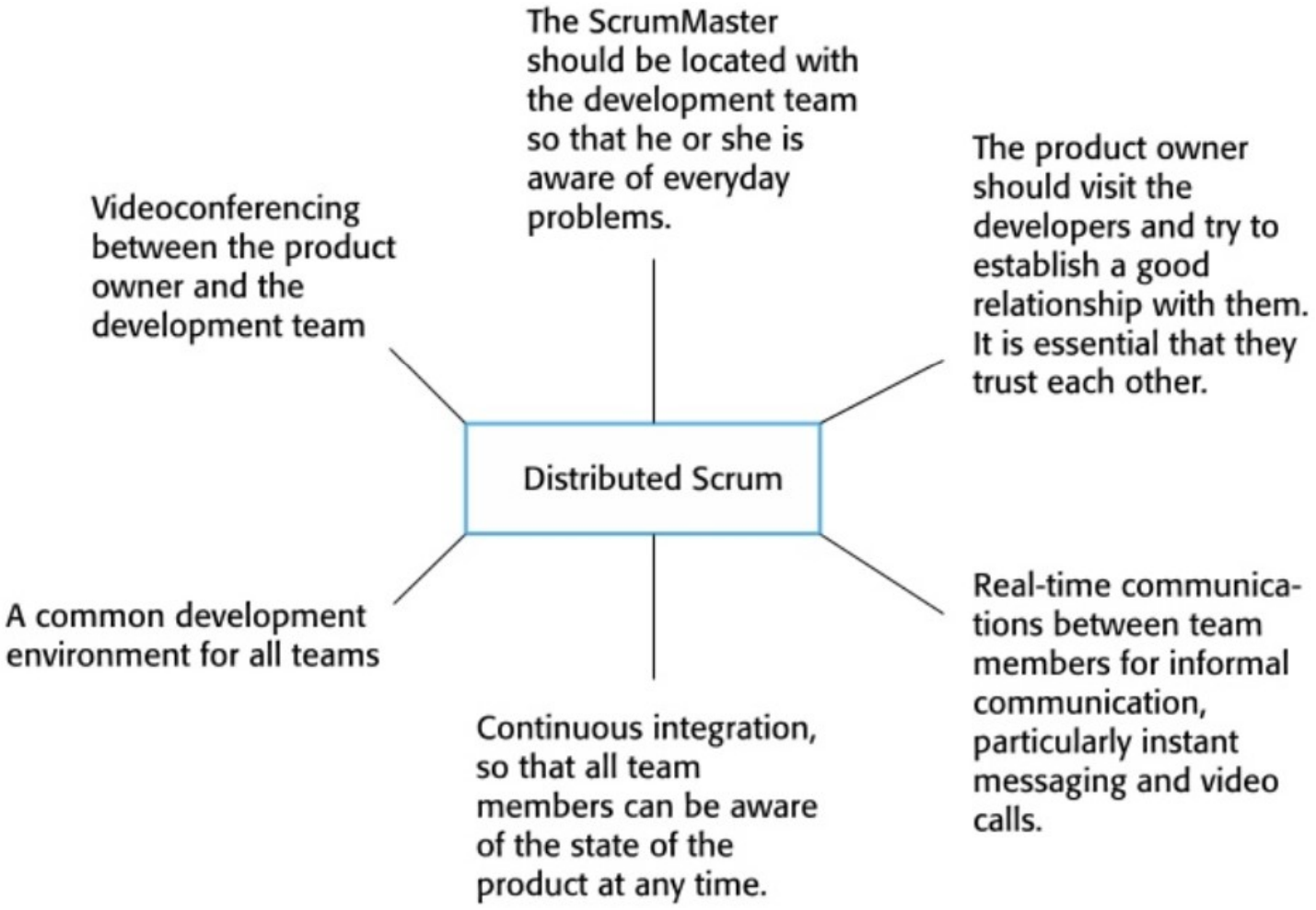
Teamwork in Scrum

- The 'Scrum master' is a facilitator who arranges daily meetings,
  - track the backlog of work to be done, record decisions, measure progress against the backlog
  - and communicates with customers and management outside of the team
- **The whole team** attends short daily meetings (Scrums)
  - where all team members share information,
  - describe their progress since the last meeting, problems that have arisen and what is planned 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

- **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 in which everyone expects the project succeeds

Distributed Scrum



Scrum vs. XP

- Scrum is a framework for helping teams develop complex projects in an adaptive manner
  - Scrum doesn't dictate how developers do the work
  - Scrum can be applied to any project that benefits from an iterative approach
- XP puts much emphasis on **good programming practices**
- It's highly recommended to use XP and Scrum together!

