THE UNIVERSITY *of York*

BSc, BEng, MEng Examinations, 2012–2013

COMPUTER SCIENCE, COMPUTER SYSTEMS AND SOFTWARE ENGINEERING, COMPUTER SCIENCE WITH BUSINESS ENTERPRISE SYSTEMS, COMPUTER SCIENCE WITH ARTIFICIAL INTELLIGENCE
Part 1b

SOFTWARE ENGINEERING PROJECT (SEPR)
Open Examination

Issued at:
Aut/2/Wed, 17 October 2012

Submission due:
Assessment 1: Aut/7/Wed, 21 November 2012 [22.5%]
Assessment 2: Spr/3/Wed, 23 January 2013 [22.5%]
Assessment 3: Spr/7/Wed, 20 February 2013 [22.5%]
Assessment 4: Sum/3/Wed, 8 May 2013 [22.5%]

Your attention is drawn to the Guidelines on Mutual Assistance and Collaboration in the Student's Handbook and the Department's 'Guide to Assessment Policies and Procedures'
(http://www.cs.york.ac.uk/exams/statementonassessment/).

Should you wish to request an extension see the appropriate section of the Department's 'Guide to Assessment Policies and Procedures'.

- All queries on this assessment should be addressed to Richard Paige, by email, richard.paige@york.ac.uk. Answers will appear on the SEPR website.
- The four team assessments account for 90% of the total module mark. The final 10% is for the Individual Report, which is issued Sum/1/Mon, and due to be submitted by Sum/4/Wed.
- All hand-ins are by electronic submission, due by Noon on the stated date.
- All deliverables should start with the team's name. The names and examination numbers of the team members must not appear anywhere in the assessment deliverables, except in any Self-assessment Table submitted for the assessment (see Section 2).

1 SEPR Assessment Structure

The Software Engineering Project (SEPR) module assessment comprises four team open assessments and an individual report. This document defines the four team assessments. All the assessments are related, and all are based on the same scenario (Section 5); they constitute a complete software engineering project, and require teams to follow a lifecycle to build a software product. SEPR runs for the full academic year; the last task is submission of the Individual Report in Summer week 4. The purpose of the project is to give you experience in the requirements, design and implementation of software using a team approach and using modern software engineering methods and tools. The structure attempts to simulate many aspects of real-world software engineering, but in a controlled way. In this document, Section 2 outlines team set-up and marking arrangements; Section 3 describes the four team assessments, and outlines the deliverables required for each task; Section 4 addresses pass-fail criteria and reassessment; Section 5 describes the scenario and requirements for the project.

2 Project Teams and Team Marks

The project is designed to be undertaken by teams of 5, 6 or (exceptionally) 7 students. Team rules and guidance for SEPR are given on the module website, http://www-module.cs.york.ac.uk/sepr/Teams/TeamInfo.html. The module lecturers (Prof. Richard Paige, Dr Fiona Polack, Prof. Tim Kelly) are the final arbiters in any dispute arising from the composition, performance or size of teams or the distribution of marks to members of a team.

2.0.1 Team marks and self-assessment

Each assessment is marked out of 100 (as required by the University). Each team assessment is weighted as 22.5% of the module mark.

In the team assessments, the mark awarded to each student is the sum of the team mark, the result of the team self-assessment exercise, and any applicable bonus marks. This section explains the self-assessment mark.

- The deliverables for each assessment may optionally include a self-assessment table.
- If no self-assessment table is included in the deliverables for an assessment, or if a submitted table is incomplete, or the self-assessment marks are not correctly allocated, or the table is not signed by all members of the

team, then all the members of the team will be awarded the same mark, calculated as $team\ mark\ +\ any\ bonus\ marks\ +\ 10\ marks$.

