Software Engineering Project: **Technopoly**

Deliver a well-designed and well-documented working system that satisfies the customer's requirements.

The emphasis in this project is on the process of requirements analysis, system design, software implementation and system testing that delivers reliable and appropriate functionality. The project will demonstrate your understanding of, and ability to put into practice, object-oriented software engineering principles, and your ability to work in a software engineering team. Your requirements analysis and system design will be represented in the graphical notation known as the Unified Modelling Language (UML). This will be a 'use case driven' development process, in which each use case describes "a set of sequences of actions, including variants, that a system performs to yield an observable result of value to an actor" (Booch, Rumbaugh and Jacobson).

The system to be developed is a Monopoly-type game – but with some important differences! It does not have an elaborate graphical user interface. Instead the game is to be played via the console of the development environment. It will also have its own distinctive theme, drawn from the world of computing and informatics! Instead of buying and developing properties and charging rent as you do in the traditional game, think of taking over different fields of computer technology, developing products in different areas, and generating resources from them or devoting resources to them. Rather than use cash for your transactions, you might want to think about allocating 'human resources' – investing in an area from your own pool of talent, and acquiring new talent from other companies. It's your job to think through and apply the 'metaphor' you want to use in your game.

The simple console-based interface gives you the opportunity to concentrate on the process of determining and designing the underlying object-oriented system, rather than focus on visual or audio effects. The system uses English to convey the state of the game and to ask its players what they want to do next. Though you don't have to develop a speech user interface (SUI), imagine a game where the state of play can be conveyed in words alone – whether a new development is being reported or the current state of play summarised.

A game of this kind might start and unfold in the following manner (the example is not taken from Technopoly – it's more like a very restricted version of traditional Monopoly – though the behaviour of your game will be broadly similar):

```
What is the first player's name? Janet
What is the second player's name? John

Janet, would you like to roll? y/n: y

Janet, you've rolled a 5.
Janet, you've landed on Pentonville Road
Janet, do you want to buy this property (y/n) ? y

Property bought. Your old kitty was £1500.
Your new kitty is £1440.

Square Owned by Janet: Pentonville Road

John, would you like to roll? y/n: […]
```

Because the game is to be conducted in natural language only (i.e. English phrases that convey the state of play, etc.), it will have fewer 'squares' than a conventional Monopoly game. A separate guide is required (you may create this with any suitable drawing tool, such as PowerPoint or the drawing tool of Word, and should include this in the *Requirements Analysis* section of the *Short Printed Report* [see below]) – you should bear in mind that, in a full realisation of the game, the guide *could* [you don't have to do this!] be converted into a non-visual equivalent, with, for example, embossed lines and Braille captions.

This game guide will be very simple, along the following lines. Again, the example shown is NOT for Technopoly, nor is it necessarily complete; it is intended only to represent the manner in which such a graphical representation might be drawn. Remember that, through the comments it writes to the console, the *software* itself must remind players of their positions, and their ownership of squares and the properties of those squares.

Investment Fund Collect $$X_I$	Metlife Colour Brown Site costs $$X_M$ Office costs $$Y_M$ Centre costs $$Z_M$	Seagram Colour Brown Site costs $$X_S$ Office costs $$Y_S$ Centre costs $$Z_S$	Empire Colour Blue Site costs $$X_E$ Office costs $$Y_E$ Centre costs $$Z_E$	Chrysler Colour Blue Site costs $$X_C$ Office costs $$Y_C$ Centre costs $$Z_C$	Rockefeller Colour Blue Site costs $$X_R$ Office costs $$Y_R$ Centre costs $$Z_R$

Your game will have many of the features that players of the original Monopoly would expect, but in a much simpler form. Within the constraints of the ‘natural language interface’, your customer’s core requirements are set out below, between the dashed lines (■■■■■■)] .

Remember, this is what your customer is asking for and expects to be delivered. You can realize these requirements while giving them your own creative ‘twist’ – but your customer is not asking for additional features (e.g. the customer does not want more squares, more types of squares, or more players than the number stipulated below). On the other hand, where checks are needed to ensure that the game is usable (for example, to avoid a situation where two players have exactly the same name), then such checks should be implemented, even if they are not explicitly requested. That is simply good design.

