Agile Methods

XP & Scrum

Traditional and Agile Methods for:

1. Software Development Process
2. Software Project Management

Plan-driven and Agile processes

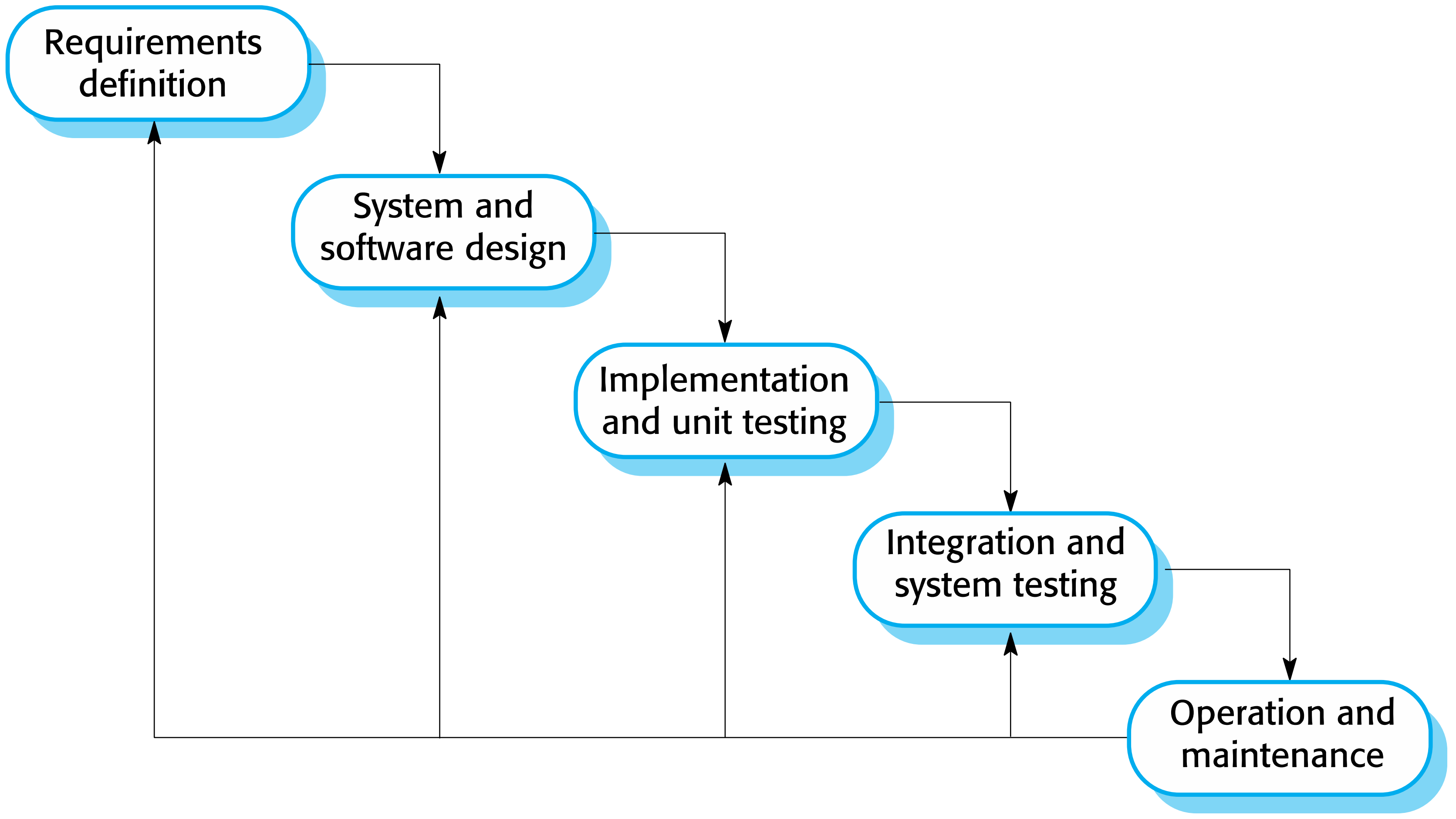
- Plan-driven processes are processes where all of the process activities are planned in advance and progress is measured against this plan.
- In agile processes, planning is incremental, and it is easier to change the process to reflect changing customer requirements.
- In practice, most practical processes include elements of both plan-driven and agile approaches.
- There are no right or wrong software processes.

Software process models

Software process models

- **The waterfall model**
  - Plan-driven model. Separate and distinct phases of specification and development.
- **Incremental development**
  - Specification, development and validation are interleaved. May be plan-driven or agile.
- **Integration and configuration**
  - The system is assembled from existing configurable components. May be plan-driven or agile.
- **In practice, most large systems are developed using a process that incorporates elements from all of these models.**

The waterfall model


Requirements definition
System and software design
Implementation and unit testing
Integration and system testing
Operation and maintenance

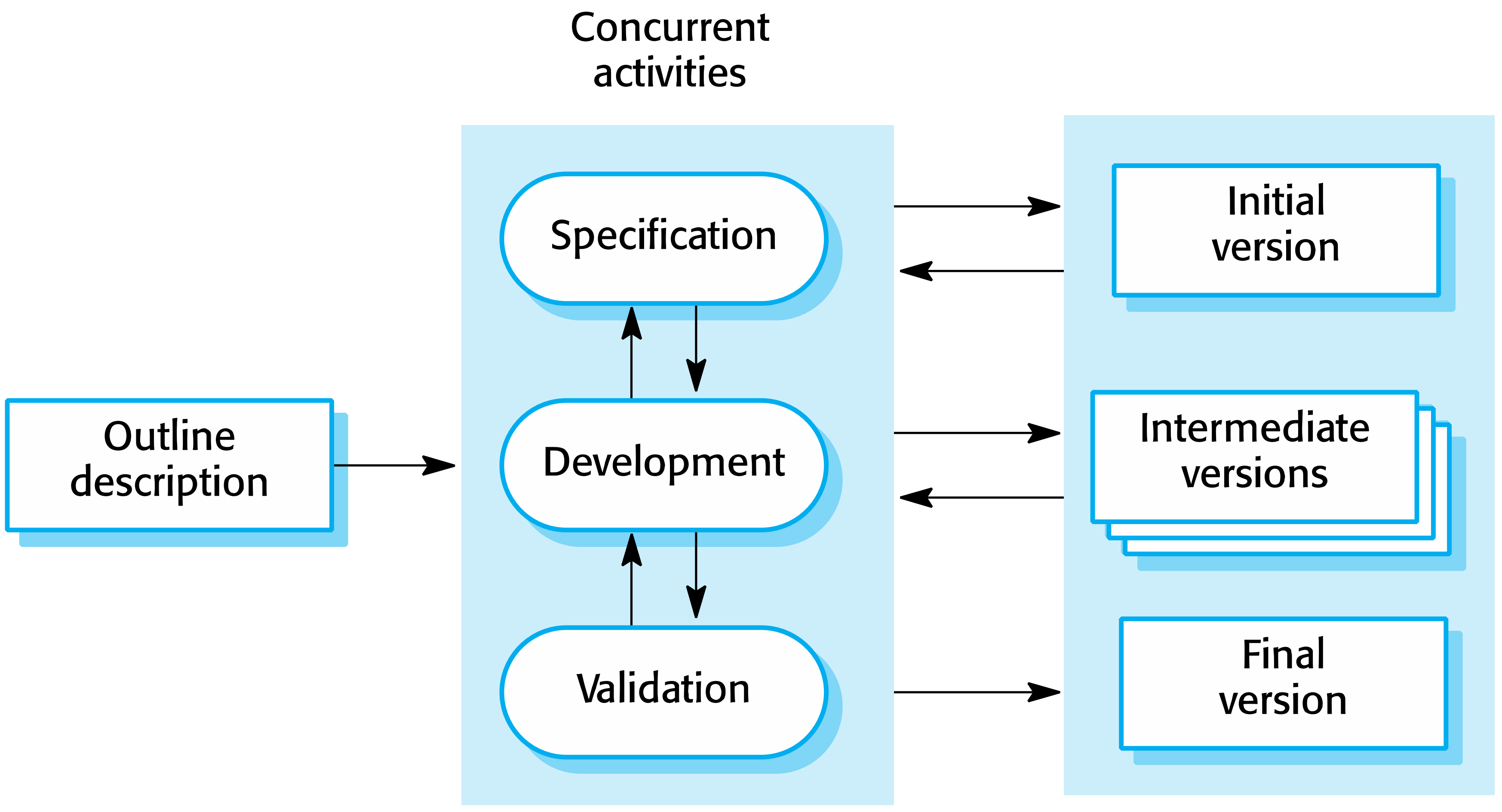
Waterfall model phases

- **There are separate identified phases in the waterfall model:**
  - Requirements analysis and definition
  - System and software design
  - Implementation and unit testing
  - Integration and system testing
  - Operation and maintenance
- **The main drawback of the waterfall model is the difficulty of accommodating change after the process is underway. In principle, a phase has to be complete before moving onto the next phase.**

Waterfall model problems

- **Inflexible partitioning of the project into distinct stages makes it difficult to respond to changing customer requirements.**
    - Therefore, this model is only appropriate when the requirements are well-understood and changes will be fairly limited during the design process.
    - Few business systems have stable requirements.
- **The waterfall model is mostly used for large systems engineering projects where a system is developed at several sites.**
    - In those circumstances, the plan-driven nature of the waterfall model helps coordinate the work.