- If a team believes that the individual contributions are unequal, it must consult the team rules and guidance, and is strongly encouraged to contact the module lecturers (Prof. Richard Paige, Dr Fiona Polack, Prof. Tim Kelly) for assistance. Module lecturers can be consulted at any point during the year, and timely intervention has been shown to resolve many team problems.
- In a case where a team, after due consideration and consultation, decides that an unequal self-assessment mark is appropriate for one of the team assessments, the team must include in the deliverables a *self-assessment table* laid out as shown in Table 1. The **`Self-assessment Mark`** for each member of the team must be an **integer** in the range 0 to 10 (i.e. $\{0, 1, 2, ..., 10\}$).

Table 1: Self-assessment Table

Team Name:		**Assessment:**
NAME	**SIGNED**	**Self-assessment Mark**
...	...	...
...	...	...
e.g. D. Duck	*[a signature]*	*1.5*

NOTE: If a team allocates a mark of less than 3 to one or more team members, it is likely that the team would be asked to justify its allocation to the module lecturers. This is important, as it would allow checking of the fairness of marking, and, more importantly, will identify, and promote resolution of, team problems that could also affect subsequent assessments.

2.0.2 Individual Penalties

It is important that **all** problems with participation by team members in SEPR activities and assessments are discussed in a timely manner with the module lecturers. Problems can be raised at any time, not only during an assessment activity. Whilst the module lecturers can help with problems, they cannot ensure equal participation.

A student who does not participate adequately in team activities for whatever reason, or who is awarded a low self-assessment mark for an assessment, or who otherwise misses a substantial part of SEPR, may have marks deducted from one or more team assessments. Individual penalties are applied by the module lecturers. It is normal that penalisation only occurs after discussion with the individual concerned.

Total non-participation results in **a zero mark** for the individual student, for all affected team assessments (that is, a zero assessment mark, no team bonus marks and no self-assessment mark).

3 The Team Assessments

Each team assessment addresses specific software engineering aspects of a game based on simulated control of, and fault injection to, a nuclear reactor (as described in Section 5). Teams must bear their own costs (if any), and are responsible for all aspects of management of their own work. Teams should take special note of the constraints on the software environment outlined in Section 5; there are no other restrictions on the software engineering methods, techniques, tools, etc. that teams can use.

The deliverables of later assessments build on the deliverables of earlier assessments. Teams may reuse and extend previous deliverables, including (in Assessments 3 and 4) material provided with a selected software product. For example, each software deliverable requires a brief user manual; this can reuse or extend the existing manual as appropriate.

3.0.3 Electronic Submission

All assessment deliverables are to be submitted electronically, and must conform to the requirements for electronic submission set out on the submission site, http://www.cs.york.ac.uk/submit/.

Your submission for each team assessment must be one .zip zipfile. The details of what the zipfile should contain for each assessment are given below. The zipfile must be named after the team, e.g. anchovy.zip, dab.zip.

3.1 Assessment 1: Requirements specification and design [22.5 %] due: noon, Aut/7/Wed, 21 November 2012

Assessment 1 covers the early stages of requirements specification and design, including high-level specification, planning and risk assessment. As part of the assessment, each team will need to (at least) identify appropriate software engineering methods and techniques, set up a regular meeting schedule, and allocate tasks to team members according to the available resources (e.g., team member availability and skills).

3.1.1 Deliverables for Assessment 1

1. A specification and design document of no more than 20 pages, covering the following points:
    - Requirements specification and design: a specification of the software requirements, including necessary environmental assumptions, objectives, feasibility and risks and alternatives. [25 marks]
    - A conceptual proposal for the structure of the team's software (using, e.g. classes, objects, functions, procedures or any other appropriate building blocks), and a brief justification for the structure. [25 marks]
    - Team organisation and project plan: a detailed SEPR project plan outlining tasks and their earliest starting date and latest finishing date; task priorities; a critical path and task dependencies. Outline and justification of proposed software engineering approaches, together with plans for use of development and collaboration tools to support the chosen tasks and team working. [25 marks]
    - Risk assessment and mitigation plan: risks for the SEPR project must be carefully identified and assessed, taking into account their likelihood and impact. Mitigation for each risk must be specified. [15 marks]
2. The self-assessment table for this assessment, if appropriate (Section 2). [10 marks]

The format of deliverables for Assessment 1 is summarised in Table 2.

