Hochschule
Bonn-Rhein-Sieg
University of Applied Sciences

b-it Bonn-Aachen International Center for Information Technology

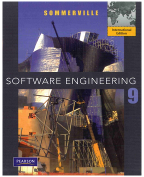
Institute for AI and Autonomous Systems

# Software Engineering Methodologies

## How to Manage the Development Process

Dr. Alex Mitrevski
Master of Autonomous Systems

# Structure

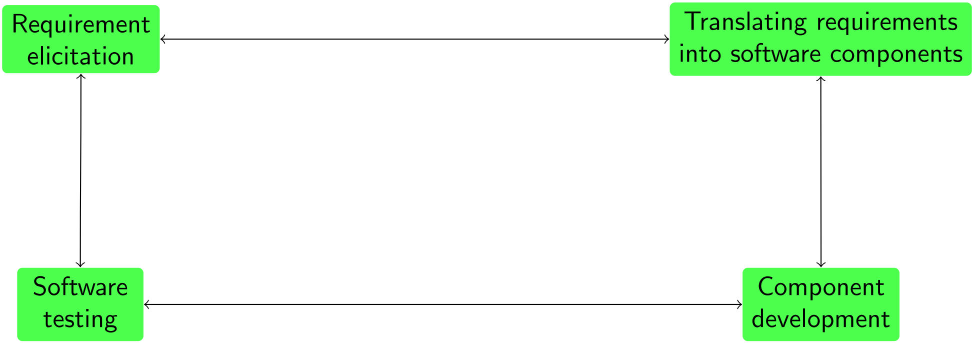


- Preliminaries
- Software engineering methodologies
- Agile development
- Paradigm challenges and implications for robotics

# Preliminaries

# Essential Steps in Software Development

- In our first lecture, we mentioned the following steps as being essential in software engineering



Requirement elicitation

Translating requirements into software components

Software testing

Component development

- There are, however, a variety of ways to perform these steps

# Software Engineering Methodology

A software engineering methodology is a domain-independent collection of guidelines and procedures that define how software should be designed, developed, tested, and managed throughout its lifecycle

# Software Engineering Methodology

A software engineering methodology is a domain-independent collection of guidelines and procedures that define how software should be designed, developed, tested, and managed throughout its lifecycle

- Software development, starting from the conceptualisation and up to the deployment, should be governed by a **software engineering methodology** (aka a software process)
  - Without a suitable methodology, the process is highly unlikely to lead to well-developed software

# Software Engineering Methodology

A software engineering methodology is a domain-independent collection of guidelines and procedures that define how software should be designed, developed, tested, and managed throughout its lifecycle

- Software development, starting from the conceptualisation and up to the deployment, should be governed by a **software engineering methodology** (aka a software process)
  - Without a suitable methodology, the process is highly unlikely to lead to well-developed software

- A variety of software engineering methodologies exist in the literature — these are **domain-independent** and **follow the complete software lifecycle**

# Software Engineering Methodology

A software engineering methodology is a domain-independent collection of guidelines and procedures that define how software should be designed, developed, tested, and managed throughout its lifecycle

- Software development, starting from the conceptualisation and up to the deployment, should be governed by a **software engineering methodology** (aka a software process)
  - Without a suitable methodology, the process is highly unlikely to lead to well-developed software
- A variety of software engineering methodologies exist in the literature — these are **domain-independent** and **follow the complete software lifecycle**
- From an engineering point of view, **robots do not differ from other software systems** — robot software development should follow the same procedures as other software products
  - The practical implementation of individual steps along the process may differ though

# Who Decides on the Methodology to Follow?

- The decision on the methodology usually depends on **the context in which software is developed**:
  - Large(r) organisations are likely to already have a methodology that all team members need to adopt
  - Small(er) teams are usually flexible in deciding on an appropriate methodology

# Who Decides on the Methodology to Follow?

- The decision on the methodology usually depends on **the context in which software is developed**:
  - Large(r) organisations are likely to already have a methodology that all team members need to adopt
  - Small(er) teams are usually flexible in deciding on an appropriate methodology
- **There is no one-size-fits-all methodology** — how to manage the development process may depend on several factors, such as:

# Who Decides on the Methodology to Follow?

- The decision on the methodology usually depends on **the context in which software is developed**:
  - Large(r) organisations are likely to already have a methodology that all team members need to adopt
  - Small(er) teams are usually flexible in deciding on an appropriate methodology
- **There is no one-size-fits-all methodology** — how to manage the development process may depend on several factors, such as:
  - **the size of the team developing the software** (larger teams may need more fine-grained management than smaller teams)

# Who Decides on the Methodology to Follow?

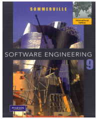
- The decision on the methodology usually depends on **the context in which software is developed**:
  - Large(r) organisations are likely to already have a methodology that all team members need to adopt
  - Small(er) teams are usually flexible in deciding on an appropriate methodology
- **There is no one-size-fits-all methodology** — how to manage the development process may depend on several factors, such as:
  - **the size of the team developing the software** (larger teams may need more fine-grained management than smaller teams)
  - **the associated software risks and costs** (robots interacting with people pose different risks than extraterrestrial exploration robots)

# Who Decides on the Methodology to Follow?

- The decision on the methodology usually depends on **the context in which software is developed**:
  - Large(r) organisations are likely to already have a methodology that all team members need to adopt
  - Small(er) teams are usually flexible in deciding on an appropriate methodology
- **There is no one-size-fits-all methodology** — how to manage the development process may depend on several factors, such as:
  - **the size of the team developing the software** (larger teams may need more fine-grained management than smaller teams)
  - **the associated software risks and costs** (robots interacting with people pose different risks than extraterrestrial exploration robots)
  - **the availability of users during the development process** (frequent user feedback will shape the development differently than feedback only at specific milestones)

