Introduction

ENGI 5895: Software Design

Andrew Vardy

Faculty of Engineering & Applied Science
Memorial University of Newfoundland

January 4, 2018

What is this course about?

Many people can program, but are they Software Engineers? A Software Engineer is someone who can do the following:

- Ensure the correctness of their software
- Develop efficient solutions
- Design systems which are flexible, reusable, and maintainable
- Communicate the design and behaviour of a software system

This course focusses on the last two points, but we will always be mindful of correctness and efficiency.

What is this course about?

Many people can program, but are they Software Engineers? A Software Engineer is someone who can do the following:

- Ensure the **correctness** of their software
- Develop **efficient** solutions
- Design systems which are **flexible**, **reusable**, and **maintainable**
- **Communicate** the design and behaviour of a software system

This course focusses on the last two points, but we will always be mindful of correctness and efficiency.

What is this course about?

Many people can program, but are they Software Engineers? A Software Engineer is someone who can do the following:

- Ensure the **correctness** of their software
- Develop **efficient** solutions
- Design systems which are **flexible**, **reusable**, and **maintainable**
- **Communicate** the design and behaviour of a software system

This course focusses on the last two points, but we will always be mindful of correctness and efficiency.

What is this course about?

Many people can program, but are they Software Engineers? A Software Engineer is someone who can do the following:

- Ensure the **correctness** of their software
- Develop **efficient** solutions
- Design systems which are **flexible**, **reusable**, and **maintainable**
- **Communicate** the design and behaviour of a software system

This course focusses on the last two points, but we will always be mindful of correctness and efficiency.

What is this course about?

Many people can program, but are they Software Engineers? A Software Engineer is someone who can do the following:

- Ensure the **correctness** of their software
- Develop **efficient** solutions
- Design systems which are **flexible**, **reusable**, and **maintainable**
- **Communicate** the design and behaviour of a software system

This course focusses on the last two points, but we will always be mindful of correctness and efficiency.

What is this course about?

Many people can program, but are they Software Engineers? A Software Engineer is someone who can do the following:

- Ensure the **correctness** of their software
- Develop **efficient** solutions
- Design systems which are **flexible**, **reusable**, and **maintainable**
- **Communicate** the design and behaviour of a software system

This course focusses on the last two points, but we will always be mindful of correctness and efficiency.

What is this course about?

Many people can program, but are they Software Engineers? A Software Engineer is someone who can do the following:

- Ensure the **correctness** of their software
- Develop **efficient** solutions
- Design systems which are **flexible**, **reusable**, and **maintainable**
- **Communicate** the design and behaviour of a software system

This course focusses on the last two points, but we will always be mindful of correctness and efficiency.

Flexibility, Reusability, and Maintainability

You have developed software that behaves correctly and efficiently in scenario A.

- **Flexibility**: Your boss adds scenario B. How much effort does it take to make it work?
- **Reusability**: You realize that part of your code might actually be useful in scenarios X and Y. How much effort does it take to isolate the parts that you need?
- **Maintainable**: Over time features are added and bugs are corrected. How much effort does it take to make these changes and to continue making similar changes in the long run?

We will discuss object-oriented design principles and patterns to address these concerns.

Flexibility, Reusability, and Maintainability

You have developed software that behaves correctly and efficiently in scenario A.

- **Flexibility**: Your boss adds scenario B. How much effort does it take to make it work?
- **Reusability**: You realize that part of your code might actually be useful in scenarios X and Y. How much effort does it take to isolate the parts that you need?
- **Maintainable**: Over time features are added and bugs are corrected. How much effort does it take to make these changes and to continue making similar changes in the long-run?

We will discuss object-oriented design principles and patterns to address these concerns.

Flexibility, Reusability, and Maintainability

You have developed software that behaves correctly and efficiently in scenario A.

- **Flexibility**: Your boss adds scenario B. How much effort does it take to make it work?
- **Reusability**: You realize that part of your code might actually be useful in scenarios X and Y. How much effort does it take to isolate the parts that you need?
- **Maintainable**: Over time features are added and bugs are corrected. How much effort does it take to make these changes and to continue making similar changes in the long-run?

We will discuss object-oriented design principles and patterns to address these concerns.

Flexibility, Reusability, and Maintainability

You have developed software that behaves correctly and efficiently in scenario A.