Table 2: Assessment 1 zipfile contents

	Deliverable	**File Name**	**File Format**
1.	document (max 20 pages)	ReqRep	pdf
2.	optional self-assessment table	SelfAss1	pdf

3.2 Assessment 2: Preliminary Implementation [22.5 %] due: noon, Spr/3/Wed, 23 January 2013

Assessment 2 concerns the software engineering implementation of part of the software system, based on the requirements specification and design created for Assessment 1.

In Assessment 2, you must:

1. design and implement your nuclear power plant game, with all software and hardware components. At this stage, you must only implement a textual interface, and only *hardware components* may fail.

2. set up a website for your nuclear power plant game.

The purpose of the website is both to advertise the team's product, and to make available the software engineering documentation and code needed to extend, re-engineer and run the game. The website should highlight to a potential customer (i.e., a team that may choose this software) the selling points and strengths of this particular software. It is a reasonable assumption that a good website might attract teams to select your product, which gains bonus marks.

In Assessment 2, you must not exceed this brief: do not implement any other features.

3.2.1 Deliverables for Assessment 2:

1. A working implementation of the nuclear power plant controller game, and a brief user manual explaining how to invoke and play the game. [30 marks]

2. A test plan for the implementation and evidence of appropriate testing. [23 marks]

3. A report of no more than 20 pages that explains and justifies the architecture of the nuclear power plant controller game (what classes are used, how are they connected, etc.), as well as the design of the game's interactions. Include a sketch of your plant's architecture (your team's version of Figure 1). If the implementation does not fully address any of the features specified for the nuclear power plant controller game, then the report should state and explain what is missing. [35 marks]

4. The URL of the website for the nuclear power plant controller game as a software product. [2 marks]

5. The self-assessment table for this assessment, if appropriate (Section 2). [10 marks]

The format of deliverables for Assessment 2 is summarised in Table 3.

Table 3: Assessment 2 zipfile contents

	Deliverable	**File Name**	**File Format**
1.	Game code	A directory called Game2, containing all the files needed to run the game, including any required libraries etc.	...
	User manual	Manual2	pdf or html
2.	Test plan and testing evidence	file or directory called Test2	files: pdf
3.	report (max 20 pages)	DesRep	pdf
4.	URL of website	url2	txt
5.	optional self-assessment table	SelfAss2	pdf

3.3 Assessment 3: Selection, Extension and Integration [22.5 %] due: noon, Spr/7/Wed, 20 February 2013

Assessment 3 requires each team to work on a product other than that which the team worked on in the first two assessments. There are thus two phases to Assessment 3, as follows.

Phase 1: selection: completed Spr/3/Fri: after the submission of Assessment 2 (Spr/3/Wed, above), each team has two days to consider the products of the other teams. The **Spr/3/Thurs lecture class** can be used to discuss products with other teams. On Spr/3/Friday, each team will be required to register their choice of product to work on in Assessment 3.

The only constraint on selection is that a team is not allowed to use its own Assessment 2 software product in Assessment 3.

In selecting a software product, criteria that may be considered include: (1) the overall quality of the software product; (2) estimates of effort remaining to complete the implementation; (3) clarity and quality of the requirements specification and design.

A bonus of 5 marks will be added to the team's Assessment 2 mark for each registered selection of the team's software product, with the condition that no individual can attain more than 100% for Assessment 2.

Phase 2: extension and integration: The selected product should have designed and implemented a nuclear power plant game with all components, but only a textual interface, and only hardware components failing (as specified in Assessment 2, above). For Assessment 3, the selected software product must be extended by adding a GUI. In addition, the game must be modified so that, in the game, software components of the nuclear power plant may fail, as well as the hardware component failures introduced in Assessment 2.