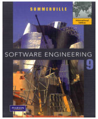
# Software Engineering Methodologies

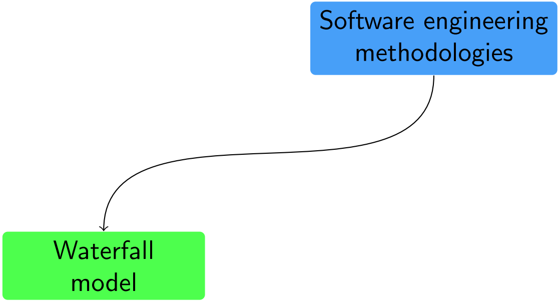
# Methodologies Overview



Software engineering methodologies

# Methodologies Overview

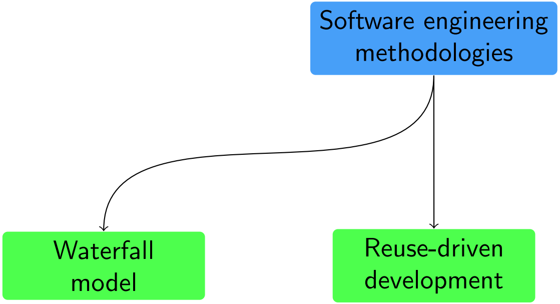




Software engineering methodologies

Waterfall model

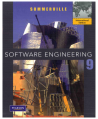
# Methodologies Overview

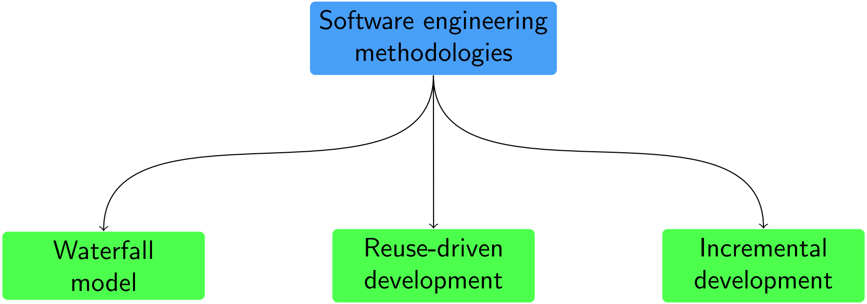

Software engineering methodologies

Waterfall model

Reuse-driven development

# Methodologies Overview





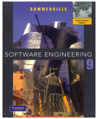
Software engineering methodologies

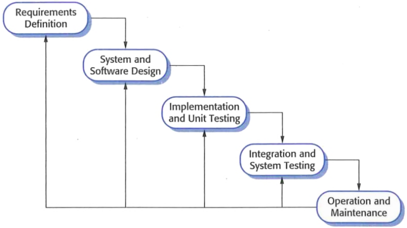
Waterfall model

Reuse-driven development

Incremental development

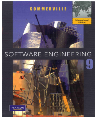
# Waterfall Model

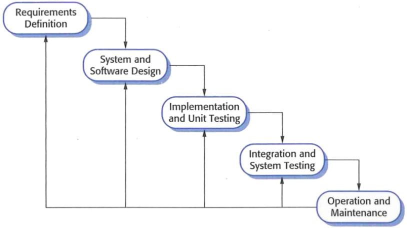



Requirements Definition
System and Software Design
Implementation and Unit Testing
Integration and System Testing
Operation and Maintenance

- The waterfall model is a **mostly sequential engineering methodology — the development proceeds to the next step only when the current step is completed**
  - But later stages may notice issues with the previous steps, which then requires going a step back and reworking certain aspects

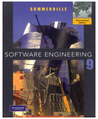
# Waterfall Model

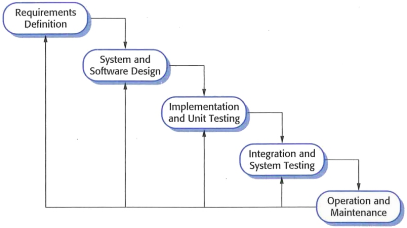



Requirements Definition
System and Software Design
Implementation and Unit Testing
Integration and System Testing
Operation and Maintenance

- The waterfall model is a **mostly sequential engineering methodology — the development proceeds to the next step only when the current step is completed**
  - But later stages may notice issues with the previous steps, which then requires going a step back and reworking certain aspects
- The completion of each stage is **followed by producing documentation** that is approved before the next stage of the process can start
  - Corrections in previous steps require changes in the documentation as well

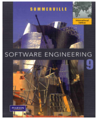
# Waterfall Model

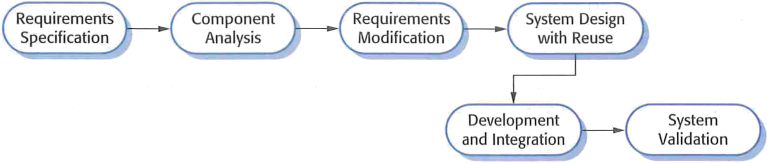



Requirements Definition
System and Software Design
Implementation and Unit Testing
Integration and System Testing
Operation and Maintenance

- The waterfall model is a **mostly sequential engineering methodology — the development proceeds to the next step only when the current step is completed**
  - But later stages may notice issues with the previous steps, which then requires going a step back and reworking certain aspects
- The completion of each stage is **followed by producing documentation** that is approved before the next stage of the process can start
  - Corrections in previous steps require changes in the documentation as well
- The waterfall model is a **plan-driven process**