Incremental development



Incremental development benefits

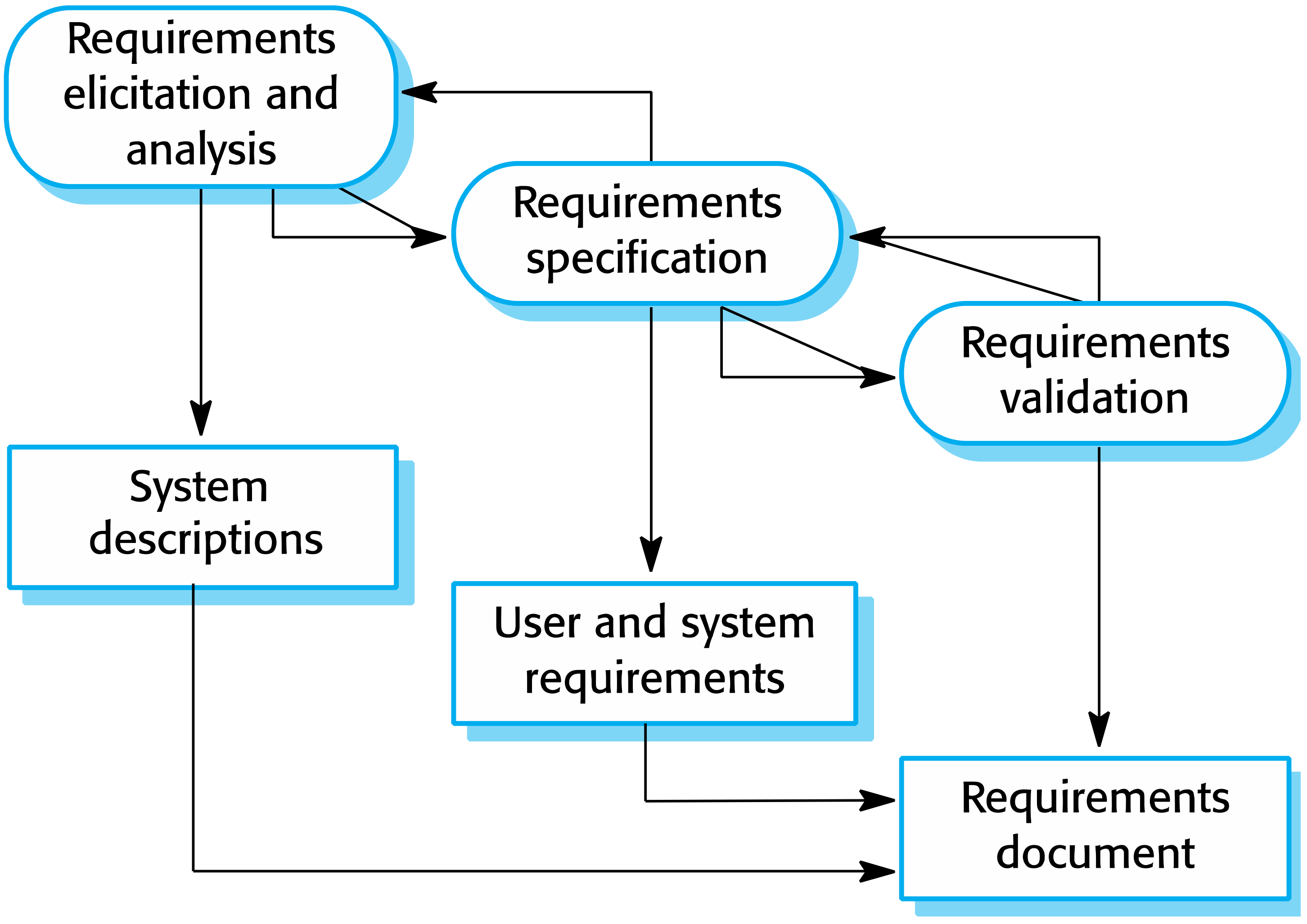
- **The cost of accommodating changing customer requirements is reduced.**
  - The amount of analysis and documentation that has to be redone is much less than is required with the waterfall model.
- **It is easier to get customer feedback on the development work that has been done.**
  - Customers can comment on demonstrations of the software and see how much has been implemented.
- **More rapid delivery and deployment of useful software to the customer is possible.**
  - Customers are able to use and gain value from the software earlier than is possible with a waterfall process.

Process activities

Process activities

- Real software processes are inter-leaved sequences of technical, collaborative and managerial activities with the overall goal of specifying, designing, implementing and testing a software system.
- The four basic process activities of specification, development, validation and evolution are organized differently in different development processes.
- For example, in the waterfall model, they are organized in sequence, whereas in incremental development they are interleaved.

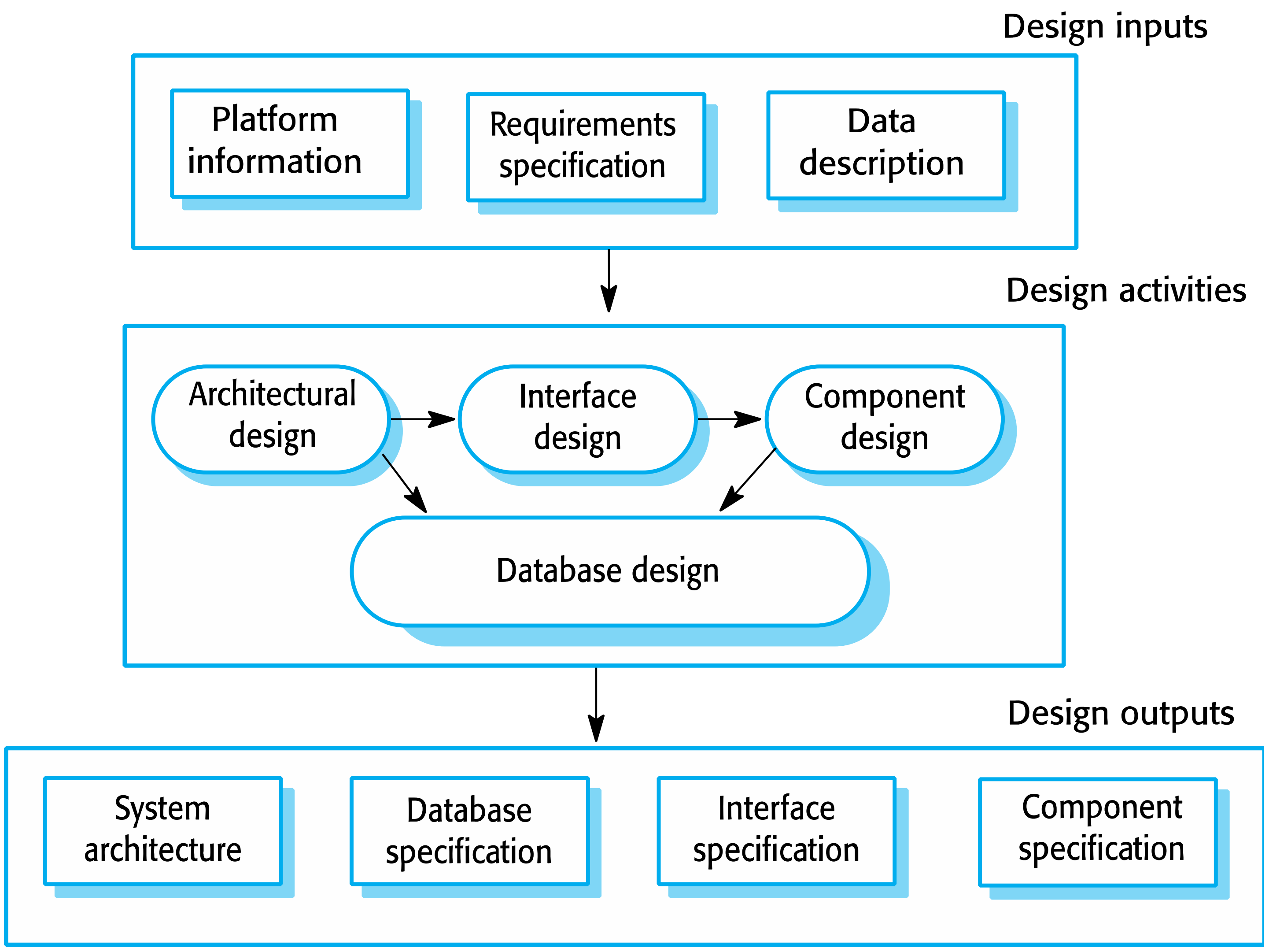
The requirements engineering process


Requirements elicitation and analysis
Requirements specification
Requirements validation
System descriptions
User and system requirements
Requirements document

Software design and implementation

- The process of converting the system specification into an executable system.
- Software design
  - Design a software structure that realises the specification;
- Implementation
  - Translate this structure into an executable program;
- The activities of design and implementation are closely related and may be inter-leaved.

A general model of the design process



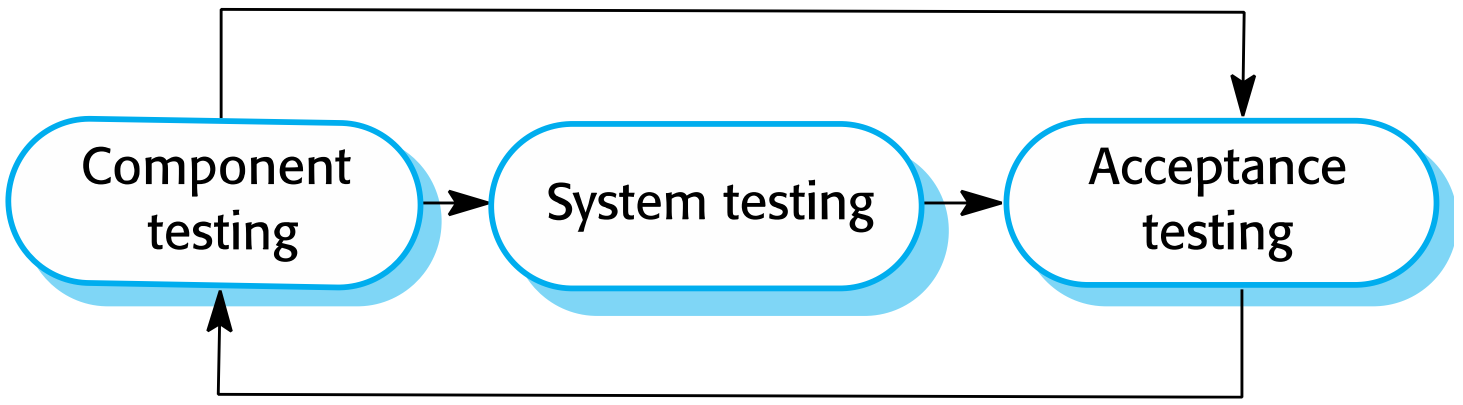
System implementation

- The software is implemented either by developing a program or programs or by configuring an application system.
- Design and implementation are interleaved activities for most types of software system.
- Programming is an individual activity with no standard process.
- Debugging is the activity of finding program faults and correcting these faults.