- **Flexibility**: Your boss adds scenario B. How much effort does it take to make it work?
- **Reusability**: You realize that part of your code might actually be useful in scenarios X and Y. How much effort does it take to isolate the parts that you need?
- **Maintainable**: Over time features are added and bugs are corrected. How much effort does it take to make these changes and to continue making similar changes in the long-run?

We will discuss object-oriented design principles and patterns to address these concerns.

Flexibility, Reusability, and Maintainability

You have developed software that behaves correctly and efficiently in scenario A.

- **Flexibility**: Your boss adds scenario B. How much effort does it take to make it work?
- **Reusability**: You realize that part of your code might actually be useful in scenarios X and Y. How much effort does it take to isolate the parts that you need?
- **Maintainable**: Over time features are added and bugs are corrected. How much effort does it take to make these changes and to continue making similar changes in the long-run?

We will discuss object-oriented design principles and patterns to address these concerns.

Flexibility, Reusability, and Maintainability

You have developed software that behaves correctly and efficiently in scenario A.

- **Flexibility**: Your boss adds scenario B. How much effort does it take to make it work?
- **Reusability**: You realize that part of your code might actually be useful in scenarios X and Y. How much effort does it take to isolate the parts that you need?
- **Maintainable**: Over time features are added and bugs are corrected. How much effort does it take to make these changes and to continue making similar changes in the long-run?

We will discuss object-oriented design principles and patterns to address these concerns.

Communication

This course is also about communication. Software gets very complicated very quickly! How can you communicate the essential ideas behind your design...

- to yourself, when you look at your code two years from now
- to your peers
- to your boss, to explain the difficulty of the problem you are addressing

In this course you will learn to describe your designs using the Unified Modelling Language (UML).

Communication

This course is also about communication. Software gets very complicated very quickly! How can you communicate the essential ideas behind your design...

- to yourself, when you look at your code two years from now
  - You pepper your code with comments, but what's the big picture?
- to your peers
  - How can they use, modify, or extend your code? Isn't it so hard to explain that you might as well make the changes yourself?
- to your boss, to explain the difficulty of the problem you are addressing
  - "What do you mean you need more time? Can't you just make it work?"

In this course you will learn to describe your designs using the Unified Modelling Language (UML).

Communication

This course is also about communication. Software gets very complicated very quickly! How can you communicate the essential ideas behind your design...

- to yourself, when you look at your code two years from now
  - You pepper your code with comments, but what's the big picture?
- to your peers
  - How can they use, modify, or extend your code? Is it so hard to explain that you might as well make the changes yourself?
- to your boss, to explain the difficulty of the problem you are addressing
  - "What do you mean you need more time? Can't you just make it work?"

In this course you will learn to describe your designs using the Unified Modelling Language (UML).

Communication

This course is also about communication. Software gets very complicated very quickly! How can you communicate the essential ideas behind your design...

- to yourself, when you look at your code two years from now
  - You pepper your code with comments, but what's the big picture?
- to your peers
  - How can they use, modify, or extend your code? Is it so hard to explain that you might as well make the changes yourself?
- to your boss, to explain the difficulty of the problem you are addressing
  - "What do you mean you need more time? Can't you just make it work?"

In this course you will learn to describe your designs using the Unified Modelling Language (UML).

Communication

This course is also about communication. Software gets very complicated very quickly! How can you communicate the essential ideas behind your design...

- to yourself, when you look at your code two years from now
  - You pepper your code with comments, but what's the big picture?
- to your peers
  - How can they use, modify, or extend your code? Is it so hard to explain that you might as well make the changes yourself?
- to your boss, to explain the difficulty of the problem you are addressing
  - "What do you mean you need more time? Can't you just make it work?"

In this course you will learn to describe your designs using the Unified Modelling Language (UML).

Communication

This course is also about communication. Software gets very complicated very quickly! How can you communicate the essential ideas behind your design...

- to yourself, when you look at your code two years from now
  - You pepper your code with comments, but what's the big picture?
- to your peers
  - How can they use, modify, or extend your code? Is it so hard to explain that you might as well make the changes yourself?
- to your boss, to explain the difficulty of the problem you are addressing
  - "What do you mean you need more time? Can't you just make it work?"

In this course you will learn to describe your designs using the Unified Modelling Language (UML).