# Reuse-Oriented Development

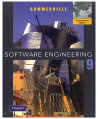


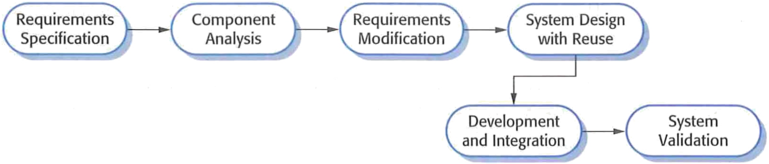


Requirements Specification
Component Analysis
Requirements Modification
System Design with Reuse
Development and Integration
System Validation

- In reuse-oriented development, **software is designed and developed so that existing components can be reused**

# Reuse-Oriented Development

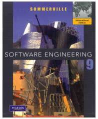


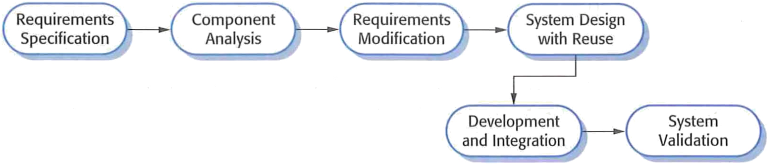

Requirements Specification
Component Analysis
Requirements Modification
System Design with Reuse
Development and Integration
System Validation

- In reuse-oriented development, **software is designed and developed so that existing components can be reused**

- An important part of the engineering process using this methodology is thus **the identification of suitable existing components** and **the update of the initial requirements to fit the constraints of the existing components**

# Reuse-Oriented Development





Requirements Specification

Component Analysis

Requirements Modification

System Design with Reuse

Development and Integration

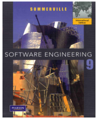
System Validation

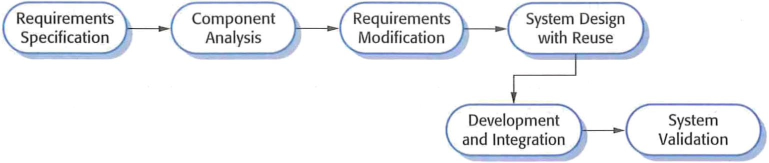
- In reuse-oriented development, **software is designed and developed so that existing components can be reused**

- An important part of the engineering process using this methodology is thus **the identification of suitable existing components** and **the update of the initial requirements to fit the constraints of the existing components**

- Such components are then integrated with custom software that needs to be developed

# Reuse-Oriented Development




Requirements Specification → Component Analysis → Requirements Modification → System Design with Reuse → Development and Integration → System Validation

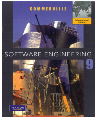
- In reuse-oriented development, **software is designed and developed so that existing components can be reused**

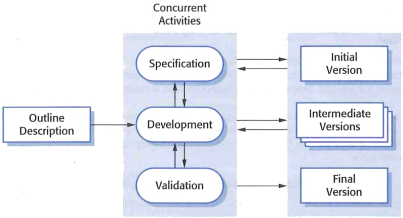
- An important part of the engineering process using this methodology is thus **the identification of suitable existing components** and **the update of the initial requirements to fit the constraints of the existing components**

- Such components are then integrated with custom software that needs to be developed

- **Robotics software development almost always involves elements of reuse-oriented development** — as we saw in the last lecture, there are many standard components that are used in robot software

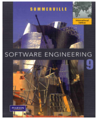
# Incremental Development Model

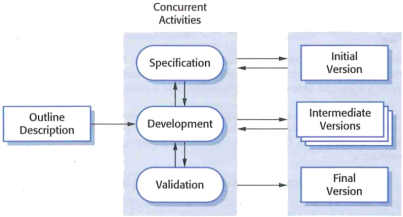




- Incremental development is a process that **interleaves different stages of the design and development process** — a strict sequential structure is not followed

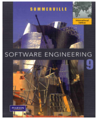
# Incremental Development Model

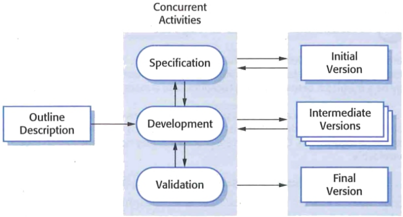




- ▶ Incremental development is a process that **interleaves different stages of the design and development process** — a strict sequential structure is not followed
- ▶ An essential element of incremental development is that **prototypes are developed early** and **are then iteratively improved based on frequent user feedback**

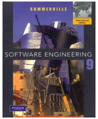
# Incremental Development Model



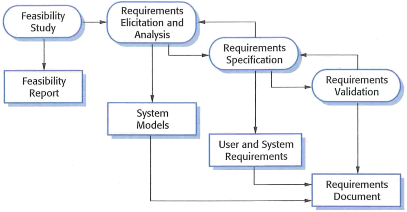


- Incremental development is a process that **interleaves different stages of the design and development process** — a strict sequential structure is not followed
- An essential element of incremental development is that **prototypes are developed early** and **are then iteratively improved based on frequent user feedback**
- Unlike in the waterfall model, incremental development **focuses more on the development and deployment** and **less on documenting the process**