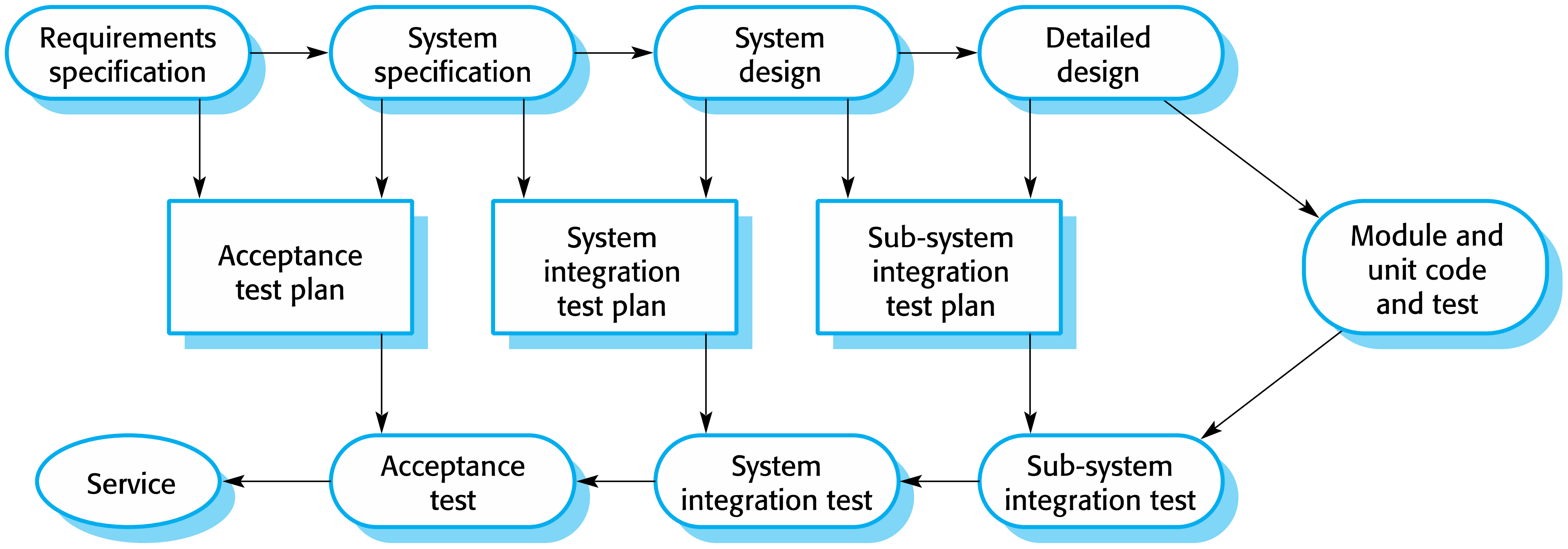
Software validation

- Verification and validation (V & V) is intended to show that a system conforms to its specification and meets the requirements of the system customer.
- Involves checking and review processes and system testing.
- System testing involves executing the system with test cases that are derived from the specification of the real data to be processed by the system.
- Testing is the most commonly used V & V activity.

Stages of testing



Testing phases in a plan-driven software process (V-model)

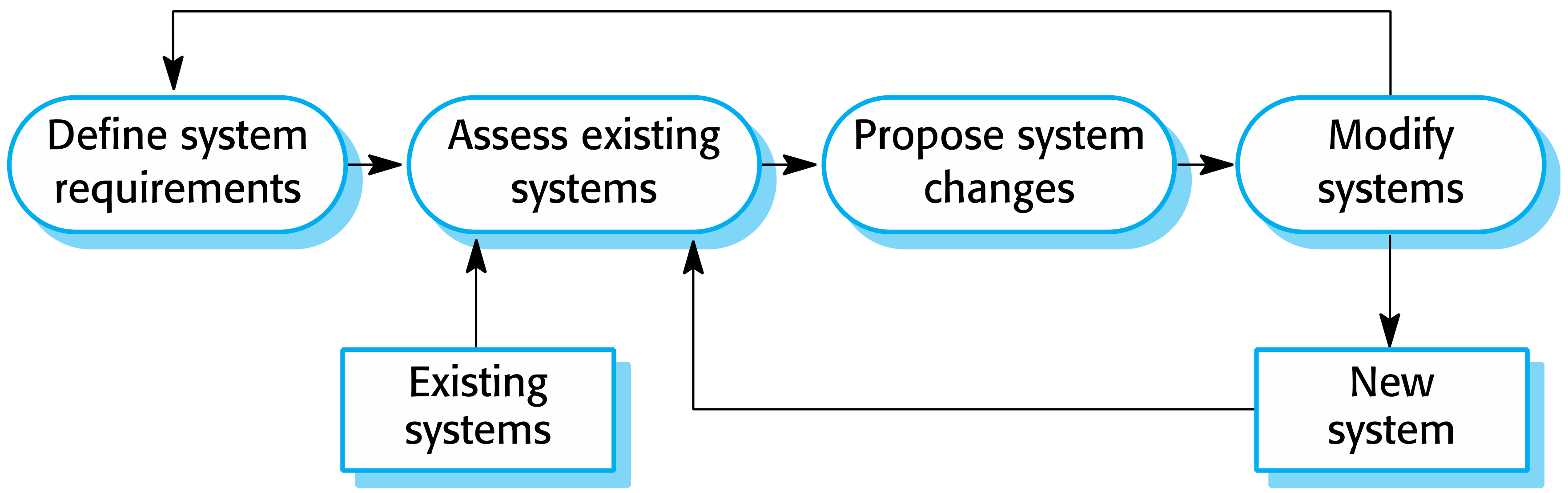


Requirements specification
System specification
System design
Detailed design
Acceptance test plan
System integration test plan
Sub-system integration test plan
Module and unit code and test
Service
Acceptance test
System integration test
Sub-system integration test

Software evolution

- Software is inherently flexible and can change.
- As requirements change through changing business circumstances, the software that supports the business must also evolve and change.
- Although there has been a demarcation between development and evolution (maintenance) this is increasingly irrelevant as fewer and fewer systems are completely new.

System evolution



Coping with change

Coping with change

- **Change is inevitable in all large software projects.**
  - Business changes lead to new and changed system requirements
  - New technologies open up new possibilities for improving implementations
  - Changing platforms require application changes
- **Change leads to rework so the costs of change include both rework (e.g. re-analysing requirements) as well as the costs of implementing new functionality**

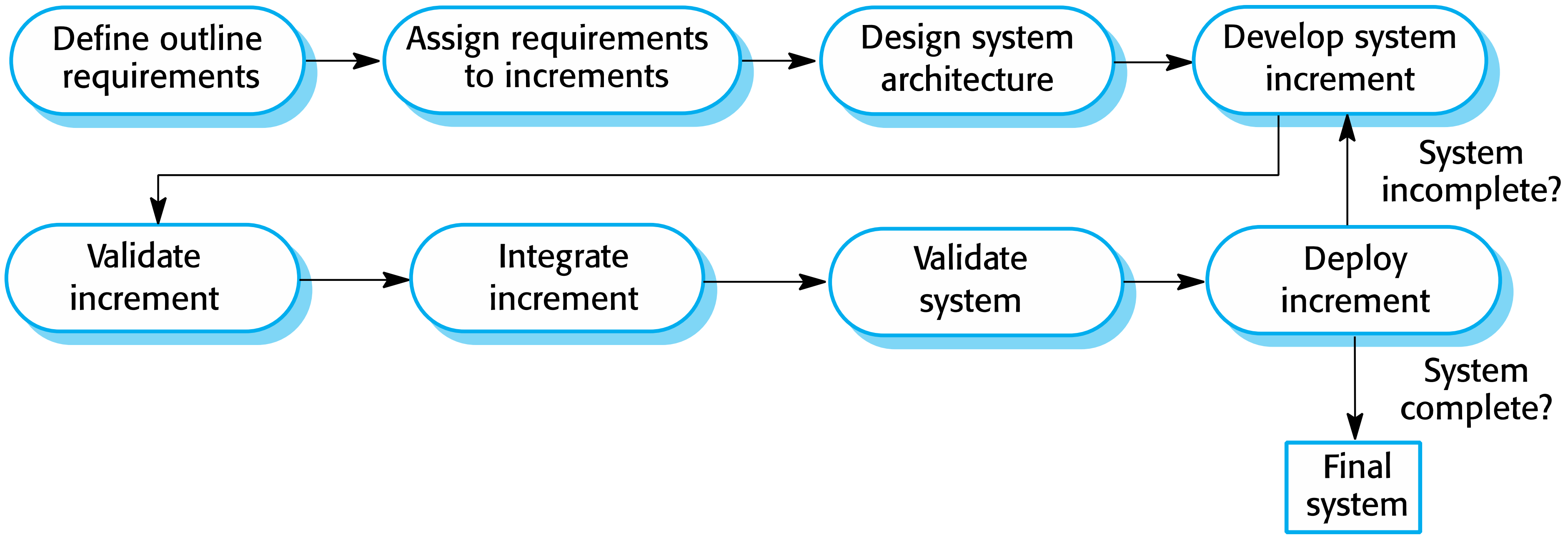
Incremental delivery

- Rather than deliver the system as a single delivery, the development and delivery is broken down into increments with each increment delivering part of the required functionality.
- User requirements are prioritised and the highest priority requirements are included in early increments.
- Once the development of an increment is started, the requirements are frozen though requirements for later increments can continue to evolve.

Incremental development and delivery

- **Incremental development**
  - Develop the system in increments and evaluate each increment before proceeding to the development of the next increment;
  - Normal approach used in agile methods;
  - Evaluation done by user/customer proxy.
- **Incremental delivery**
  - Deploy an increment for use by end-users;
  - More realistic evaluation about practical use of software;
  - Difficult to implement for replacement systems as increments have less functionality than the system being replaced.

Incremental delivery



Software Product Blueprints

Project Plan and Development Process

- In the Software Industry, we have to distinguish between:

1. Software Project Plan
2. Software Development Process

Different Types of Languages

- In the Software Industry, we have three types of languages:

1. Specification Languages (Requirements) English/SDL
2. Modeling Languages(Design) UML
3. Programming Languages(implementation) Java

A Language: whats in the name?

- Every couple of years (more or less) we get a new Programming Language
- Why we don't get a new Natural Language every couple of years?
- Why we don't get a new Modeling Language every couple of years?