1. Design and implement the GUI.
2. Design and implement the facility for appropriate software failures to arise in the playing of the game.
3. Update the website for the nuclear power plant game.

As in Assessment 2, the purpose of the website is both to advertise the team's product, and to make available the software engineering documentation and code needed to extend, re-engineer and run the product. The website is not marked directly (though marks will be lost if there is no website). It is a reasonable assumption that a good website might attract teams to select your product, which gains bonus marks.

In Assessment 3, teams are allowed, but not required, to extend the game by adding new hardware faults or plant behaviours, as desired, but these need to be fully explained and justified in the deliverables.

3.3.1 Deliverables for Assessment 3:

1. A working software application that extends the nuclear power plant controller game selected in phase 1 of Assessment 3, meeting the additional requirements of Assessment 3. A brief user manual explaining how to invoke and play the nuclear power plant controller game. [30 marks]
2. A test plan with evidence of appropriate testing, to include explicit coverage of how extensions and modifications to the product were tested. [23 marks]

3. A report of no more than 10 pages outlining the challenges that were encountered in extending the nuclear power plant controller game and in implementing the requirements of Assessment 3. For each modification, the report should include the reason for the change, the affected elements of the design and implementation and detail of the change that was made, as well as a cross-reference to the relevant parts of the test plan and testing evidence. The report should also explain and justify the software engineering solutions and approaches adopted. [35 marks]
4. The URL of the website for the nuclear power plant controller game as a software product. [2 marks]
5. The self-assessment table for this assessment, if appropriate (Section 2). [10 marks]

The format of deliverables for Assessment 3 is summarised in Table 4.

Table 4: Assessment 3 zipfile contents

	Deliverable	**File Name**	**File Format**
1.	Game code User manual	A directory called Game3, containing all the files needed to run the game, including any required libraries etc. Manual3	. . . pdf or html
2.	Test plan and testing evidence	file or directory called Test3	files: pdf
3.	report (max 10 pages)	ExtRep	pdf
4.	URL of website	url3	txt
5.	optional self-assessment table	SelfAss3	pdf

3.4 Assessment 4: Selection, Requirements Change [22.5 %] due: noon, Sum/3/Wed, 8 May 2013

Assessment 4 requires each team to work on a product other than that which the team worked on in any of the first three assessments. The teams are also required to make a presentation of their final product to an external client: the date of the presentation will be published on the SEPR website and notified by email to all students.

There are two phases to Assessment 4, as follows.

Phase 1: selection: completed Spr/7/Fri: after the submission of Assessment 3 (Spr/7/Wed, above), each team has two days to consider the products of the other teams. The **Spr/7 lecture class** can be used to discuss products with other teams. On Spr/7/Friday, each team will be required to register their choice of product to work on in Assessment 4.

The only constraint on selection is that a team is not allowed to use its own Assessment 3 software product for Assessment 4. A team is allowed to use another team's Assessment 3 extension of its own Assessment 2 product.

In selecting a software product, criteria that may be considered include: (1) the overall quality of the software product; (2) estimates of effort remaining to complete the implementation; (3) clarity and quality of the requirements specification and design.

A bonus of 5 marks will be added to the team's Assessment 3 mark for each registered selection of the team's software product for Assessment 4, with the condition that no individual can attain more than 100% for Assessment 3.

Phase 2: requirements change: A small set of requirements changes will be issued on Spr/8/Mon; these may include changes to the scenario requirements and/or introduction of new requirements. Using the selected software product, the team must modify the software product to address these requirements changes.