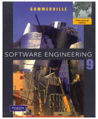
# Requirements Engineering

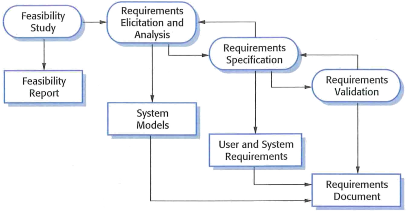


- Regardless of the concrete methodologies that is followed, it is necessary to **identify the requirements of the users of a robot system** (or even just a small component)



# Requirements Engineering

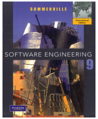


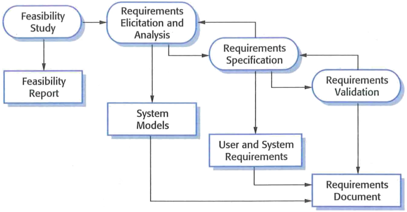


- Regardless of the concrete methodologies that is followed, it is necessary to **identify the requirements of the users of a robot system** (or even just a small component)

- This is achieved with a requirements engineering process, during which **the needs and expectations of the system are collected and analysed**

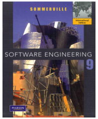
# Requirements Engineering

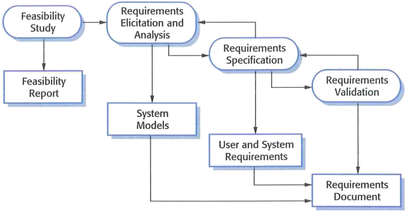




- Regardless of the concrete methodologies that is followed, it is necessary to **identify the requirements of the users of a robot system** (or even just a small component)
- This is achieved with a requirements engineering process, during which **the needs and expectations of the system are collected and analysed**
- During the process, it is essential to **ensure that the requirements are correctly understood**, but also to **verify that their realisation is feasible**

# Requirements Engineering





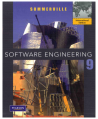
- Regardless of the concrete methodologies that is followed, it is necessary to **identify the requirements of the users of a robot system** (or even just a small component)

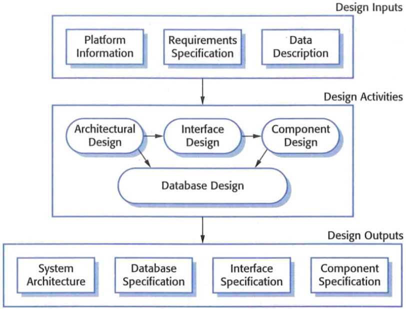
- This is achieved with a requirements engineering process, during which **the needs and expectations of the system are collected and analysed**

- During the process, it is essential to **ensure that the requirements are correctly understood**, but also to **verify that their realisation is feasible**

- The requirements engineering process usually ends with documentation that clearly specifies the requirements — **a requirement specification**

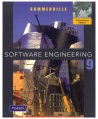
# Software System Design

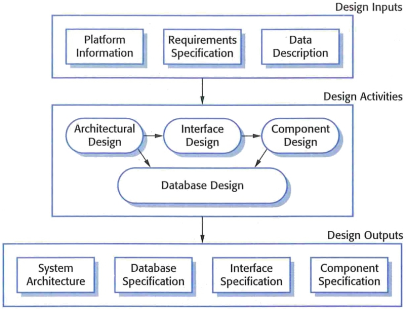




- The requirements engineering process is followed by a **system design process**

# Software System Design





Design Inputs

Platform Information

Requirements Specification

Data Description

Design Activities

Architectural Design

Interface Design

Component Design

Database Design

Design Outputs

System Architecture

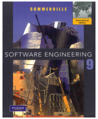
Database Specification

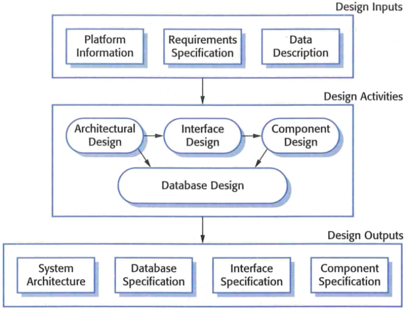
Interface Specification

Component Specification

- The requirements engineering process is followed by a **system design process**
- During this process, **the operational, system, and technical architectures are designed**
  - As discussed in our session on robot software architectures, **the operational architecture is typically based on standard models**

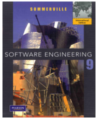
# Software System Design

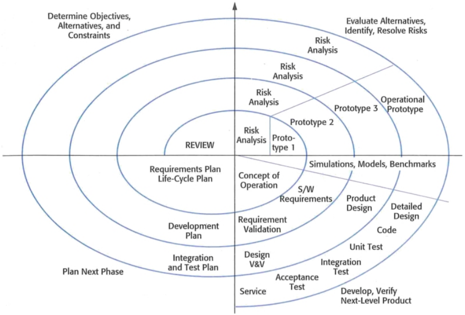




- The requirements engineering process is followed by a **system design process**
- During this process, **the operational, system, and technical architectures are designed**
  - As discussed in our session on robot software architectures, **the operational architecture is typically based on standard models**
- The design step may end with an architecture design document (in the waterfall or reuse-oriented models) or an initial system prototype (in incremental development)

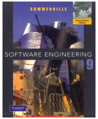
# Risk Management: Boehm's Spiral Model

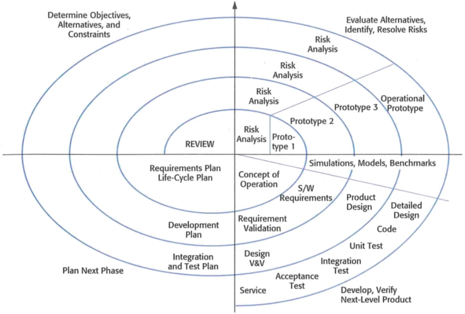




- The software engineering process is always **associated with risks**, particularly due to changing requirements, but also due to internal project risks