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

Scaling out and scaling up

- ‘Scaling up’ is concerned with using agile methods for developing large software systems that cannot be developed by small team
- ‘Scaling out’ is concerned with how agile methods can be 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 and good team communications

Practical problems with agile methods

- 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 are based around a specification, which sets out what has to be implemented by the system developer for the system customer
- However, this precludes interleaving specification and development as is the norm in agile development
- A contract that pays for developer time rather than functionality is required
  - However, this is seen as a high risk by many legal departments because what has to be delivered cannot be guaranteed

Agile methods & software maintenance

- Most organizations spend more on maintaining existing software **than** they do on new software development
  - So if agile methods are to be successful, they have to support maintenance as well as original development
- 2 key issues
  - Are systems that are developed using an agile approach maintainable, given the emphasis in the development process of minimizing formal documentation?
  - Can agile methods be used effectively for evolving a system in response to customer change requests?
- Problems may arise **if** original development team cannot be maintained

Agile maintenance

- Key problems are
  - Lack of product documentation
  - Keeping customers involved in the development process
  - Maintaining the continuity of the development team
- Agile development **relies on what** 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 & plan-driven methods

- **Most projects** include **elements** of plan-driven and agile processes
- Deciding on the balance depends on
  - It is important to have a very detailed specification and design before moving to implementation? If so, you probably need to use a plan-driven approach
  - Is an incremental delivery strategy, where you deliver the software to customers and get rapid feedback from them, realistic? If so, consider using agile methods
  - How large is the system that is being developed? Agile methods are most effective when the system can be developed with a small co-located team who can communicate informally. This may not be possible for large systems that require larger development teams so a plan-driven approach may have to be used

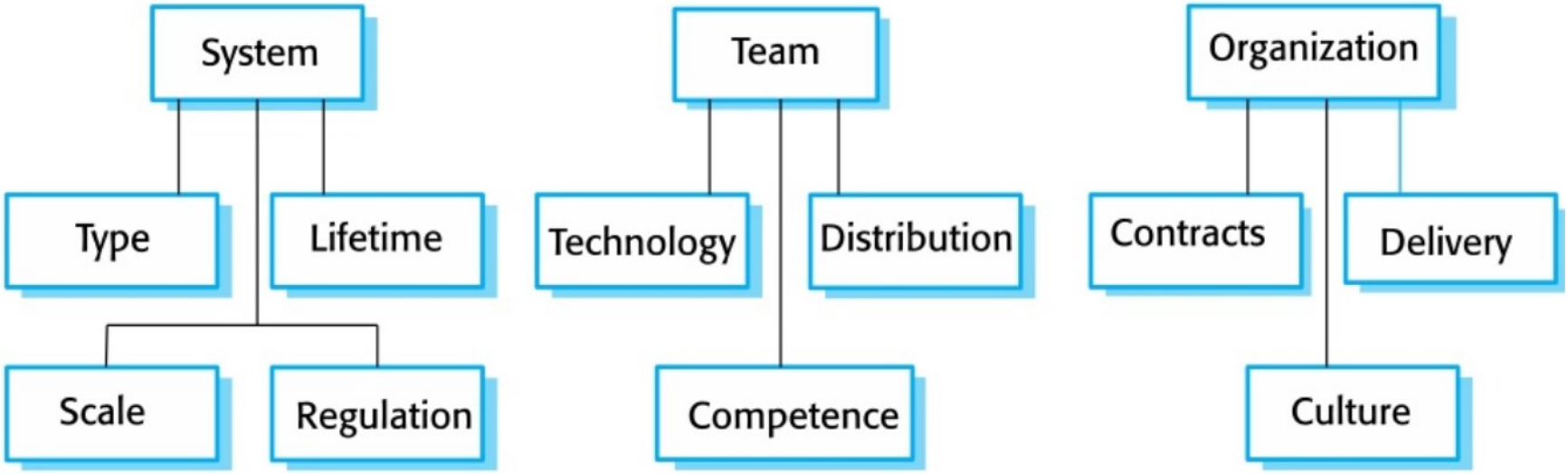
Agile principles & organizational practice

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're 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 what product features several months in advance to operate an effective marketing campaign

Agile principles & organizational practice

Principle	Practice
Maintain simplicity	Under pressure from deliver 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 intensive involvement that is typical of agile methods, and therefore may not interact well with other team members

Agile & plan-based factors



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 time
- 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 teams

- 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 programmer simply translate a detailed design into code
- How is the development team organized?
  - Design documents may be required **if** the team is distributed
- What support technologies are available?
  - IDE support for visualization 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 of system increments?
- Can informal agile development fit into the organizational culture of detailed documentation?