The presentation: In addition to the usual deliverables, each team is required to make a presentation of the completed Assessment 4 product to an external client. The presentation should assume that the client is interested in buying (or has commissioned) the product, and is thus aware of the requirements specified for the product (including the changes in Assessment 4). The client will also be interested in the major design decisions, and the quality of the software. The presentation will take place in a meeting room with access to the University computer network: within this constraint, it can use any appropriate media, and any combination of team members may take part. The presentation itself should take at most 15 minutes; it will be scheduled in a 25-minute slot, allowing at least 10 minutes for discussion with the external client. Teams should expect the client to try out their product, and potentially ask for access to designs, testing and other software engineering products.

The demonstration and (optionally) other presentation materials must be included on the product website.

3.4.1 Deliverables for Assessment 4:

Table 5: Assessment 4 zipfile contents

	Deliverable	**File Name**	**File Format**
1.	Game code	A directory called Game4, containing all the files needed to run the game, including any required libraries etc.	...
	User manual	Manual4	pdf or html
2.	Test plan and testing evidence	file or directory called Test4	files: pdf
3.	report (max 10 pages)	ChangeRep	pdf
4.	URL of website	url4	txt
5.	optional self-assessment table	SelfAss4	pdf

1. A working software application that implements the changed requirements on the game selected in phase 1 of Assessment 4. A brief user manual explaining how to invoke and play the nuclear power plant controller game. [20 marks]

2. A test plan with evidence of appropriate testing, to include explicit coverage of how modifications made to implement the required changes to the product. [15 marks]

3. A report of no more than 10 pages outlining the challenges that were encountered in addressing the requirements changes. For each modification needed to address each of the requirements changes, the report should include the reason for the change, the affected elements of the design and implementation, and detail of the change that was made, as well as a cross-reference to the relevant parts of the test plan and testing evidence. The report should also explain and justify the software engineering solutions and approaches adopted. [25 marks]

4. (Not part of the zipfile.) A presentation, suitable for an external client, that explains the capabilities of the product with reference to the requirements. The presentation should last no more than 15 minutes, and should include a demonstration, as described above. [30 marks: 20 for the presentation and 10 for the subsequent interaction: awarded by the client]

5. The self-assessment table for this assessment, if appropriate (Section 2). [10 marks]

The format of deliverables for Assessment 4 (other than the presentation) is summarised in Table 5.

4 Passing and Failing

To pass SEPR, a student must reach the overall module pass mark. There is no requirement to pass any one of the component assessments.

In marking each team assessment, the following criteria are considered:

- deliverables must be in accordance with the instructions for that element of assessment (Section 3), including zipfile instructions;
- there should be evidence of justification and rationale in the choice of methods, techniques, languages, tests and test plans, and other aspects of software engineering that are appropriate to the assessments;
- documents must be clearly expressed, in concise, precise language;
- supporting evidence (citations, diagrams, listings etc.) should be included: credit will be given for clear attribution and referencing (e.g. using the department's advocated IEEE Referencing style), where appropriate;
- design decisions and assumptions should be clearly stated;
- the requirements (as outlined in Section 5 and the individual assessments) should be traceable in the design, implementation and accompanying deliverables;
- where required by the assessment, there must be working software, with a user manual that gives a sufficient explanation for the nuclear power plant controller game to be played by the module lecturers (Prof. Richard Paige, Dr Fiona Polack, Prof. Tim Kelly);

- requirements, design and implementation, as appropriate, should exhibit clarity and efficacy; the nuclear power plant controller game should be attractive and playable;
- deliverables should demonstrate good and appropriate use of software engineering practices, and adoption of existing solutions where appropriate.

Failure of a team assessment might be due to many things. If a team is not functioning and feels that it may not be able to complete an assessment task it is vital that Prof. Richard Paige is informed as soon as possible (email richard.paige@york.ac.uk, copied to Dr Fiona Polack, fiona.polack@york.ac.uk and Prof. Tim Kelly, tim.kelly@york.ac.uk). As outlined on the SEPR website http://www-module.cs.york.ac.uk/sepr/Teams/TeamInfo.html, there are many ways we can try to help, but we cannot help if we do not know there is a problem, or if we only find out at or after the submission date.