A Language: what's in the name?

	Programming Languages	Modeling Language	Natural Language
1950	Fortran Cobol		English
1960	Basic Algol Simula		English
1970	C Pascal Smaltalk		English
1980	C++ Ada Objective-C Perl		English
1990	Java Python VisualBasic PHP Ruby	UML 1.x	English
2000	C#	UML 2.x	English
2010	Swift	UML 2.x	English

Agile Software Development

Agile: what’s in the name?

- Agile development methods
  - XP
- Agile project management
  - Scrum

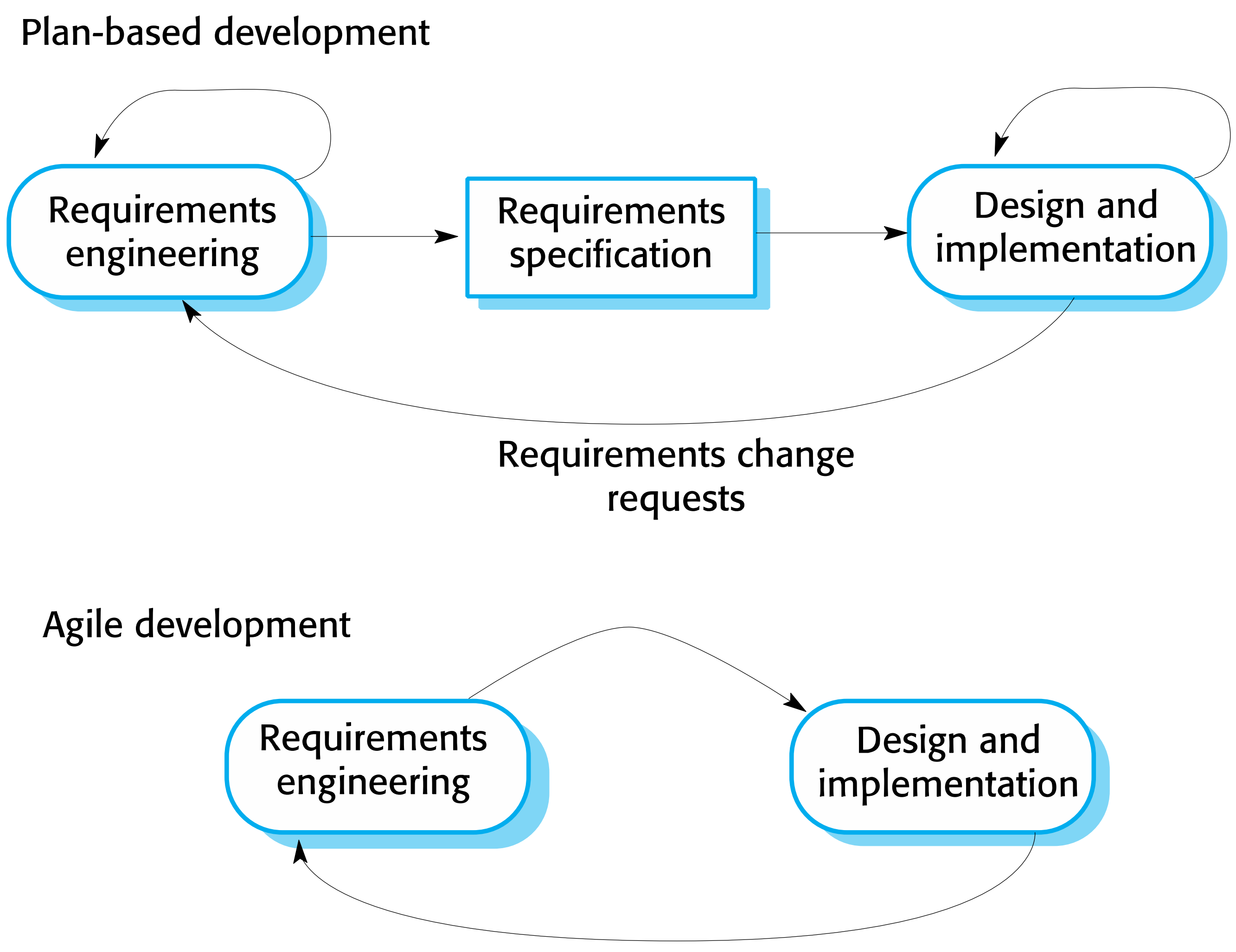
Rapid software development

- **Rapid development and delivery is now often the most important requirement for software systems**
  - Businesses operate in a fast –changing requirement 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.**
- **Agile development methods emerged in the late 1990s whose aim was to radically reduce the delivery time for working software systems**

Agile development

- Program specification, design and implementation are inter-leaved
- The system is developed as a series of versions or increments with stakeholders involved in version specification and evaluation
- 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

Plan-driven and agile development


Plan-based development
Requirements engineering
Requirements specification
Design and implementation
Requirements change requests
Agile development
Requirements engineering
Design and implementation

Plan-driven and agile development

- **Plan-driven development**
  - A plan-driven approach to software engineering 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 inter-leaved and the outputs from the development process are decided through a process of negotiation during the software development process.

Agile methods

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 this quickly to meet changing requirements.

- **The aim of agile methods is to reduce overheads in the software process (e.g. by limiting documentation) and to be able to respond quickly to changing requirements without excessive rework.**

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.***

The principles of agile methods

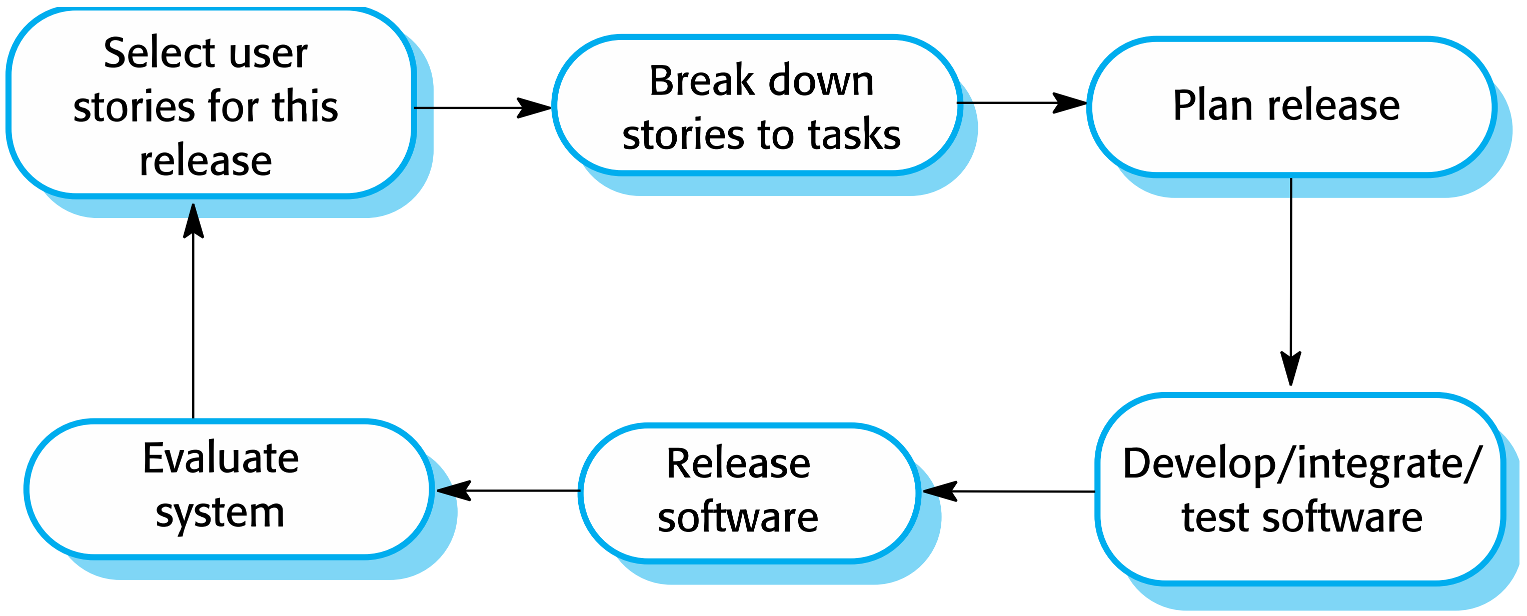
Principle	**Description**
Customer involvement	Customers should be closely involved throughout the development process. Their role is provide and prioritize new system requirements and to evaluate the iterations of the system.
Incremental delivery	The software is developed in increments with the customer 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 development techniques

Extreme programming

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


Select user stories for this release
Break down stories to tasks
Plan release
Develop/integrate/ test software
Release software
Evaluate system

Extreme programming practices (a)

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'. See Figures 3.5 and 3.6.
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)

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 (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.

XP and 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.

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 three 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.