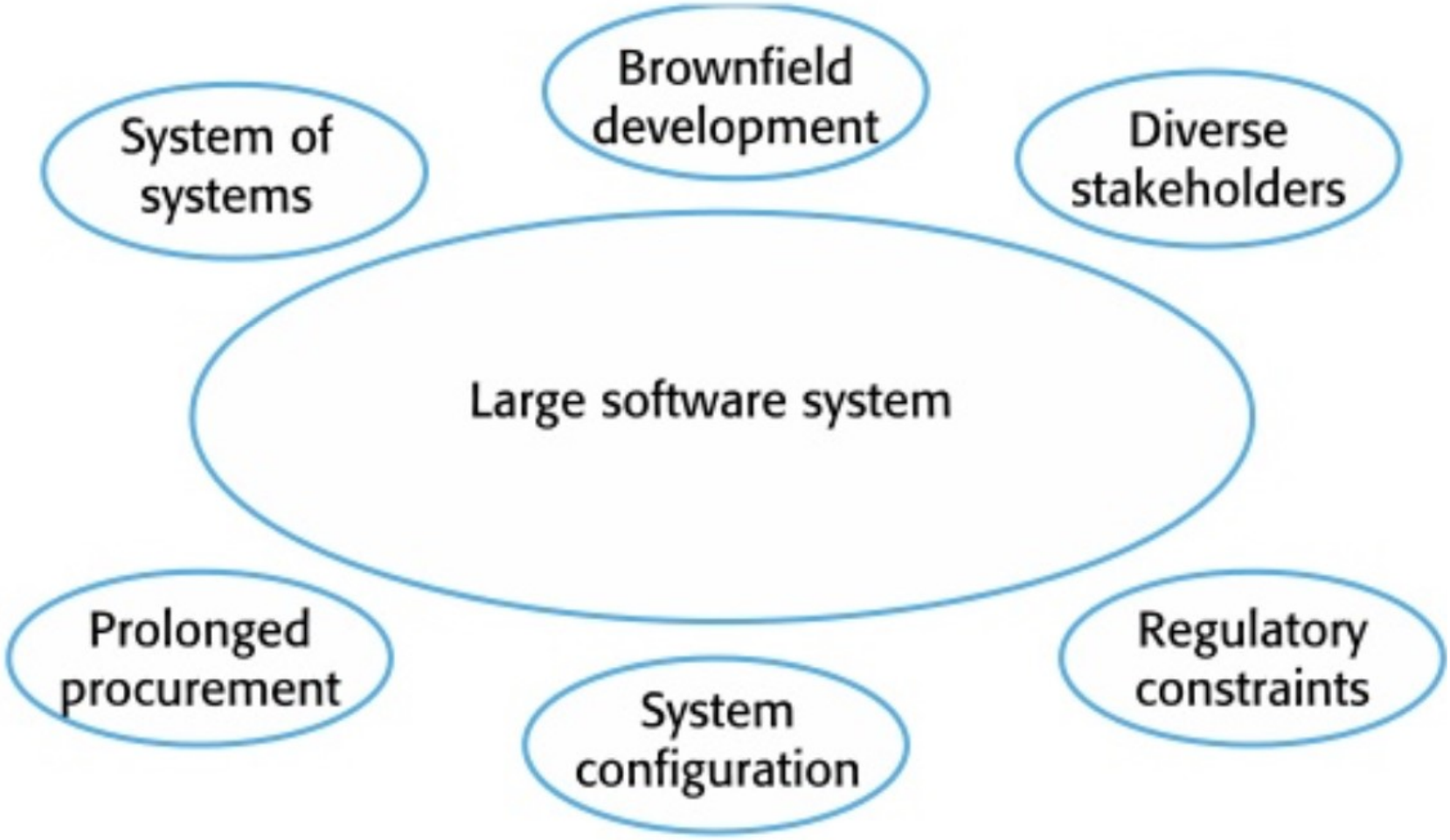
Agile methods for large systems

- Large systems are usually collections of separate, communicating systems, where **separate teams** develop each system
  - Frequently these teams are working in different places, sometimes in different time zones
- Large systems are ‘brownfield systems’, that is they include and interact with a number of existing systems
  - Many of the system requirements are concerned with this interaction and so don’t really lend themselves to flexibility and incremental development
- 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