Various characteristics of deliverables might result in low marks being awarded. For example:

- omission of required deliverables, or other failure to meet the requirements of the assessment, including zipfile instructions;
- unclear writing and reporting; poor use of notations; illegible diagrams; poor attribution or referencing, etc.;
- where relevant, code that does not compile and run in the stated software environment;
- where relevant, code that is poorly structured, poorly commented, omits conventional preamble, ignores open-source conventions, etc.;
- lack of evidence of validation between levels of abstraction, e.g. code that is not obviously an implementation of the design;
- inadequate test plans, unrepeatable tests, etc.;
- lack of clarity in rationale for, or justification of:
  - choice of methods, techniques, tools, languages, etc.;
  - any tailoring of methods etc., required to accommodate the needs of the project;
- where relevant, failure to provide and update the product website.

A student who does not pass (and cannot compensate) SEPR will be required to take a reassessment in the summer vacation. The reassessment will be an individual exercise, based on the existing team deliverables. It is the student's own responsibility to arrange access to any materials that may be held by members of their team.

5 Scenario

You are to specify the requirements and design for, and implement, a nuclear power plant controller game. The game involves simulating the control of a simplified nuclear reactor, particularly in response to faults that arise from various sources – including faults in hardware, software, and due to environmental conditions. The objective of the game is to keep the reactor operating, in spite of faults (and corresponding failures) for as long as possible while still generating power.

5.1 Simplified Example of a Boiled-Water Nuclear Power Plant

The diagram in Figure 1 is a very simplified representation of a boiled-water nuclear plant. Such plants are widely used, and are generally considered to be safe. Note that this diagram is just an example – you will build your own plant (with its own architecture) in your game.

The plant consists of three major components: the *reactor* (on the far left), *turbine* (in yellow), and *condenser* (the tank at the bottom right). Furthermore, there are three pumps (indicated in the diagram with 0 rpm – this means they aren't pumping anything!) and four valves (indicated in the diagram with WV or SV). The reactor boils the water and the steam generated drives the turbine. After the turbine, the condenser cools the steam. In turn, external cooling water cools the condenser. The cooling pumps transport the water from the condenser tank back to the reactor tank.

The reactor contains the essential components (and also the most dangerous bits!). This vessel consists of the core, control rods and reactor vessel. The core is a bundle of fuel rods containing pellets of uranium fuel. Between the rods are spaces for circulation of cooling water and insertion of the control rods.

The control rods are made of material that absorbs the neutrons that trigger chain reactions. When the control rods are lowered all the way into the reactor, all neutrons are absorbed, all chain reactions stop, and the core cannot generate power. If the control rods are lowered partway into the core, some neutrons remain to trigger chain reactions, and the core generates some power. If the h