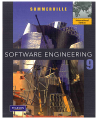
# Risk Management: Boehm's Spiral Model

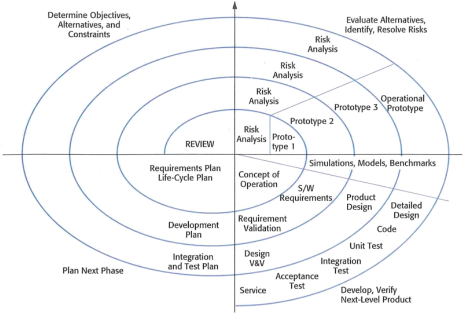




- The software engineering process is always **associated with risks**, particularly due to changing requirements, but also due to internal project risks
- Boehm's spiral model is particularly developed for risk management — **risks are re-evaluted at every stage of the design and development process**

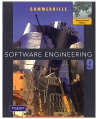
# Risk Management: Boehm's Spiral Model

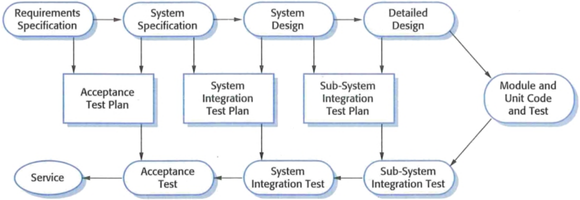




- The software engineering process is always **associated with risks**, particularly due to changing requirements, but also due to internal project risks
- Boehm's spiral model is particularly developed for risk management — **risks are re-evaluted at every stage of the design and development process**
- According to this model, **the evaluation of risks directly informs the subsequent development**
  - The individual development cycles can be based on any of the three development methodologies that we discussed before

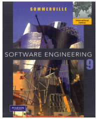
# Software Testing Stages

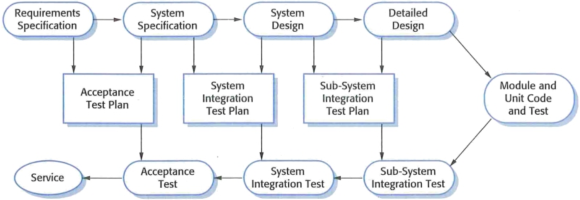



Requirements Specification
System Specification
System Design
Detailed Design
Acceptance Test Plan
System Integration Test Plan
Sub-System Integration Test Plan
Module and Unit Code and Test
Service
Acceptance Test
System Integration Test
Sub-System Integration Test

- The software engineering process needs to be accompanied by **software testing**
  - In robotics, we have the additional challenge that testing needs to involve the hardware platform for which the software is developed

# Software Testing Stages

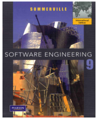


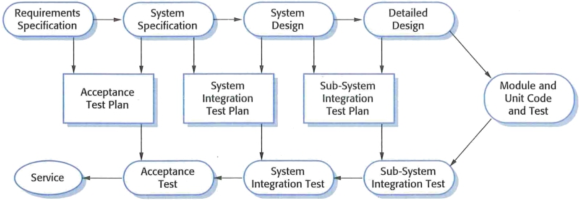

Requirements Specification
System Specification
System Design
Detailed Design
Acceptance Test Plan
System Integration Test Plan
Sub-System Integration Test Plan
Module and Unit Code and Test
Service
Acceptance Test
System Integration Test
Sub-System Integration Test

- The software engineering process needs to be accompanied by **software testing**
  - In robotics, we have the additional challenge that testing needs to involve the hardware platform for which the software is developed

- Software tests can be done at different levels — we have **component (unit), integration, and system tests** (as briefly discussed in our introductory lecture)

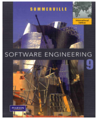
# Software Testing Stages

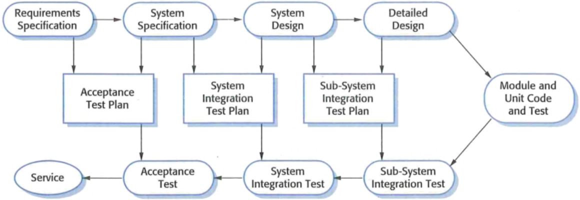



Requirements Specification
System Specification
System Design
Detailed Design
Acceptance Test Plan
System Integration Test Plan
Sub-System Integration Test Plan
Module and Unit Code and Test
Service
Acceptance Test
System Integration Test
Sub-System Integration Test

- The software engineering process needs to be accompanied by **software testing**
  - In robotics, we have the additional challenge that testing needs to involve the hardware platform for which the software is developed
- Software tests can be done at different levels — we have **component (unit), integration, and system tests** (as briefly discussed in our introductory lecture)
- **The testing process should also involve the users of a robot**, such that there are two types of tests that are performed with users:

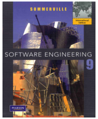
# Software Testing Stages

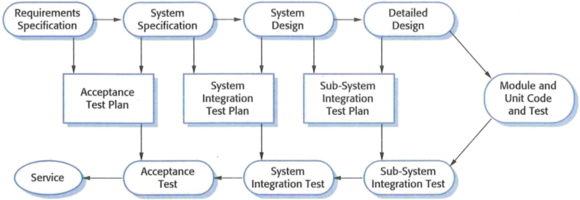



Requirements Specification
System Specification
System Design
Detailed Design
Acceptance Test Plan
System Integration Test Plan
Sub-System Integration Test Plan
Module and Unit Code and Test
Service
Acceptance Test
System Integration Test
Sub-System Integration Test