Similarly, although your customer has no *immediate* plans to ‘adapt’ or ‘upgrade’ Technopoly to a more specialised or more complex game, there may be (OO) design features that, with only a little additional effort, you can incorporate ‘behind the scenes’ to make your system more *maintainable* and *extensible* – e.g. a well-designed game would allow the number of squares or the maximum number of players to be increased or reduced easily in the code, should that requirement arise in future. Good software design not only meets present requirements but can easily accommodate change. [N.B. A separate user interface for major re-configuration of the game is NOT one of the requirements for Technopoly.]

Here are the core requirements…

■■

The game has up to four players, and their names should be entered.

The players take turns.

They throw 2 virtual dice.

There is a start square, where players pick up their ‘resources’ (it’s your choice what the ‘resources’ represent – whether people or money, for example – and you should be inventive with the name of the square – this is the equivalent of standard Monopoly’s ‘Collect £/$/€200 salary as you pass Go’ square).

There is an equivalent of a ‘Free Parking’ square – again, you decide what it is called in your game.

There are four ‘fields’, two consisting of three ‘areas’ and two consisting of two ‘areas’.

[A number of related areas form a ‘field’ in Technopoly. You decide what the fields are called and what they represent. For example, Artificial Intelligence (whether as an academic discipline or an area of product development) might be a ‘field’. ‘Fields’ are equivalent to the colour groups in Monopoly. Different areas make up a field: Dialogue Systems (again, whether it represents a research area or families of commercial products) might be an area in the field of Artificial Intelligence (say it out loud, and see if it makes sense!) You decide what the technology-related fields are in your game, and what areas they will include.]

One of the two-area fields is the most expensive field on the board to acquire and resource; another two-area field is the least expensive field to acquire and resource.

Before you can develop an area within a field, you must own/manage/'be in charge of' (you decide what 'ownership' means!) the *whole* field – and on your turn you can develop an area in a field that you already own even if you are not positioned on that area.

You develop a field by building the equivalent of a 'house' on it: you decide what a 'house' is called, what it represents and how much it costs in your game [it might be something in the physical world or something from the 'knowledge economy'; others might have to pay money to use it, or provide people to help develop it further. Again, you decide the nature and the significance of the transactions between players.]

Three 'houses' are needed before you can…

establish (and pay for, or otherwise 'resource') the equivalent of an 'hotel' [again you decide what this represents and what it costs].

Players taking a turn are told where they have landed and what their obligations or opportunities are.

Where appropriate, they may indicate their choice of action.

If a player's resources have changed, the system indicates the reason for the change and announces the player's new 'balance' (e.g. the 'funds' or 'people' that are still available).

Not only is there a cost associated with developing areas within fields: when you land on an area, but do not 'own' it yourself, you have to give up some of your resources for it – the more developed the area, the greater the resource required.

When one player runs out of resources, the player with most resources is declared the winner. If one player no longer wants to play, the game ends. In both cases, the amount of resource each player holds is shown. There is no need to convert 'properties', etc., to an equivalent in your 'resource units'. Find forms of words that express the outcome of the game in a manner appropriate to the overall style of your version of Technopoly. Is winning a matter of vanquishing the opposition, or might you be 'taking over' from fellow players, giving them the opportunity to 'take it easy'? You decide.

■■■

Functionality that satisfies the requirements between the dashed lines (■■■■■■) attracts up to **15** marks. Up to **5** additional marks are available for systems that, within the constraints of the text user interface, deliver the required functionality in a manner that demonstrates excellent usability, including clear, timely and engaging interaction with the players, and a novel and coherent interpretation of the Technopoly theme. (Though it is sometimes fun, elaborate 'pretty printing' – i.e. making shapes, etc., out of text – is not required for this system [remember the SUI/'words-alone' guideline above!]).

The **working system** must be uploaded to GitLab by **15:00, Thursday 14th March 2019 (Semester 2 Week 9)** and will be assessed during a **live demonstration** in Semester 2 Week 10. A detailed demonstration schedule will be published closer to the demonstration period.

