Manual Software Architecture

Term 2.2

Course Software Architecture study year 2022/2023

Bachelor Creative Media and Game Technologies (CMGT)
School of Creative Technology

Publication date	November 2022
Version	1.0
Module coordinator	Yiwei Jiang (YJI01),
Lecturers	Yiwei Jiang (YJI01), Hans Wichman (HWI05)
CMGT roles	Engineer

1 General overview

Module Name	Software Architecture
Unit code	T.51701
Year and Term	2.2
CMGT roles	Engineer
Credits	3 ECTS
Lessons	5* 4 = 20 hours
Study load	84 hours
Responsible lecturers	Yiwei Jiang, Hans Wichman
Lesson structure	1.5 hours lecture, 2.5 hours lab
Module summary	The students learn how to use UML to visualize the design of their code, apply design patterns to improve the project's structure, and do unit test.
Industry relevance	In real industry, a programmer usually works in a team and on long running projects, good software architecture helps to work better with fellow programmers and avoid unnecessary waste of precious time.
Type of exam	Product
Exam code	T.51701
CMGT Competencies	1. Technical research and analysis 2. Designing, prototyping and realizing 3. Testing and rolling out 9. Working in a project-based way 12. Responsibility
Required prior knowledge and skills / conditions for enrolment	Programming, Game Programming, Physics Programming, Unity Game Scripting, Algorithms.
Preparatory for:	2nd year, 3rd project

2 Why this module?

This module is in term 2.2, at this point the students have learned much theoretical knowledge of programming from previous modules and gained much programming experience from the group projects. The students should have the knowledge and skills to make the code function as required.

However, functionality is only one aspect of programming, there are many other aspects that are crucial to a successful project: flexibility, reusability, readability, error tolerance, etc. To get fully prepared for the real projects in the industry, students have to consider all these during the software design process, and this is where software architecture comes in.

In this module the students will learn how to use UML to visualize the code structure, apply design patterns to improve their code quality, use unit tests to ensure functions work correctly and how to do proper code reviews with peers.

2.1 What happens in the labs and lectures?

In lectures the students will learn software architecture theories.

In lab classes the students will follow agile process to make different iterations for the assignment. In each lab the student will refactor the previous version by applying the theories from the lectures.

2.2 How does this module relate to other modules in the CMGT study programme?

This module will help the students to refactor their projects from other engineering modules and prepare the students for the up-coming projects, and other large-scale projects for their minors.

3 What are you going to learn in this module (learning objectives)?

The student:

1. knows how to use UML to visualize the structure of their code. (20%)
2. shows how to apply design patterns in a project. (60%)
3. shows how to do unit tests for their code. (10%)
4. knows how to do code reviews. (10%)

4 Which resources do you need?

Tools:
Unity(personal license)
Websites:
Design patterns:
http://gameprogrammingpatterns.com/contents.html
https://refactoring.guru/design-patterns
UML :
Basic:
https://people.cs.pitt.edu/~chang/153/UML/UML.htm
Advanced:
http://uml.org/
Unit test :
http://www.nunit.org/
https://docs.unity3d.com/Packages/com.unity.test-framework@1.1/manual/index.html

5 What does the programme of this module look like?

On Blackboard you'll find the course content and a detailed course overview.

week	Lecture/Lab	Topic(s)
2.4	Lecture	Go over the manual and assignment. Composition and interface, UML.
2.4	Lab + homework	Analyse the assignment's requirements, play existing TD games, or check existing tutorials for inspiration (no copying tutorials allowed), make a working prototype with basic functionalities.
2.5	Lecture	Design patterns.
2.5	Lab + homework	Refactor the code, apply at least 1 design pattern to reduce coupling and improve reusability and extendibility.
2.6	Lecture	Scriptable objects and unit tests.
2.6	Lab + homework	Do a code review with your peers for the current progress. Apply additional patterns or use scriptable objects to refactor the code. Design unit tests for at least 2 main functions, e.g., attack detection, enemy spawning, etc.
2.7	Lecture	Design patterns and performance.
2.7	Lab + homework	If applicable, optimize your project for better performance. Build the beta version of the assignment, this version should meet all the must-have requirements in the pre-conditions of the rubrics. Draw a UML class diagram based on your final code structure. Optionally, design unit tests for more functions.
2.8	Lecture	Design patterns and memory profiling.
2.8	Lab + homework	Testing and profiling your project, build the final version, have the lab teacher review the assignment.

6 How is this module assessed?

6.1 Assignment

A tower defense game that meets the following requirements:

Wave requirements

- The game has at least 5 waves of enemies, each wave is more difficult than the previous one.
- Properties of a wave can be configured in the Unity Editor without changing the code, e.g. one or multiple of: enemy types, enemy amounts, enemy combinations, delay between enemies, percentage of each enemy type, chance of spawning, etc.
- In between waves the players have a short building phase to sell/destroy, build and upgrade the towers.