- The software engineering process needs to be accompanied by **software testing**
  - In robotics, we have the additional challenge that testing needs to involve the hardware platform for which the software is developed
- Software tests can be done at different levels — we have **component (unit), integration, and system tests** (as briefly discussed in our introductory lecture)
- **The testing process should also involve the users of a robot**, such that there are two types of tests that are performed with users:
  - **Alpha (aka acceptance) testing**, where the system is evaluated with respect to the requirements

# Software Testing Stages




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

- The software engineering process needs to be accompanied by **software testing**
  - In robotics, we have the additional challenge that testing needs to involve the hardware platform for which the software is developed

- Software tests can be done at different levels — we have **component (unit), integration, and system tests** (as briefly discussed in our introductory lecture)

- **The testing process should also involve the users of a robot**, such that there are two types of tests that are performed with users:
  - **Alpha (aka acceptance) testing**, where the system is evaluated with respect to the requirements
  - **Beta testing**, where a larger base of potential users interacts with the system (typically for a prolonged period) and can report issues to the developers

# Agile Development

# Agile Development Motivation

- Agile development is a **rapid development paradigm** that focuses on fast development and continuous integration of changes
  - The paradigm is an instantiation of the incremental development process that we discussed before

# Agile Development Motivation

- Agile development is a **rapid development paradigm** that focuses on fast development and continuous integration of changes
  - The paradigm is an instantiation of the incremental development process that we discussed before
- The primary motivation for agile development is change management
  - Changes are **naturally integrated into the development process** since software is developed incrementally, with constant user involvement to guide the development process

# Agile Development Motivation

- Agile development is a **rapid development paradigm** that focuses on fast development and continuous integration of changes
  - The paradigm is an instantiation of the incremental development process that we discussed before
- The primary motivation for agile development is change management
  - Changes are **naturally integrated into the development process** since software is developed incrementally, with constant user involvement to guide the development process
- Agile development is a **collection of methods** rather than a monolithic structure of conventions and practices

# The Agile Manifesto[1]

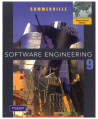
"We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

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

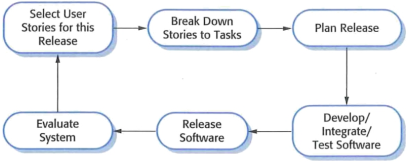
That is, while there is value in the items on the right, we value the items on the left more."

[1] https://agilemanifesto.org

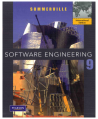
# Extreme Programming (XP)



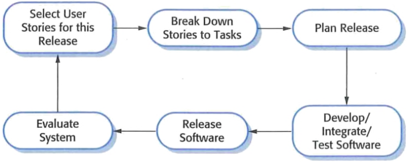
- XP is one particular agile development method
  - The name "extreme" refers to the fact that best practices for development are pushed to the extreme


Select User Stories for this Release
Break Down Stories to Tasks
Plan Release
Develop/ Integrate/ Test Software
Release Software
Evaluate System

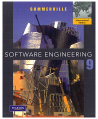
# Extreme Programming (XP)

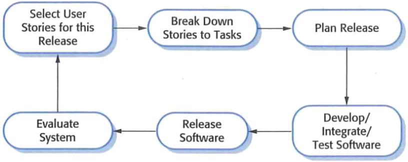


- XP is one particular agile development method
  - The name "extreme" refers to the fact that best practices for development are pushed to the extreme
- In this paradigm, requirements are represented as **user stories** from which **concrete tasks are derived**
  - Stories are developed on so-called **story cards**


Select User Stories for this Release
Break Down Stories to Tasks
Plan Release
Develop/ Integrate/ Test Software
Release Software
Evaluate System

# Extreme Programming (XP)

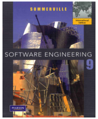


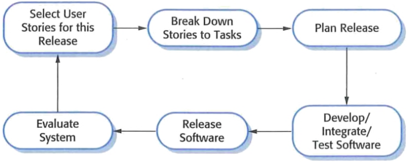


Select User Stories for this Release
Break Down Stories to Tasks
Plan Release
Develop/ Integrate/ Test Software
Release Software
Evaluate System

- XP is one particular agile development method
  - The name "extreme" refers to the fact that best practices for development are pushed to the extreme
- In this paradigm, requirements are represented as **user stories** from which **concrete tasks are derived**
  - Stories are developed on so-called **story cards**
- Version releases in extreme programming are **done at frequent rates**, such that **users are directly involved in the development process** (to define development priorities and acceptance tests)

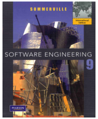
# Extreme Programming (XP)

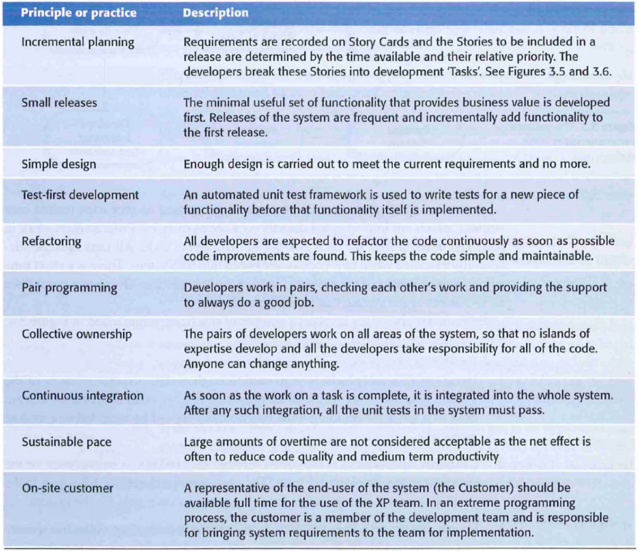



Select User Stories for this Release
Break Down Stories to Tasks
Plan Release
Develop/ Integrate/ Test Software
Release Software
Evaluate System