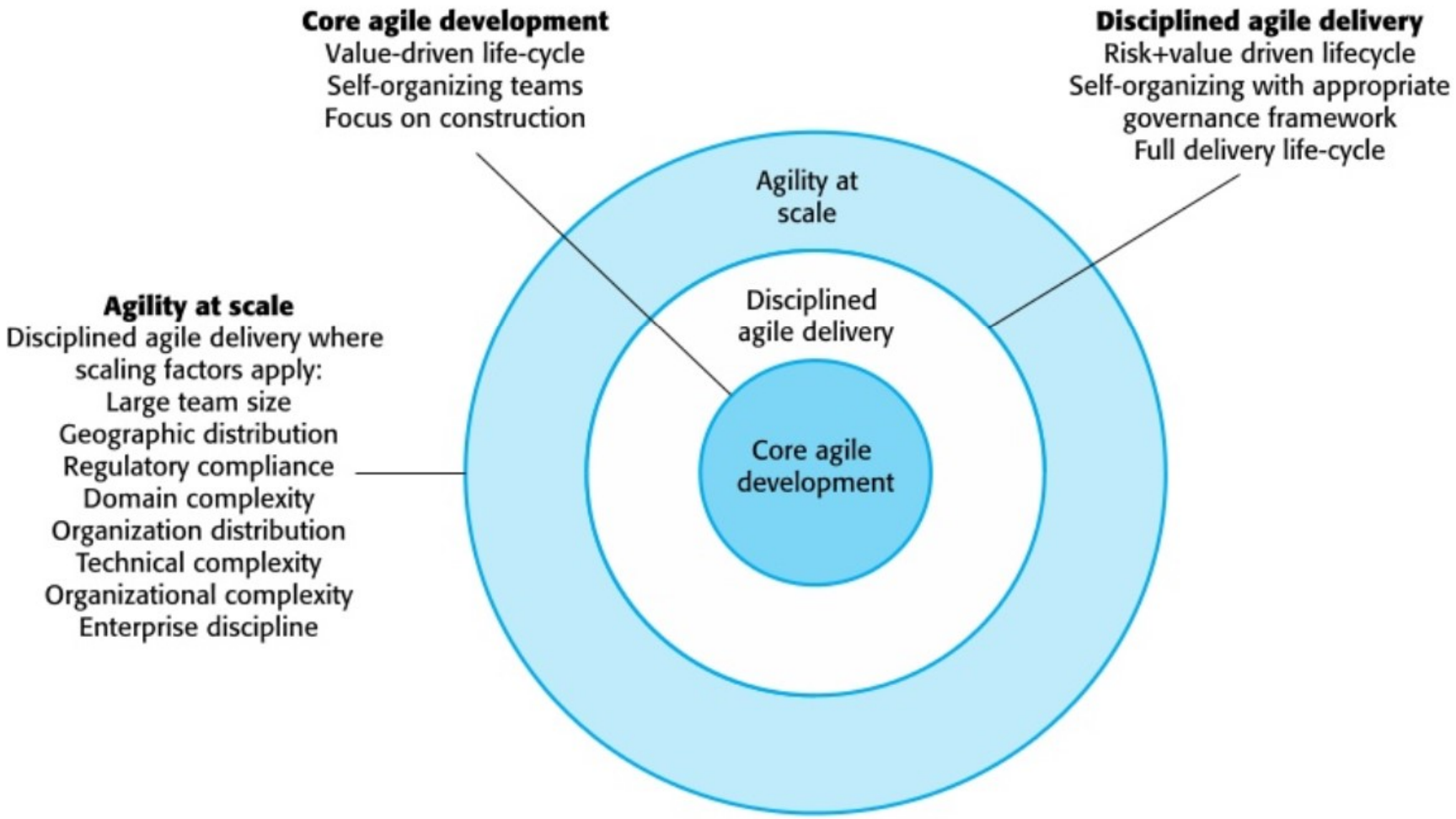
Large system development

- Large systems and their development processes are often constrained by external rules and regulations limiting the way 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 as, inevitably, people move on to other jobs and projects
- Large systems usually have a diverse set of stakeholders
  - It is practically impossible to involved all of these different stakeholders in the development process

Factors in large systems



IBM’s agility at scale model



Scaling up to large systems

- 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 of the system
- Cross-team communication mechanisms have to be designed and used
- Continuous integration is practically impossible
  - However, it's essential to maintain frequent system builds and regular releases of the system

Multi-team Scrum

- Role replication
  - Each team has a Product Owner for their work component and ScrumMaster
- Product architects
  - Each team chooses a product architecture 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 demonstrated 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

- Project managers who don't 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 allow 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 points

- Agile methods are incremental development methods that focus on rapid software development, frequent releases of the software, 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

- Scrum in an agile method that provides a project management framework
  - It is centered around 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