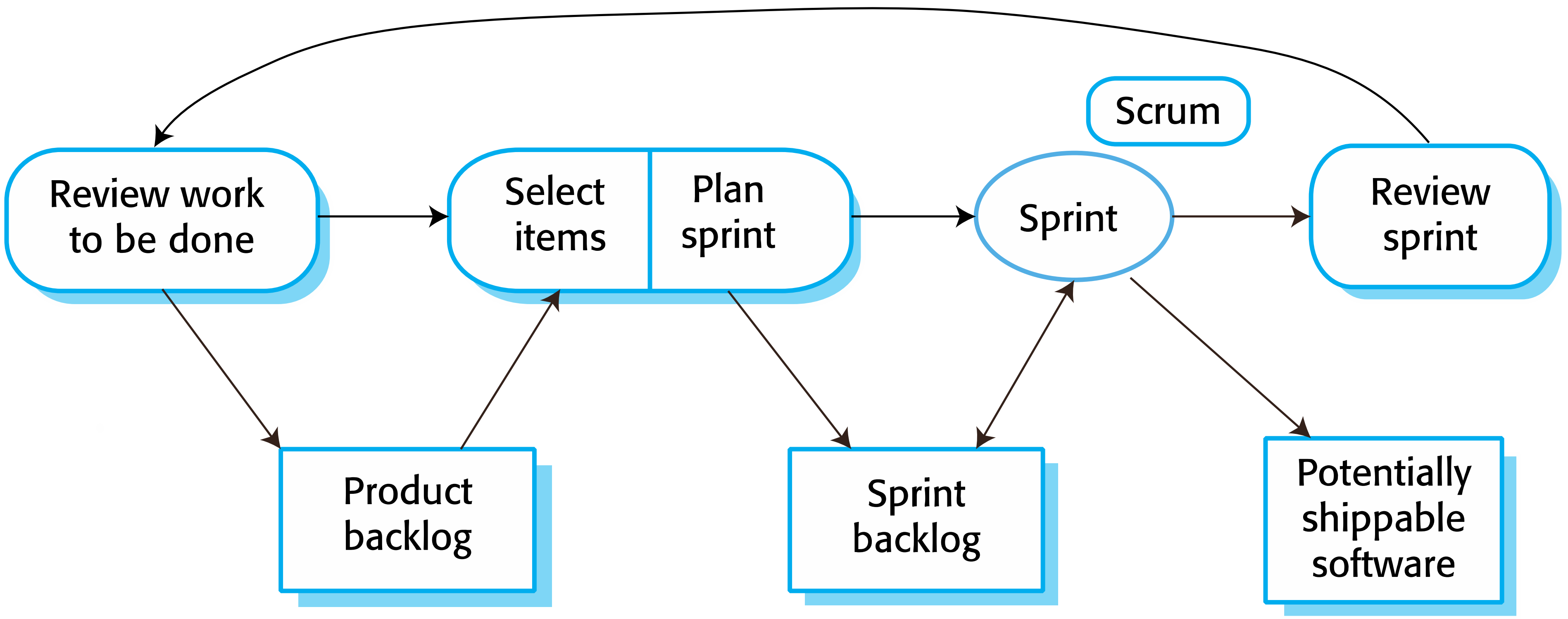
Scrum terminology (a)

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. 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 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 stakeholder representative.

Scrum terminology (b)

Scrum term	Definition
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 2-4 weeks long.
Velocity	An estimate of how much product backlog effort that 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 sprint cycle



The 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 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 from the product backlog to be developed during the sprint.

The Sprint cycle

- Once these 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 so-called ‘Scrum 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

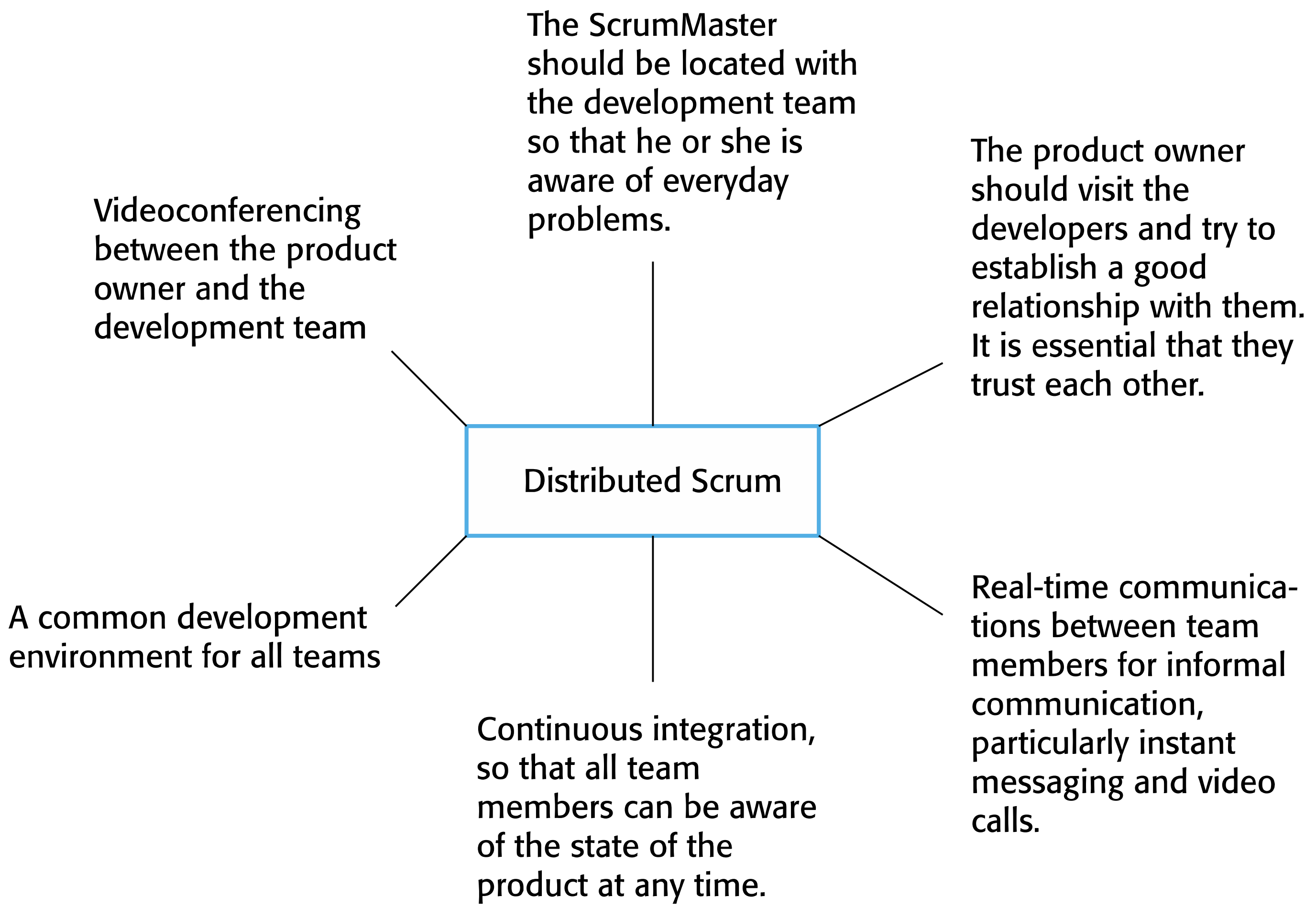
- The 'Scrum master' is a facilitator who arranges daily meetings, tracks the backlog of work to be done, records decisions, measures progress against the backlog and communicates with customers and management outside of the team.

- 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 to succeed.

Distributed Scrum


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

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 a 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.

Agile methods and 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.**

- **Two 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 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

- **Most projects include elements of 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? 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.

Key points 1

- The best way to develop software products is to use agile software engineering methods that are geared to rapid product development and delivery.
- Agile methods are based around iterative development and the minimization of overheads during the development process.
- Extreme programming (XP) is an influential agile method that introduced agile development practices such as user stories, test-first development and continuous integration. These are now mainstream software development activities.
- Scrum is an agile method that focuses on agile planning and management. Unlike XP, it does not define the engineering practices to be used. The development team may use any technical practices that they believe are appropriate for the product being developed.
- In Scrum, work to be done is maintained in a product backlog – a list of work items to be completed. Each increment of the software implements some of the work items from the product backlog.

Key points 2

- Sprints are fixed-time activities (usually 2–4 weeks) where a product increment is developed. Increments should be ‘potentially shippable’ i.e. they should not need further work before they are delivered.
- A self-organizing team is a development team that organizes the work to be done by discussion and agreement amongst team members.
- Scrum practices such as the product backlog, sprints and self-organizing teams can be used in any agile development process, even if other aspects of Scrum are not used.

Unified, Iterative and Agile

Software Development Processes

Iterative Development, Agile Modeling, and an Agile UP

- Iterative and evolutionary development contrasted with a sequential or "waterfall" lifecycle involves early programming and testing of a partial system, in repeating cycles. It also normally assumes development starts before all the requirements are defined in detail; feedback is used to clarify and improve the evolving specifications.

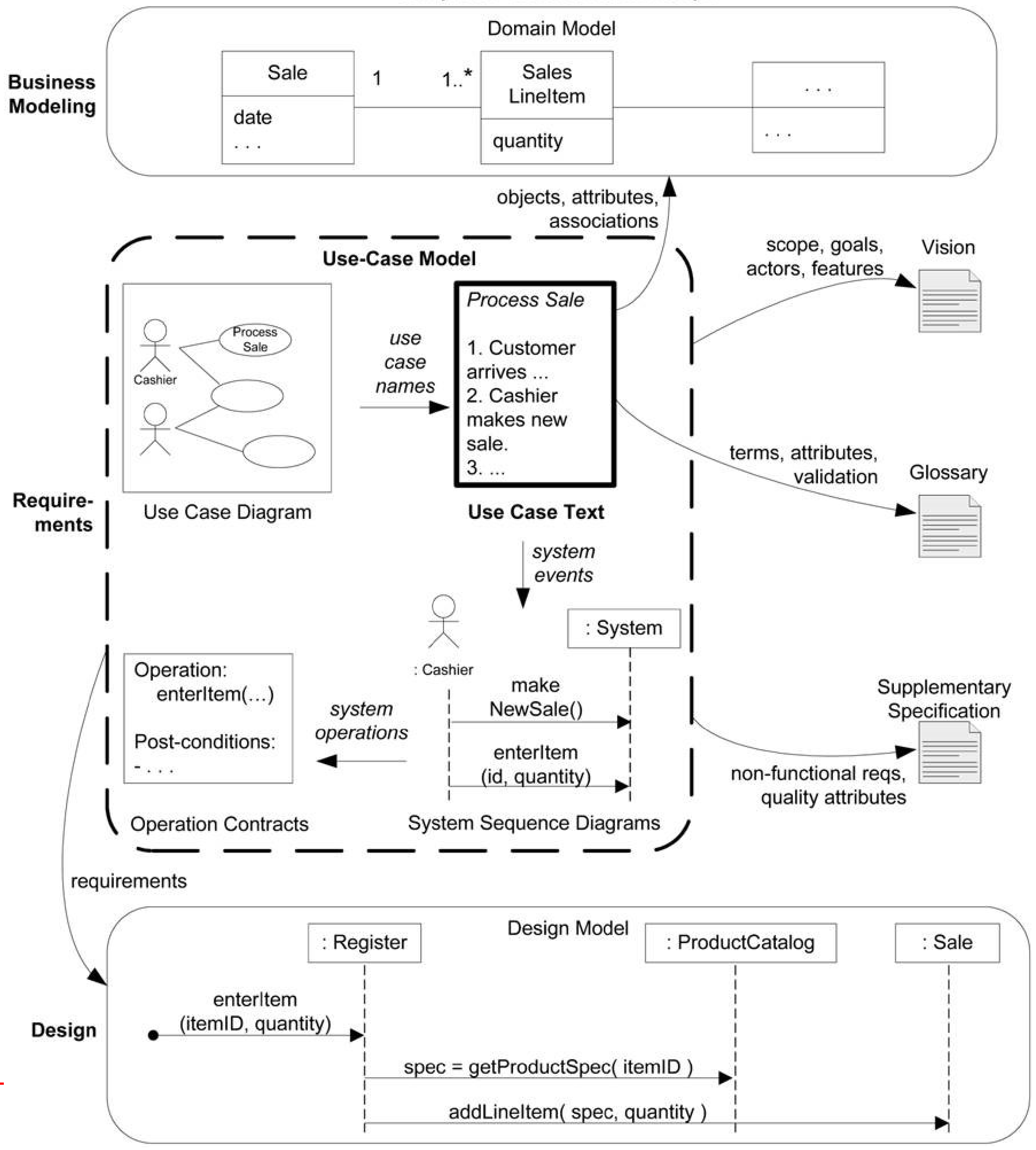
- Research demonstrates that iterative methods are associated with higher success and productivity rates, and lower defect levels.