Communication

This course is also about communication. Software gets very complicated very quickly! How can you communicate the essential ideas behind your design...

- to yourself, when you look at your code two years from now
  - You pepper your code with comments, but what's the big picture?
- to your peers
  - How can they use, modify, or extend your code? Is it so hard to explain that you might as well make the changes yourself?
- to your boss, to explain the difficulty of the problem you are addressing
  - "What do you mean you need more time? Can't you just make it work?"

In this course you will learn to describe your designs using the Unified Modelling Language (UML).

Communication

This course is also about communication. Software gets very complicated very quickly! How can you communicate the essential ideas behind your design...

- to yourself, when you look at your code two years from now
  - You pepper your code with comments, but what's the big picture?
- to your peers
  - How can they use, modify, or extend your code? Is it so hard to explain that you might as well make the changes yourself?
- to your boss, to explain the difficulty of the problem you are addressing
  - "What do you mean you need more time? Can't you just make it work?"

In this course you will learn to describe your designs using the Unified Modelling Language (UML).

Communication

This course is also about communication. Software gets very complicated very quickly! How can you communicate the essential ideas behind your design...

- to yourself, when you look at your code two years from now
  - You pepper your code with comments, but what's the big picture?
- to your peers
  - How can they use, modify, or extend your code? Is it so hard to explain that you might as well make the changes yourself?
- to your boss, to explain the difficulty of the problem you are addressing
  - "What do you mean you need more time? Can't you just make it work?"

In this course you will learn to describe your designs using the Unified Modelling Language (UML).

Deliverables

- Two assignments (15%):
  - Complete individually or in pairs
  - Assign. 1: Implement a given design
  - Assign. 2: Design a system
- Labs (0%)
  - Familiarization with Java IDE (Eclipse) and Visual Paradigm CASE tool
- Project (60%):
  - Teams of 2-3
  - Choose your own stand-alone software system to implement (e.g. game, simulation, application, ...)
  - Project includes both design and implementation
- Two mid-term exams (25%)

Deliverables

- Two assignments (15%):
  - Complete individually or in pairs
  - Assign. 1: Implement a given design
  - Assign. 2: Design a system
- Labs (0%)
  - Familiarization with Java IDE (Eclipse) and Visual Paradigm CASE tool
- Project (60%):
  - Teams of 2-3
  - Choose your own stand-alone software system to implement (e.g. game, simulation, application, ...)
  - Project includes both design and implementation
- Two mid-term exams (25%)

Deliverables

- Two assignments (15%):
  - Complete individually or in pairs
  - Assign. 1: Implement a given design
  - Assign. 2: Design a system
- Labs (0%)
  - Familiarization with Java IDE (Eclipse) and Visual Paradigm CASE tool
- Project (60%):
  - Teams of 2-3
  - Choose your own stand-alone software system to implement (e.g. games, simulation, application, ...)
  - Project includes both design and implementation
- Two mid-term exams (25%)

Deliverables

- Two assignments (15%):
  - Complete individually or in pairs
  - Assign. 1: Implement a given design
  - Assign. 2: Design a system
- Labs (0%)
  - Familiarization with Java IDE (Eclipse) and Visual Paradigm CASE tool
- Project (60%):
  - Teams of 2-3
  - Choose your own stand-alone software system to implement (e.g. games, simulations, applications, ...)
  - Project includes both design and implementation
- Two mid-term exams (25%)

Deliverables

- Two assignments (15%):
  - Complete individually or in pairs
  - Assign. 1: Implement a given design
  - Assign. 2: Design a system
- Labs (0%)
  - Familiarization with Java IDE (Eclipse) and Visual Paradigm CASE tool
    - CASE = Computer-Aided Software Engineering
- Project (60%):
  - Teams of 2-3
  - Choose your own stand-alone software system to implement (e.g. games, simulations, applications, ...)
  - Project includes both design and implementation
- Two mid-term exams (25%)

Deliverables

- Two assignments (15%):
  - Complete individually or in pairs
  - Assign. 1: Implement a given design
  - Assign. 2: Design a system
- Labs (0%)
  - Familiarization with Java IDE (Eclipse) and Visual Paradigm CASE tool
    - CASE = Computer-Aided Software Engineering
- Project (60%):
  - Teams of 2-3
  - Choose your own stand-alone software system to implement (e.g. game, simulation, mobile app, ...)
  - Project includes both design and implementation
- Two mid-term exams (25%)