- XP is one particular agile development method
  - The name "extreme" refers to the fact that best practices for development are pushed to the extreme
- In this paradigm, requirements are represented as **user stories** from which **concrete tasks are derived**
  - Stories are developed on so-called **story cards**
- Version releases in extreme programming are **done at frequent rates**, such that **users are directly involved in the development process** (to define development priorities and acceptance tests)
- Extreme programming performs **continuous automated testing** (new versions are accepted only if tests pass) and **regular refactoring** (to handle the complexity of rapidly changing software)

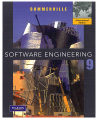
# Extreme Programming Practices





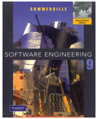
| Principle or practice | Description |
|---|---|
| Incremental planning | Requirements are recorded on Story Cards and the Stories to be included in a release are determined by the time available and their relative priority. The developers break these Stories into development 'Tasks'. See Figures 3.5 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. |
| 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. |

# Pair Programming



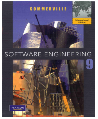
- One of the main principles of extreme programming is a practice called **pair programming**, which involves a pair of programmers jointly working on the same component — **including sitting and programming together**

# Pair Programming



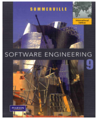
- One of the main principles of extreme programming is a practice called **pair programming**, which involves a pair of programmers jointly working on the same component — **including sitting and programming together**
- Pair programming has various benefits compared to individual development:
  - **The code is written and looked at by two people** — likelier to discover bugs early
  - **There is shared knowledge of the code** — makes the development team more resistant to people leaving the team
  - **Enables experience sharing** — if more experienced developers are paired with less experienced ones

# Pair Programming

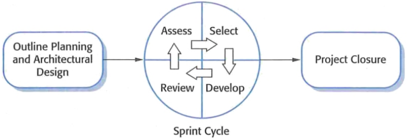


- One of the main principles of extreme programming is a practice called **pair programming**, which involves a pair of programmers jointly working on the same component — **including sitting and programming together**
- Pair programming has various benefits compared to individual development:
  - **The code is written and looked at by two people** — likelier to discover bugs early
  - **There is shared knowledge of the code** — makes the development team more resistant to people leaving the team
  - **Enables experience sharing** — if more experienced developers are paired with less experienced ones
- Pair programming may, however, be less useful for senior developers or even negatively affect their productivity

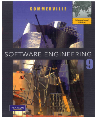
# Scrum

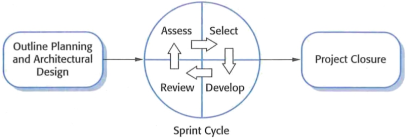


- Scrum is an **agile development management methodology** and can be used in conjunction with other agile methods (such as extreme programming)



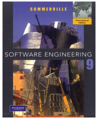
# Scrum

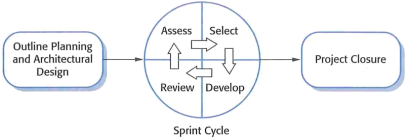




- Scrum is an **agile development management methodology** and can be used in conjunction with other agile methods (such as extreme programming)
- The idea behind scrum is to organise the complete development process into iterative cycles called **sprints**

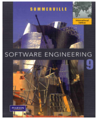
# Scrum

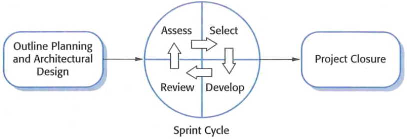




- Scrum is an **agile development management methodology** and can be used in conjunction with other agile methods (such as extreme programming)
- The idea behind scrum is to organise the complete development process into iterative cycles called **sprints**
- While a sprint is ongoing, short daily **stand-up meetings** are conducted **with all team members**
  - The meetings are guided by a **scrum master**, who keeps track of the development progress and communicates this with users and outside managers

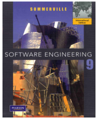
# Scrum





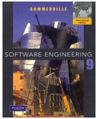
- Scrum is an **agile development management methodology** and can be used in conjunction with other agile methods (such as extreme programming)
- The idea behind scrum is to organise the complete development process into iterative cycles called **sprints**
- While a sprint is ongoing, short daily **stand-up meetings** are conducted **with all team members**
  - The meetings are guided by a **scrum master**, who keeps track of the development progress and communicates this with users and outside managers
- We use scrum frequently in the b-it-bots team, particularly before and during competitions

# Scrum Sprints



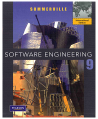
- Scrum sprints are **short development cycles** that typically last only a few weeks

# Scrum Sprints



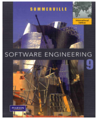
- Scrum sprints are **short development cycles** that typically last only a few weeks
- Each sprint starts with an evaluation of the tasks (called **product backlog** in scrum), namely their **priorities and associated risks**
  - Users are typically involved in this stage

# Scrum Sprints



- Scrum sprints are **short development cycles** that typically last only a few weeks
- Each sprint starts with an evaluation of the tasks (called **product backlog** in scrum), namely their **priorities and associated risks**
  - Users are typically involved in this stage
- The product backlog is typically tracked using **issues**, which are **well-defined tasks that are assigned to a person**
  - Issues can have multiple statuses, such as **open, in-progress** and **completed**