Important:
In addition, teams are required to produce a **video** (with commentary; max **5 minutes**) of their working system. The video should illustrate the working implementation of the functionality requested above. The video must be in MP4 format and must run in VLC software: check that it does by using the software available at http://www.videolan.org.vlc. Please do not record your video in HD or 4K, but ensure that any on-screen text is legible. This video is required for review by the external examiner. The project cannot be marked if the video is not submitted or cannot be played by the assessors. The video should be uploaded to an appropriately named

folder in the team's project space on GitLab in Semester 2 Week 9, **at the same time as** the working system is uploaded.

Produce a Short Printed Report.

Each team should produce a short **printed** report to accompany their game.

The main body of the report should include the following sections.

A ***Requirements Analysis*** section, comprising Use Case Descriptions and an accompanying a UML Use Case Diagram (or Diagrams). The Descriptions and Diagram(s) should concentrate on the *main* sets of sequences of actions that will be realised by the system. Plan your game's behaviour so that it can cope if problems arise (e.g. what happens if two players enter the same name?), and in such circumstances make sure you have an appropriate alternative flow or an extending use case. In your Requirements Analysis, include your guide to the virtual 'board' on which your game is played (remember: this graphical representation is for guidance only; it will NOT be implemented as a GUI in this text-only system!) (**15** marks)

A ***Realisation*** section, comprising one or more UML Sequence Diagrams with a brief written commentary. The sequence diagram(s) show(s) how your software components make method calls to each other, and interact with the players, in order to realise the behaviour of the main use cases described in the previous section. (**15** marks)

A ***Design*** section, comprising a UML Class Diagram, that describes the system components. The class diagram will correspond closely to the coded implementation of the game; it should show classes and methods that support the sequences of method calls described in the previous section. Again, provide a brief written commentary on your design, pointing out any instances of good design where you have considered questions of *maintainability* and *extensibility*. (**15** marks)

*Adherence to **Process** should* be documented in appendices. Place a ***Test Plan*** for the implemented system (including details of any Junit tests that were performed) in Appendix I. A set of set of weekly ***Team Minutes*** should be placed in Appendix II. (**5** marks)

The main body of the report (excluding appendices) should not exceed **20 pages**. Individual team members should place their initials (e.g. [A.B.; C.D.]) next to the sections for which they were the principal authors.

Once the team's GitLab space becomes available, authors (whether of code or documentation) should upload their work regularly to appropriate team folders.

Teams should meet and agree the **Peer Assessment** prior to the submission deadline and include this at the front of the Printed Report.

As with the working system, the submission deadline for the Printed Report is 15:00, Thursday 14th March 2019 (Semester 2 Week 9). Submit the report at Reception, Computer Science Building. Please make sure to staple or bind your pages. Do not submit loose pages in a folder or polypocket. In addition, please submit an electronic copy of your report (PDF) to an appropriately named folder in GitLab. See the separate Activity Plan also.

Electronic assessments

There will be two short **electronic formative feedback exercises** during the semester. These exercises will test basic UML for representing *requirements analysis (use case diagrams and descriptions)* and *system design (class diagrams and sequence diagrams)* – the same elements of the UML that you will be using to document your property game, during the semester and in the final *Printed Report*. The exercises will take place on **Thursday 7th February 2019** (Semester 2 Week 4) and **Thursday 28th February** (Semester 2 Week 7) in the **Ground Floor Computer Lab CSB/0G/028** at the normal lecture time (**11:00**), **See the separate Activity Plan also.**

In summary…

What is required?	*What is assessed?*	*Marks*	*When?*
Demo	Working functionality	20	Week 10 (times t.b.a.)
Printed Report	Requirements Analysis (incl. game guide)	15	15:00, Thursday 14th March 2019
	Realisation	15	
	Design	15	
	Process	5	
	Subtotal	*70*	
Peer assessment	Participation in group work		15:00, Thursday 14th March 2019
Video	(Module Requirement!)		15:00, Thursday 14th March 2019
Electronic feedback 1	UML Use Case Diagrams and Descriptions	-	11:00 Thursday 7th February 2019*****
Electronic feedback 2	UML Class and Sequence Diagrams	-	11:00 Thursday 28th February 2019*****
	Subtotal	-	
	Total	**70**	****In ground floor lab!***

Note that there will be an electronic multiple-choice/multiple response examination, worth 30% of the module mark. The examination is scheduled to take place in Week 12: 13:00, 2nd April, 2019.