- *Agile* practices such as *Agile Modeling* are key to applying the UML in an effective way.

UP Artifacts

What are the different artifacts in UP that are used in the blueprint?

Sample UP Artifact Relationships



What is UP? Are Other Methods Complementary?

- What is the SDP?
  - **A software development process describes an approach to building, deploying, and possibly maintaining software.**
  - **Documentation for who can do what, how and when**
- The Unified Process has emerged as a popular iterative software development process for building object-oriented systems.
- The Rational Unified Process or RUP, a detailed refinement of the Unified Process, has been widely adopted.

What is UP? Are Other Methods Complementary?

- The UP is very flexible and open, and encourages including skillful practices from other iterative methods, such as from Extreme Programming (XP), Scrum, and so forth

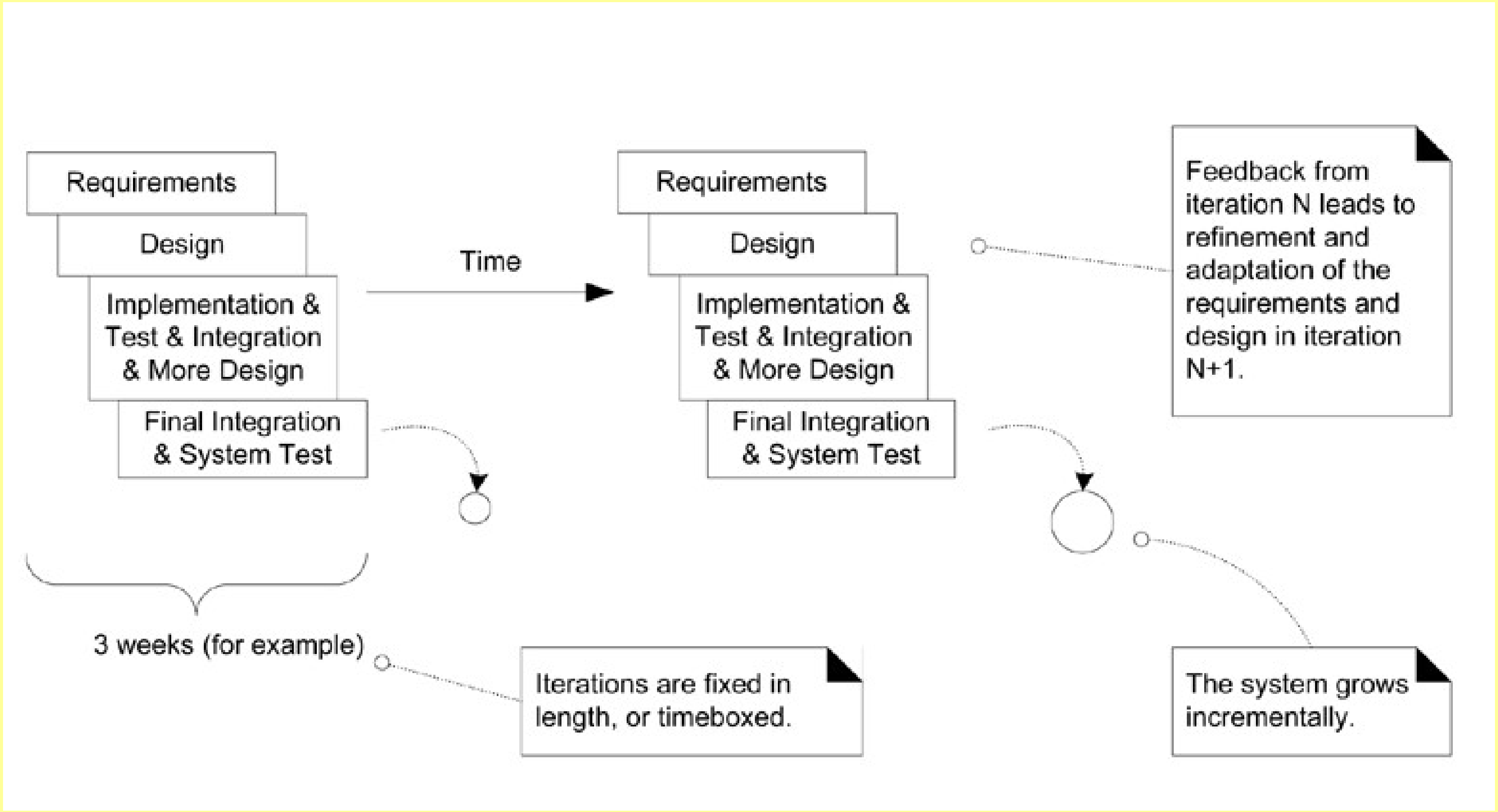
What is Iterative and Evolutionary Development?

- A key practice in UP and most other modern methods is iterative development.
- In this lifecycle approach, development is organized into a series of short, fixed-length (for example, three-week) mini-projects called iterations; the outcome of each is a tested, integrated, and executable partial system.
- Each iteration includes its own requirements analysis, design, implementation, and testing activities.

What is Iterative and Evolutionary Development?

- The iterative lifecycle is based on the successive enlargement and refinement of a system through multiple iterations, with cyclic feedback and adaptation as core drivers to converge upon a suitable system.
- The system grows incrementally over time, iteration by iteration, and thus this approach is also known as iterative and incremental development .
- Because feedback and adaptation evolve the specifications and design, it is also known as iterative and evolutionary development.

What is Iterative and Evolutionary Development?


Requirements
Design
Implementation & Test & Integration & More Design
Final Integration & System Test
Time
Requirements
Design
Implementation & Test & Integration & More Design
Final Integration & System Test
Feedback from iteration N leads to refinement and adaptation of the requirements and design in iteration N+1.
3 weeks (for example)
Iterations are fixed in length, or timeboxed.
The system grows incrementally.

How to handle change in an Iterative Project?

- Iterative and evolutionary development is based on an attitude of embracing change and adaptation as unavoidable and indeed essential drivers.

- Each iteration involves choosing a small subset of the requirements, and quickly designing, implementing, and testing.

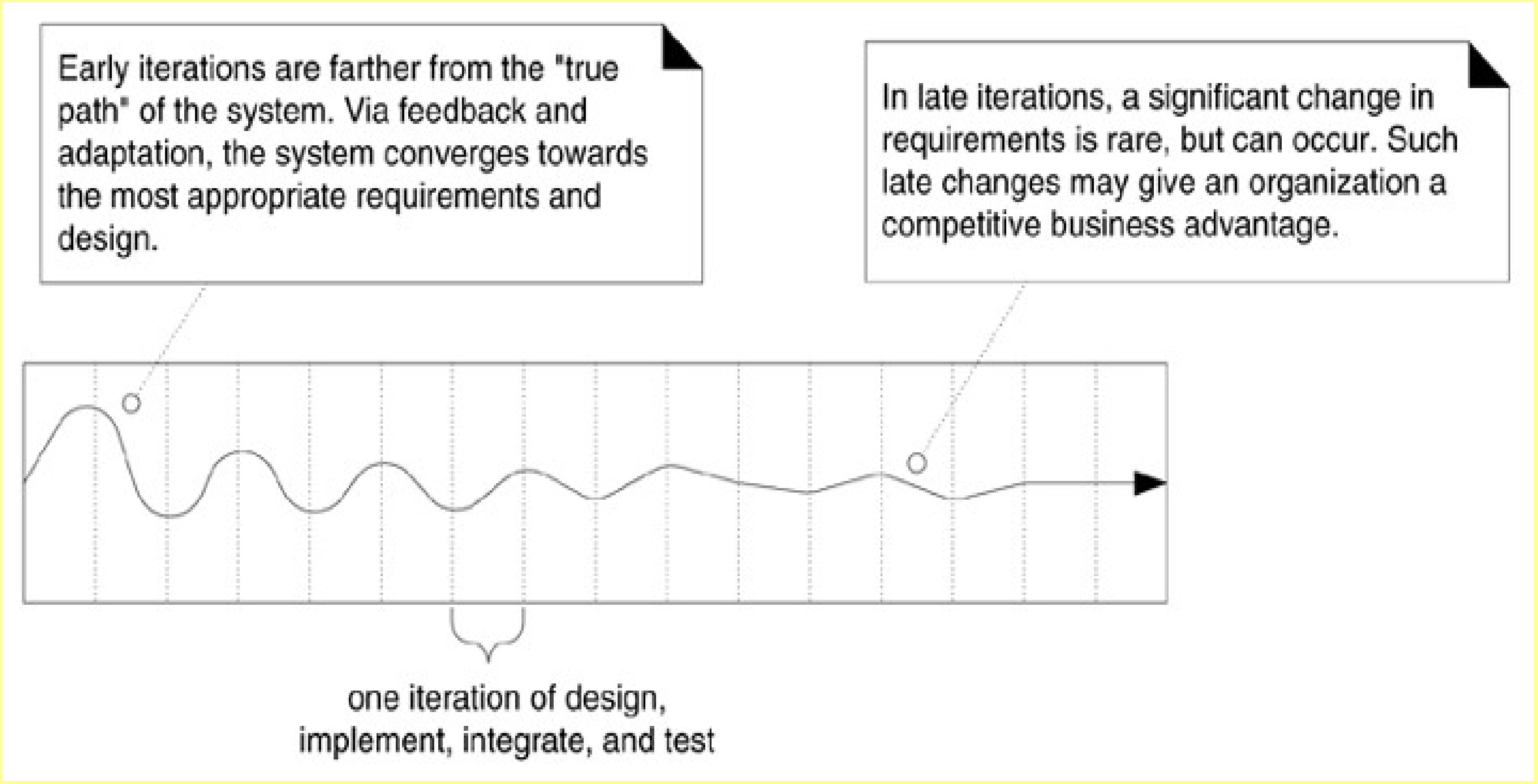
- The act of swiftly taking a small step, before all requirements are finalized, or the entire design is speculatively defined, leads to rapid feedback from the users, developers, and tests (such as load and usability tests).