# Scrum Sprints



- Scrum sprints are **short development cycles** that typically last only a few weeks
- Each sprint starts with an evaluation of the tasks (called **product backlog** in scrum), namely their **priorities and associated risks**
  - Users are typically involved in this stage
- The product backlog is typically tracked using **issues**, which are **well-defined tasks that are assigned to a person**
  - Issues can have multiple statuses, such as **open, in-progress** and **completed**
- The scrum master, who leads the stand-up meetings during a sprint, **is a member of the team** or **works closely with the team**
  - A scrum master is not an explicit team manager, but needs to have enough expertise to oversee the sprint and resolve challenges during the implementation

# Paradigm Challenges and Implications for Robotics

# Challenges With Plan-Driven Development

Plan-driven methods follow strict and often rigid procedures; this reduces the ability to adapt to requirement changes

# Challenges With Plan-Driven Development

Plan-driven methods follow strict and often rigid procedures; this reduces the ability to adapt to requirement changes

The focus on extensive documentation can slow down the development progress and can be counterproductive for smaller teams

# Challenges With Plan-Driven Development

Plan-driven methods follow strict and often rigid procedures; this reduces the ability to adapt to requirement changes

The focus on extensive documentation can slow down the development progress and can be counterproductive for smaller teams

Careful and extensive management is usually needed for executing plan-driven methods, which can affect the flexibility of the development team

# Challenges With Reuse-Oriented Development

The reuse of existing components may make it difficult or impossible to satisfy certain requirements

# Challenges With Reuse-Oriented Development

The reuse of existing components may make it difficult or impossible to satisfy certain requirements

If proprietary components are reused, there is no way to make relevant adaptations to the components

# Challenges With Reuse-Oriented Development

The reuse of existing components may make it difficult or impossible to satisfy certain requirements

If proprietary components are reused, there is no way to make relevant adaptations to the components

Existing components may impose development restrictions that lead to suboptimal software

# Challenges With Incremental Development

Careful planning is often essential for complex, safety-critical, and expensive systems; this sometimes collides with the principles of incremental methods

# Challenges With Incremental Development

Careful planning is often essential for complex, safety-critical, and expensive systems; this sometimes collides with the principles of incremental methods

Incremental development can go against established practices in larger teams and may not work within some contexts as a result

# Challenges With Incremental Development

Careful planning is often essential for complex, safety-critical, and expensive systems; this sometimes collides with the principles of incremental methods

Incremental development can go against established practices in larger teams and may not work within some contexts as a result

Due to their rapid development and deployment nature, incremental techniques, such as scrum, require a fully dedicated team that can handle the pressure of the process

# Challenges With Incremental Development

Careful planning is often essential for complex, safety-critical, and expensive systems; this sometimes collides with the principles of incremental methods

Incremental development can go against established practices in larger teams and may not work within some contexts as a result

Due to their rapid development and deployment nature, incremental techniques, such as scrum, require a fully dedicated team that can handle the pressure of the process

Daily stand-up meetings may be counterproductive in certain cases — some tasks require longer effort, and daily meetings distract rather than help

# Implications for Robotics

- In robotics, there is often a **discrepancy between user expectations and the reality of robots**
  - The best way to verify the acceptance of a robot is to **regularly interact with potential users** — incremental development methods are particularly suitable for this
  - It does not help to carefully plan the development of a robot that, in the end, will not be used by anyone

# Implications for Robotics

- In robotics, there is often a **discrepancy between user expectations and the reality of robots**
  - The best way to verify the acceptance of a robot is to **regularly interact with potential users** — incremental development methods are particularly suitable for this
  - It does not help to carefully plan the development of a robot that, in the end, will not be used by anyone
- Significant development in robotics is done through **collaborative (research) projects**; in the context of such projects, **plan-driven development is typically the method of choice** for overall project management
  - Careful planning is required for project proposals, such that the success of projects is measured not only by the results, but also by how well the progress corresponds to the promised plan
  - The expectation to produce detailed deliverables is also a common feature of such projects
  - Elements of iterative methods are, however, often adopted throughout the development

# Implications for Robotics

- In robotics, there is often a **discrepancy between user expectations and the reality of robots**
  - The best way to verify the acceptance of a robot is to **regularly interact with potential users** — incremental development methods are particularly suitable for this
  - It does not help to carefully plan the development of a robot that, in the end, will not be used by anyone

- Significant development in robotics is done through **collaborative (research) projects**; in the context of such projects, **plan-driven development is typically the method of choice** for overall project management
  - Careful planning is required for project proposals, such that the success of projects is measured not only by the results, but also by how well the progress corresponds to the promised plan
  - The expectation to produce detailed deliverables is also a common feature of such projects
  - Elements of iterative methods are, however, often adopted throughout the development

- **Incremental development is essential in the context of robotics competitions** — competition rules change frequently, so teams need to be able to adapt to those changes quickly

# Summary

- A software engineering methodology defines procedures for managing the complete workflow of a software project

- There are different types of software methodologies, which can be generally observed as plan-driven or incremental

- Reuse-oriented development is a methodology that is explicitly concerned with identifying and integrating existing components into the developed software

- Agile development methods, such as extreme programming and scrum, are popular examples of incremental development that emphasise change management and user involvement in the development process

- All of the methodologies have their own pros and cons with respect to robot software development; there is no one-size-fits-all methodology that works equally well for all cases