PTO for some UML samples…

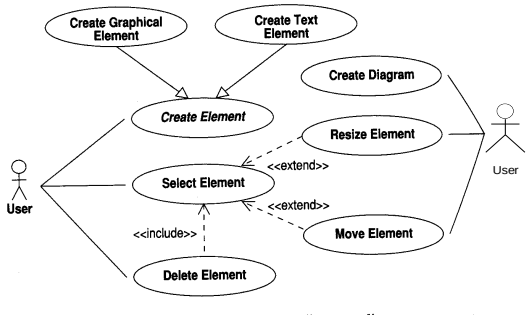
Some samples of UML notation

*It is important to look at the **full module notes** and **recommended texts** in order to appreciate the variety of ways in which the UML notation and accompanying descriptions may be used. The descriptions and diagrams below are working samples only and do not represent a full solution. Choose use case names, write use case descriptions, and create classes and objects that suit the software you are developing – your diagrams and descriptions will be different from the examples shown below!*

e.g. From Chapter 4 of the Module Notes:

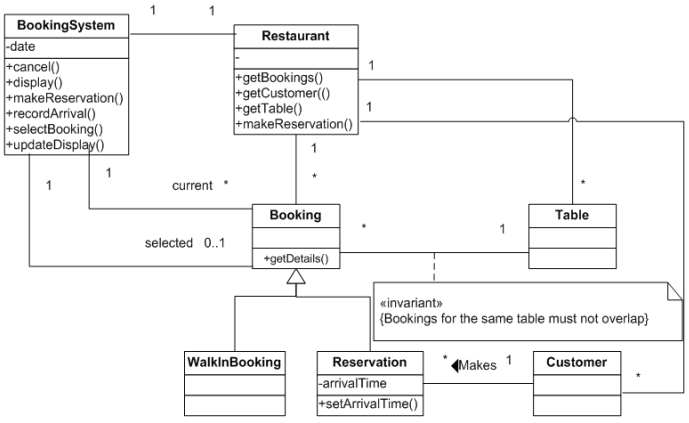
Flow of Events for the *Select Element* use-case	
Objective	***To select an element in the workspace***
Precondition	*There is an active diagram containing at least 1 element*
Main Flow	*1. The user selects the selection tool (if necessary)* *2. The user moves the cursor over an element* *3. The user presses the mouse button* *4. The element becomes selected and the control points are displayed* *5. The user releases the mouse button*
Alternative Flows	*At 3, there may not be an element. In this case no element is selected* *At 3, the element may already be selected. In this case, it remains selected*
Post-condition	*The element is selected and its control points are displayed*

A Use Case Description

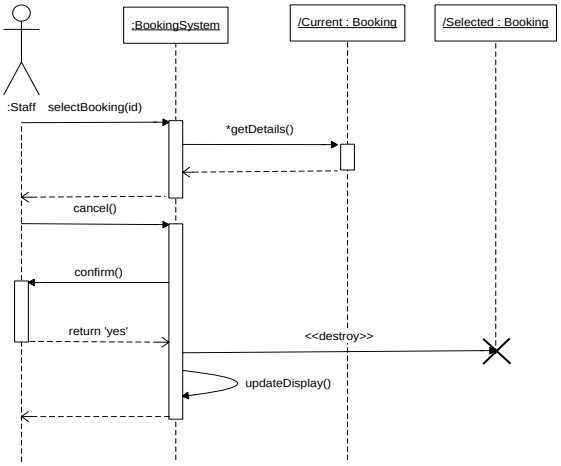


A Use Case Diagram

e.g. From Chapter 5 of the Module Notes (though this chapter is about Analysis, class and sequence diagrams are used to document design too!):



A Class Diagram



A Use Case Realisation (Sequence Diagram)