Enemy requirements

- Enemies are spawned from spawn point(s)

- Enemies follow a path, or multiple paths to the end goal
- When x enemies reach the end goal it is game over (tweakable level property)
- Enemies have at least three properties (health, speed and carried money)
- There are at least two types of different enemies
- When an enemy unit is destroyed, player gets the cash
- Attributes and visuals of enemies can be set up in the Unity Editor without hard coding.

Tower requirements

- Towers can be built on grid cell along the enemy path
- One tower-gridcell can only hold one tower (towers can't overlap with each other).
- It should be clear to the player what towers can be bought with the current amount of money.
- There are at least three different types of towers:
  * Single target attack tower
  * AOE attack tower
  * Debuff attack tower (slows downs enemies, or another effect, the same effect never stacks)
- Towers can be upgraded with money; the upgrade system can be adjusted in the Unity Editor without changing the code.
- It should be clear which towers can be upgraded with the current amount of money

GUI requirements:
The following information should be displayed in clear ways in the game:

- Wave number.
- Total money.
- Time left for the player to build/upgrade towers before next wave begins.
- Where can the towers be built.
- Cost for building/upgrading each tower.
- Available and unavailable towers and upgrades.
- Health bar of each enemy.
- Amount of money dropped by each enemy when killed.
- How many enemies can enter the end point before game over.
- Game over.
- Player wins the game.

You can use free visual and audio assets, but all scripts must be made by yourself.

6.2 Deliverables

- Functional assignment that meets all the requirements in 6.1.
- A UML class diagram to visualize the final code structure.
- A report explaining how you apply the design patterns and principles discussed in the lectures, and the overview of your design decisions.
- An optional code review report.

6.3 Deadlines:

First attempt: 11:59 pm, February 12, 2023.
Resit: 11:59 pm, April 09, 2023.

6.4 Procedure

The student should make sure to hand in all the required files before the deadline. If further assessment is needed a teacher will send you an email to make an appointment with you.

6.5 Criteria & Assessment form

The student must meet all the preconditions.
See the rubrics in section 8.

6.6 Resit

The second opportunity will be in week 3.10, the deadline is 11:59 pm, Sunday of week 3.10, 2023.

7 Who are the contact persons for this module?

Yiwei Jiang y.jiang@saxion.nl
Hans Wichman j.c.wichman@saxion.nl

8 Rubrics

Note: the rubrics are used to determine your grade and are visible in Blackboard under 'Grades and Feedback' → 'View rubrics'.

	Insufficient	Sufficient	Good
Preconditions: • The student uploaded the assignment that meets the requirements from 6.1 in the manual. • The student uploaded a UML class diagram to visualize the code structure. • The student uploaded a report explaining how they apply the design patterns and principles to fulfil the must-have requirements. If you don't meet the preconditions the assessment will be assessed with a 1.			

UML The student used a UML class diagram to visualize the code structure. (20%)	-50% The student didn't use any UML class diagram to visualize the code structure.	12% The student used a UML class diagram to visualize the code structure, but certain parts of the diagram are lacking or unclear.	20% The diagram clearly visualizes the code structure without additional explanation.
Code quality and structure The student designed high quality code structure which increases the code's reusability and extensibility. (20%)	-50% One or more of the following: The project has no clear structure. The code has too much unnecessary coupling, or the code has no reusability or extensibility. Unclear variable, method names or layout. Bad use of access modifiers. Bad or no code conventions. The report doesn't document the design decisions.	12% Clear project structure. Good code quality. The report documents the design decisions.	20% Sufficient+: At least one abstract sub-system (e.g., pathfinding system, node graph system, state machine, tween system, achievement system, GUI window management etc.) was made into a substantial library or framework that can be used in developing new similar projects with little modification. The report justifies the design decisions of the code structure.
Design Patterns The student applied at least 2 of the following design patterns to decouple different systems in the model and separate the responsibilities of the objects: **singleton, service locator, command, FSM, observer, event bus, strategy.** (40%)	-50% The student didn't correctly apply at least 2 design patterns or didn't provide an explanation on the implementation of the patterns.	24% The student correctly applied at least 2 design patterns. The report documents the implementation of the patterns.	40% Sufficient+: At least 1 additional design pattern is implemented into the project, and scriptable objects are used in the project in a sensible way.
Unit Test The student used unit test to validate that the key functions were working properly and	1% The student didn't do unit test in the code, or the unit	6% The student did unit test to ensure that the key functions worked correctly.	10% Sufficient+:

increase the efficiency of debugging. (10%)	tests didn't cover any key functions.		The unit tests cover necessary exception cases for applicable functions.
Code Review The student conducted code reviews with peers and reflected and/or implemented the feedback gathered from code review sessions. (10%)	1% The student didn't do code reviews or didn't hand in a code review report which documented: reviewed code pieces, feedback from peers.	6% The student did code reviews and handed in a code review report which documented: reviewed code pieces, feedback from peers.	10% Sufficient+: The student explained in the report how the code review feedback improved the code quality, or, if all feedback was rejected, clearly explained why the code quality was better without implementing any feedback.