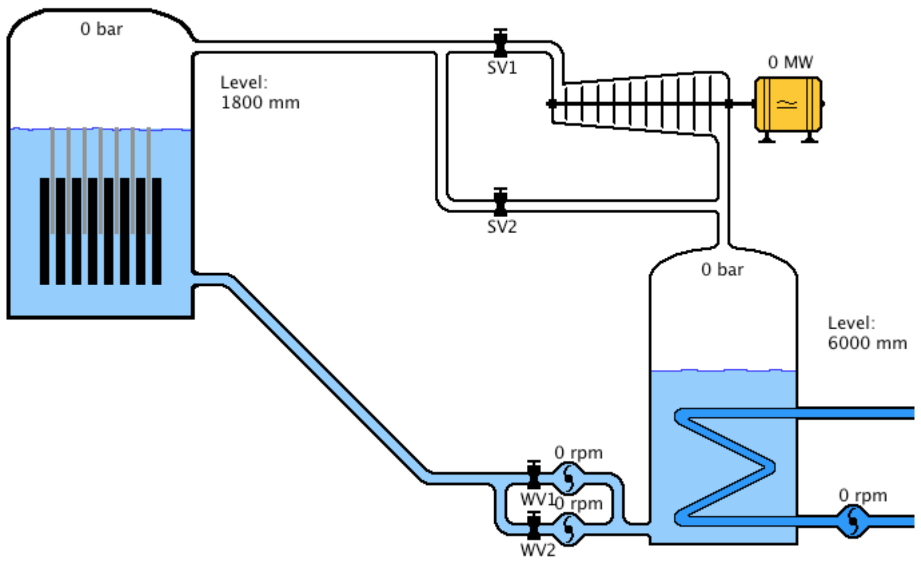

Figure 1: Simplified Boiled-Water Nuclear Power Plant: see text for explanation

control rods are raised completely out of the reactor core, neutrons trigger chain reactions and the core generates maximum power.

5.2 Game Requirements

You are to implement a nuclear plant controller game. The game should simulate the behaviour of a plant of your design. The objective of the game is to generate as much power as possible without rendering the plant inoperable. The plant should be constructed from a set of standard components (described below), which may be subject to failure. These failures must be addressed by the player, who takes the role of the plant operation, who can execute commands (as indicated in the next section) to try to resolve failures and generate power.

5.3 User Commands and Controls

It should be possible to operate the plant via reading sensors and by controlling several key components. The only things a player can do are:

- LOWER the control rods.
- RAISE the control rods.

- OPEN or CLOSE a valve.
- TURN OFF or TURN ON a pump.
- Read the temperature and pressure in the reactor tank.
- Read the temperature and pressure in the condenser.
- Read the water levels in reactor tank or condenser.

Commands may be entered either as text commands (e.g., `RAISE RODS 5`, `CLOSE VALVE 3`) or via a graphical user interface (e.g., sliding an animation of the control rods up or down via a mouse) of the team's choosing, but must conform to the requirements of each assessment.

Note: Example input with command details is intended to show the general form of commands, and is *not* a specification for any argument format or value.

It should be possible for a user to `SAVE` a game at any point, providing that the reactor hasn't failed (e.g., overheated or reached too high a pressure). It should be possible for the user to `LOAD` a saved game at any point. When a saved game is loaded, the game must continue from where the saved game left off.

It should be possible for the user to pick a name for the player-operator. All interactions between the player and the game should use this name, for example: *Timmy, your reactor has just melted down; you are fired.*

Implemented commands should be associated with appropriate behaviour. The precise commands and behaviours that are needed depend on the specific design. It is fair game to implement cheat codes, but try to avoid implementing cheat codes that make the game trivial!

5.4 The Plant

Each team should design their own plant, and are responsible for ensuring that its behaviour is (to some extent) reasonably realistic and entertaining. The plant should be constructed from the following components, which must provide the following behaviour *as a minimum*.

- **Reactor**: which contains the core. A reactor has a water level, a pressure (e.g., in PSI) and includes *control rods* as described above. A reactor may overheat (in which case it is off-line for a short period of time), exceed its maximum safe operating pressure, or may fail completely.
- **Generator**: the generator outputs power. You may assume that the generator is perfect, that is, it never fails. However, the plant itself may fail to generate power because of a fault in another component.

- **Condenser**: which condenses steam into water. The condenser has a water level, a pressure, and may operate correctly or fail completely.
- **Valve**: a valve allows water or steam to flow in either direction through a pipe at a specific rate. A valve may be open or closed, but a valve cannot fail (i.e., it is perfect).
- **Pump**: a pump distributes water through a pipe at a specific rate (in rpm). A pump may be on or off, and may fail completely, in which case it is off permanently.
- **Turbine**: the turbine takes steam as input and uses it to power the generator. A turbine is either operating, or has failed. If it has failed, then steam to the condenser is cut off and the generator will produce no power.
- **Operator Software**: this is the software component used by a human operator to control the plant. The software accepts commands (specified earlier). Commands may operate correctly (i.e., behave according to their specification), or may fail. Failure arises in two ways: either the command has no effect, or it behaves like a completely different command.