How to handle change in an Iterative Project?

- End-users have a chance to quickly see a partial system and say, "Yes, that's what I asked for, but now that I try it, what I really want is something slightly different.
- Activities such as load testing will prove if the partial design and implementation are on the right path, or if in the next iteration, a change in the core architecture is required.
- Consequently, work proceeds through a series of structured build-feedback-adapt cycles.

Iterations lead toward the Targeted System

- Iterative feedback and evolution leads towards the desired system. The requirements and design instability lowers over time.


Early iterations are farther from the "true path" of the system. Via feedback and adaptation, the system converges towards the most appropriate requirements and design.
In late iterations, a significant change in requirements is rare, but can occur. Such late changes may give an organization a competitive business advantage.
one iteration of design, implement, integrate, and test

Benefits to Iterative Development

1. less project failure, better productivity, and lower defect rates; shown by research into iterative and evolutionary methods

2. early rather than late mitigation of high risks (technical, requirements, objectives, usability, and so forth)

3. early visible progress

4. early feedback, user engagement, and adaptation, leading to a refined system that more closely meets the real needs of the stakeholders

5. managed complexity; the team is not overwhelmed by "analysis paralysis" or very long and complex steps

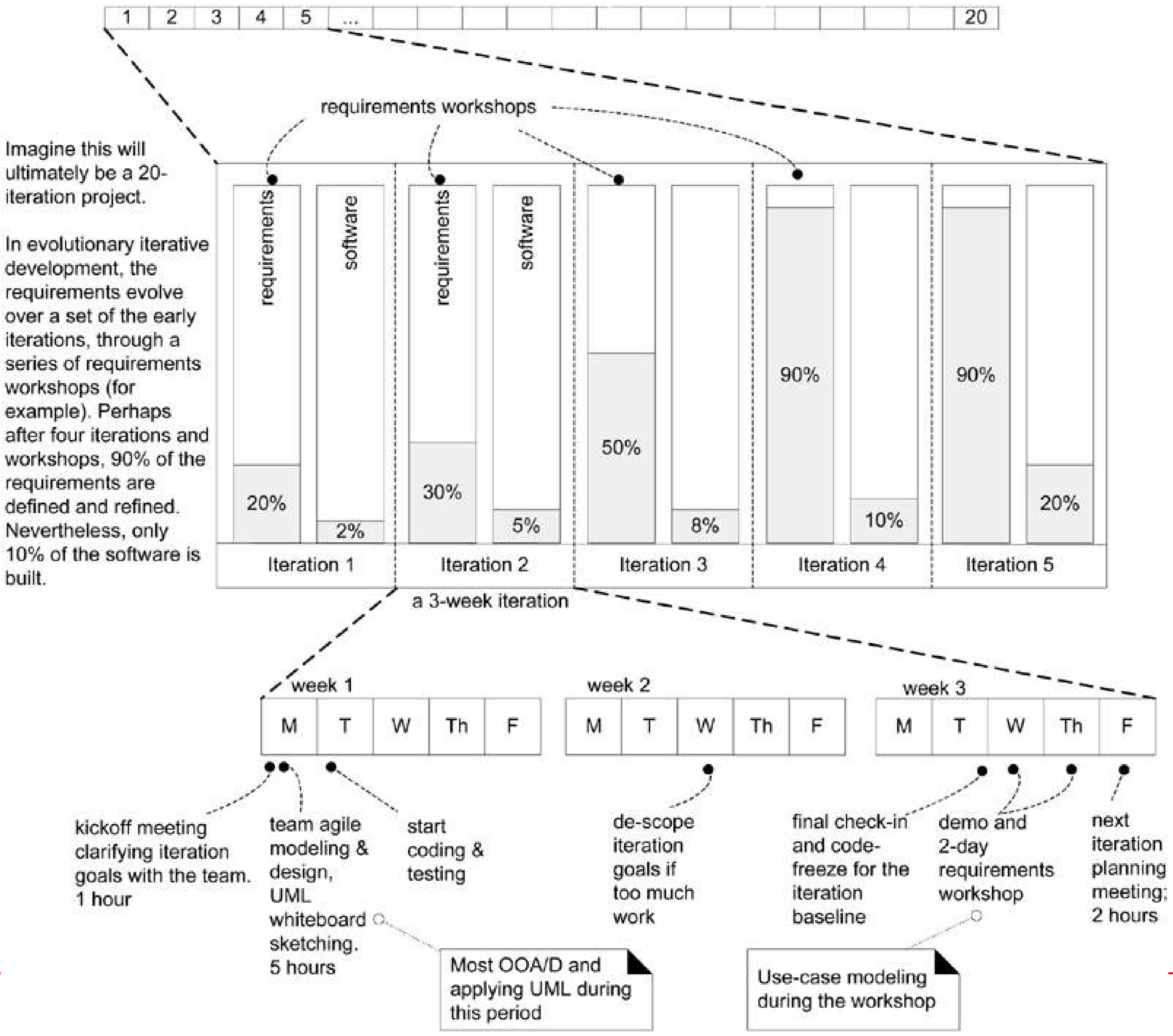
6. the learning within an iteration can be methodically used to improve the development process itself, iteration by iteration

How Long Should an Iteration Be?

- Most iterative methods recommend an iteration length between two and six weeks.
- Small steps, rapid feedback, and adaptation are central ideas in iterative development; long iterations subvert the core motivation for iterative development and increase project risk.
- A very long timeboxed iteration misses the point of iterative development. Short-iteration is good.

Agile methods and Attitudes

- Agile development methods usually apply timeboxed iterative and evolutionary development, employ adaptive planning, promote incremental delivery, and include other values and practices that encourage agility rapid and flexible response to change.
- Short timeboxed iterations with evolutionary refinement of plans, requirements, and design is a basic practice that agile methods share. In addition, they promote practices and principles that reflect an agile sensibility of simplicity, lightness, communication, self-organizing teams, and more.



Imagine this will ultimately be a 20-iteration project.

In evolutionary iterative development, the requirements evolve over a set of the early iterations, through a series of requirements workshops (for example). Perhaps after four iterations and workshops, 90% of the requirements are defined and refined. Nevertheless, only 10% of the software is built.

1
2
3
4
5
...
20
requirements workshops
requirements
software
20%
2%
30%
5%
50%
8%
90%
10%
90%
20%
Iteration 1
Iteration 2
Iteration 3
Iteration 4
Iteration 5
a 3-week iteration
week 1
week 2
week 3
M
T
W
Th
F
kickoff meeting clarifying iteration goals with the team. 1 hour
team agile modeling & design, UML whiteboard sketching. 5 hours
start coding & testing
de-scope iteration goals if too much work
final check-in and code-freeze for the iteration baseline
demo and 2-day requirements workshop
next iteration planning meeting; 2 hours
Most OOA/D and applying UML during this period
Use-case modeling during the workshop

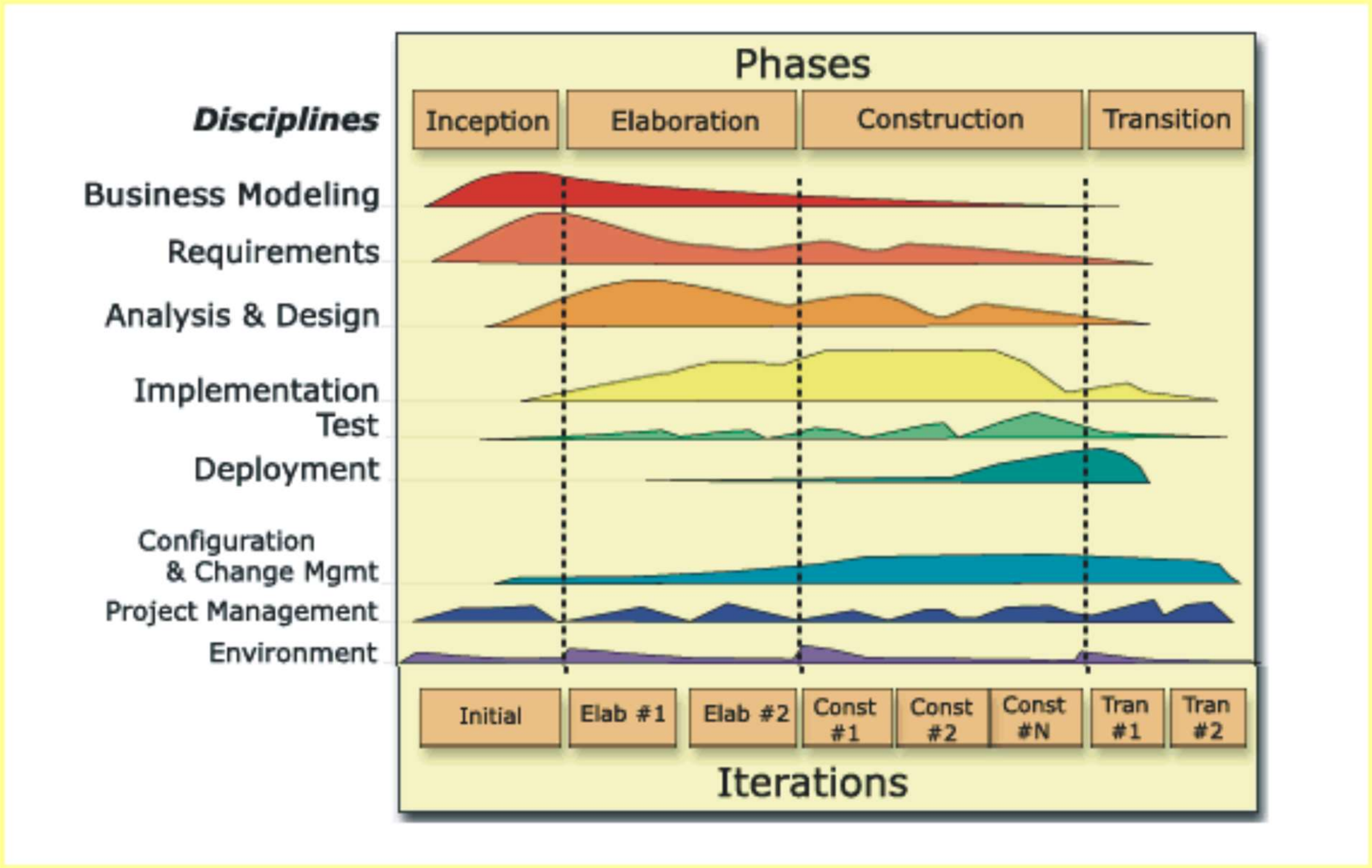
What is an Agile UP?