Deliverables

- Two assignments (15%):
  - Complete individually or in pairs
  - Assign. 1: Implement a given design
  - Assign. 2: Design a system
- Labs (0%)
  - Familiarization with Java IDE (Eclipse) and Visual Paradigm CASE tool
    - CASE = Computer-Aided Software Engineering
- Project (60%):
  - Teams of 2-3
  - Choose your own stand-alone software system to implement (e.g. games, simulations, problem solvers)
  - Project includes both design and implementation
- Two mid-term exams (25%)

Deliverables

- Two assignments (15%):
  - Complete individually or in pairs
  - Assign. 1: Implement a given design
  - Assign. 2: Design a system
- Labs (0%)
  - Familiarization with Java IDE (Eclipse) and Visual Paradigm CASE tool
    - CASE = Computer-Aided Software Engineering
- Project (60%):
  - Teams of 2-3
  - Choose your own stand-alone software system to implement (e.g. game, simulation, application,...)
  - Project includes both design and implementation
- Two mid-term exams (25%)

Deliverables

- Two assignments (15%):
  - Complete individually or in pairs
  - Assign. 1: Implement a given design
  - Assign. 2: Design a system
- Labs (0%)
  - Familiarization with Java IDE (Eclipse) and Visual Paradigm CASE tool
    - CASE = Computer-Aided Software Engineering
- Project (60%):
  - Teams of 2-3
  - Choose your own stand-alone software system to implement (e.g. game, simulation, application,...)
  - Project includes both design and implementation
- Two mid-term exams (25%)

Deliverables

- Two assignments (15%):
  - Complete individually or in pairs
  - Assign. 1: Implement a given design
  - Assign. 2: Design a system
- Labs (0%)
  - Familiarization with Java IDE (Eclipse) and Visual Paradigm CASE tool
    - CASE = Computer-Aided Software Engineering
- Project (60%):
  - Teams of 2-3
  - Choose your own stand-alone software system to implement (e.g. game, simulation, application,...)
  - Project includes both design and implementation
- Two mid-term exams (25%)

Deliverables

- Two assignments (15%):
  - Complete individually or in pairs
  - Assign. 1: Implement a given design
  - Assign. 2: Design a system
- Labs (0%)
  - Familiarization with Java IDE (Eclipse) and Visual Paradigm CASE tool
    - CASE = Computer-Aided Software Engineering
- Project (60%):
  - Teams of 2-3
  - Choose your own stand-alone software system to implement (e.g. game, simulation, application,...)
  - Project includes both design and implementation
- Two mid-term exams (25%)

Deliverables

- Two assignments (15%):
  - Complete individually or in pairs
  - Assign. 1: Implement a given design
  - Assign. 2: Design a system
- Labs (0%)
  - Familiarization with Java IDE (Eclipse) and Visual Paradigm CASE tool
    - CASE = Computer-Aided Software Engineering
- Project (60%):
  - Teams of 2-3
  - Choose your own stand-alone software system to implement (e.g. game, simulation, application,...)
  - Project includes both design and implementation
- Two mid-term exams (25%)

Communicating your design

- You will document your designs in carefully written reports that include helpful UML diagrams
- In the project, there will be design review meetings where you will present your design and explain its rationale and history
- **It is your responsibility to highlight and explain the principles and patterns utilized in your design!**
- You will demo your working programs and submit clean and carefully documented source code

Communicating your design

- You will document your designs in carefully written reports that include helpful UML diagrams
- In the project, there will be design review meetings where you will present your design and explain its rationale and history
- It is your responsibility to highlight and explain the principles and patterns utilized in your design!
- You will demo your working programs and submit clean and carefully documented source code

Communicating your design

- You will document your designs in carefully written reports that include helpful UML diagrams
- In the project, there will be design review meetings where you will present your design and explain its rationale and history
- **It is your responsibility to highlight and explain the principles and patterns utilized in your design!**
- You will demo your working programs and submit clean and carefully documented source code

Communicating your design

- You will document your designs in carefully written reports that include helpful UML diagrams
- In the project, there will be design review meetings where you will present your design and explain its rationale and history
- **It is your responsibility to highlight and explain the principles and patterns utilized in your design!**
- You will demo your working programs and submit clean and carefully documented source code

Course outline

- Introduction
- The Unified Modelling Language (UML)
  - Class and sequence diagrams; use cases
- Brief introduction to Java
- Development Processes
- Design Principles
  - e.g. The Single Responsibility Principle: "A class should have only one reason to change".
- Design Patterns
  - e.g. Iterator
- Various bits of technology needed for the project:
  - GUI, graphics, networking, etc.