The plant in the game must satisfy the following.

1. When the game starts, the plant should be initialised to a stable state, where all components (valves, pumps etc) are working correctly. Assume that the condenser and the reactor tank have reasonable levels of water, and that some power is being generated.
2. The game should execute in a sequence of time-steps. At each time step, the game should carry out the following (you may change the ordering of these steps):
   a) Check that the reactor has not overheated. If it has, the game is over and you should inform the player of this in a spectacular fashion.
   b) Randomly determine if any non-software components have failed; if so, change the status of the failed components (discussed below). The rate of failure of components should be non-zero, but shouldn't be so high as to make the game unenjoyable.
   c) Taking into account any component failures, calculate new values for the pressure and water levels in condenser and reactor. This should be derived, as much as possible, from flow rates and pump rates determined above. You will need to carry out research on how to do this, but some hints are given on the module web page.

d) Calculate the new reactor power output.
e) Execute any safety rules for the plant (see below).
f) Accept any user commands, process them, then return to step 2a.

5.5 Safety Rules for the Plant

There are some general coarse-grained safety rules for the plant that must also be established by the game.

- Any pump that has failed operates at 0 rpm.
- If the reactor water level is too low (you may define what this means) for a sufficiently large number of time-steps, the reactor overheats (i.e., the core has been exposed for a dangerous amount of time).
- If the reactor pressure exceeds its maximum then the reactor has failed.
- If the condenser pressure exceeds its maximum then it has failed.
- If the turbine has failed, then the control rods shall be fully lowered, valves that let water flow to the turbine (e.g., SV1 in the example) are fully closed, valves that let water flow from the reactor to the condenser are fully opened, and valves from the condenser to the reactor are fully closed.

5.6 Failures

Your nuclear plant, though an engineering marvel, may be subject to failure. A number of different types of failure may occur. These were discussed broadly earlier, and can be divided into two categories.

- Hardware failure, e.g., of a pump, turbine, valve, reactor, condenser.
- Software failure, i.e., of the software controller. In each time step, at most one hardware failure and at most one software failure can occur.

5.7 Calculating New Values for Water Levels and Pressures

Calculating new water levels and pressures is a little complicated. Numerous algorithms and descriptions of flow can be found in physics and numerical analysis textbooks. An appendix giving an example of how it could be done, for the simple nuclear plant shown in Figure 1 is available on the SEPR website assessment page.

5.8 Software Environment

It is important that the software produced for these assessments is usable by module lecturers and SEPR students. If you are unsure about an aspect of the required software environment, please ask the module lecturers.

The software developed for this project is required to run in the Department's software laboratories, under either Windows or Linux (or both): marks will be deducted for implementations that do not run on at least one of these platforms. It is not required to produce software that could be run elsewhere.

Teams are expected to respect open-source conventions.

It is assumed that teams will implement their game in Java; however other appropriate languages may be used (the team should provide a justification for its choice, and may risk losing marks for poor correspondence – lack of seamless development – between design models and implementation, as well as risking not attracting selection bonuses – see Section 3).

The software product must compile and execute (as appropriate) under a version of a language that is installed in the Department's undergraduate software laboratories and freely available in the Department's undergraduate software laboratories. The software may use freely-available external libraries and packages, but these must be included in any distribution made in the context of these assessments. The game cannot use any hardware that is not part of the standard issue in the Department's undergraduate software laboratories.

5.9 KISS

If you do not know what this means, find out!

This project requires a non-trivial but reasonable amount of design and implementation effort. You are STRONGLY DISCOURAGED from adding any additional features, bells and whistles, et cetera.

A key principle of software engineering is that products should meet their requirements. There is no merit in producing something that goes beyond what is required.