- **UP was meant to be adopted and applied in the spirit of adaptability and lightness is an agile UP. Some examples of how this applies:**
  - Prefer a small set of UP activities and artifacts. Some projects will benefit more than others, but, in general, keep it simple.
  - Remember that all UP artifacts are optional, and avoid creating them unless they add value. Focus on early programming, not early documenting.
  - Since the UP is iterative and evolutionary, requirements and designs are not completed before implementation. They adaptively emerge through a series of iterations, based on feedback.
  - Apply the UML with agile modeling practices.

UP Phases

- **A UP project organizes the work and iterations across four major phases:**
  - Inception approximate vision, business case, scope, vague estimates.
  - Elaboration refined vision, iterative implementation of the core architecture, resolution of high risks, identification of most requirements and scope, more realistic estimates.
  - Construction iterative implementation of the remaining lower risk and easier elements, and preparation for deployment.
  - Transition beta tests, deployment.

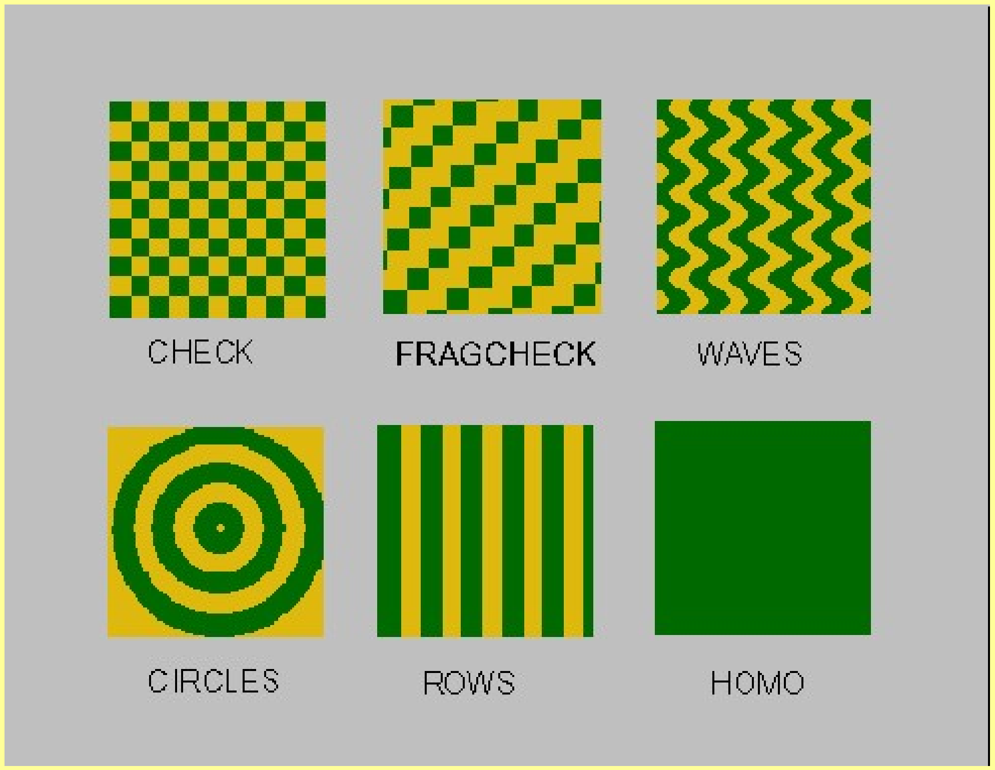
Schedule-oriented terms in the UP



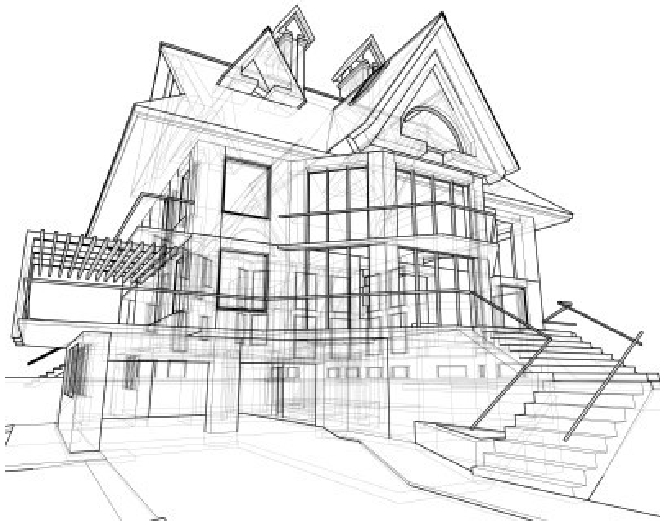
Design Patterns: Reusable Design Elements

Patterns in the Civil Architectures: House, Restaurant, High Rise Building, Factory, etc.

Patterns Everywhere

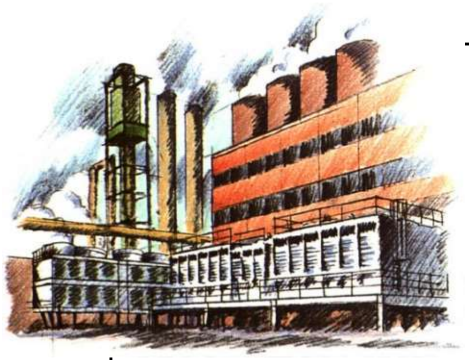


Patterns Everywhere



Object-Oriented Design Patterns

We will discuss how to Model and utilize roughly 15 Design Patterns
in our Design Models for the 3 Categories in GoF Catalog



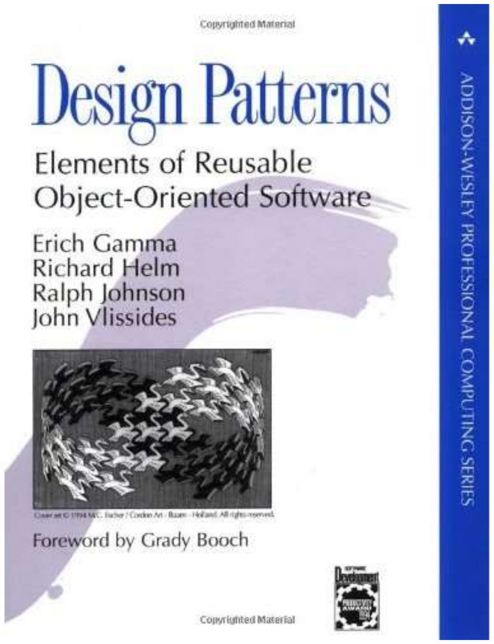
1. Creational

2. Structural

3. Behavioral

Object-Oriented Design Patterns

Design Patterns Catalogs


Copyrighted Material

Design Patterns

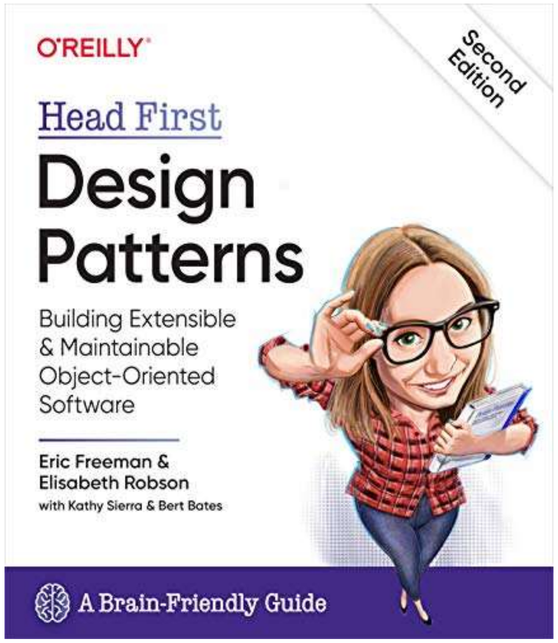
Elements of Reusable Object-Oriented Software

Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides

Foreword by Grady Booch

Copyrighted Material

ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES


O'REILLY

Second Edition

Head First

Design Patterns

Building Extensible & Maintainable Object-Oriented Software

Eric Freeman & Elisabeth Robson
with Kathy Sierra & Bert Bates

A Brain-Friendly Guide

UML to Build Flexible Designs

Healthy Design Models must leverage UML and the Design Patterns to produce design models that can adapt to meet the unforeseen future requirements

UP & UML

- **We want to**
  - Apply Object-Oriented Development Methodology (UP and UML) for software product development
  - Apply Agile methods for development as well as project planning for software product releases

References

- Sommerville, I. 2020. Engineering Software Products (1st Edition). UK: Pearson. [ISBN: 978-0-1352-1064-2]
- Sommerville, I. 2016. Software Engineering (10th Edition). UK: Pearson. [ISBN: 978-0-1339-4303-0]
- Larman, C. 2004. Applying UML and Patterns (3rd edition). New York, NY: Pearson. [ISBN: 978-0-1314-8906-6
- Pressman, R. 2019. Software Engineering: A Practitioner's Approach (8th edition). New York, NY: McGraw Hill. [ISBN-13: 978-0-0780-2212-8]
- McCarthy, B. 2018. Product Manager vs. Project Manager. Sebastopol, CA: O'Reilly Media. [ISBN: 978-1-49203-444-5]