Course outline

- Introduction
- The Unified Modelling Language (UML)
  - Class and sequence diagrams; use cases
- Brief introduction to Java
- Development Processes
- Design Principles
  - e.g. The SingleResponsibility Principle: "A class should have only one reason to change"
- Design Patterns
  - e.g. Iterator
- Various bits of technology needed for the project:
  - GUI, graphics, networking, etc.

Course outline

- Introduction
- The Unified Modelling Language (UML)
  - Class and sequence diagrams; use cases
- Brief introduction to Java
- Development Processes
- Design Principles
  - e.g. The SingleResponsibility Principle: "A class should have only one reason to change"
- Design Patterns
  - e.g. Iterator
- Various bits of technology needed for the project:
  - GUI, graphics, networking, etc.

Course outline

- Introduction
- The Unified Modelling Language (UML)
  - Class and sequence diagrams; use cases
- Brief introduction to Java
- Development Processes
- Design Principles
  - e.g. The Single Responsibility Principle: "A class should have only one reason to change"
- Design Patterns
  - e.g. Iterator
- Various bits of technology needed for the project:
  - GUI, graphics, networking, etc.

Course outline

- Introduction
- The Unified Modelling Language (UML)
  - Class and sequence diagrams; use cases
- Brief introduction to Java
- Development Processes
- Design Principles
  - e.g. The Single-Responsibility Principle: "A class should have only one reason to change"
- Design Patterns
  - e.g. Iterator
- Various bits of technology needed for the project:
  - GUI, graphics, networking, etc.

Course outline

- Introduction
- The Unified Modelling Language (UML)
  - Class and sequence diagrams; use cases
- Brief introduction to Java
- Development Processes
- Design Principles
  - e.g. The Single-Responsibility Principle: "A class should have only one reason to change"
- Design Patterns
  - e.g. Iterator
- Various bits of technology needed for the project:
  - GUI, graphics, networking, etc.

Course outline

- Introduction
- The Unified Modelling Language (UML)
  - Class and sequence diagrams; use cases
- Brief introduction to Java
- Development Processes
- Design Principles
  - e.g. The Single-Responsibility Principle: "A class should have only one reason to change"
- Design Patterns
  - e.g. Iterator
- Various bits of technology needed for the project:
  - GUI, graphics, networking, etc.

Course outline

- Introduction
- The Unified Modelling Language (UML)
  - Class and sequence diagrams; use cases
- Brief introduction to Java
- Development Processes
- Design Principles
  - e.g. The Single-Responsibility Principle: "A class should have only one reason to change"
- Design Patterns
  - e.g. Iterator
- Various bits of technology needed for the project:
  - GUI, graphics, networking, etc.

Course outline

- Introduction
- The Unified Modelling Language (UML)
  - Class and sequence diagrams; use cases
- Brief introduction to Java
- Development Processes
- Design Principles
  - e.g. The Single-Responsibility Principle: "A class should have only one reason to change"
- Design Patterns
  - e.g. Iterator
- Various bits of technology needed for the project:
  - GUI, graphics, networking, etc...

Course outline

- Introduction
- The Unified Modelling Language (UML)
  - Class and sequence diagrams; use cases
- Brief introduction to Java
- Development Processes
- Design Principles
  - e.g. The Single-Responsibility Principle: "A class should have only one reason to change"
- Design Patterns
  - e.g. Iterator
- Various bits of technology needed for the project:
  - GUI, graphics, networking, etc...

Course outline

- Introduction
- The Unified Modelling Language (UML)
  - Class and sequence diagrams; use cases
- Brief introduction to Java
- Development Processes
- Design Principles
  - e.g. The Single-Responsibility Principle: "A class should have only one reason to change"
- Design Patterns
  - e.g. Iterator
- Various bits of technology needed for the project:
  - GUI, graphics, networking, etc...