**COMP1000**

**Software Engineering 1**

**20 CREDIT MODULE**

**ASSESSMENT: 100% Coursework W1: 30% Programming Tasks**

**W2: 70% Report and Project**

**MODULE LEADER: Dr. Muhammad Asad**

**MODULE AIMS**

1. Familiarize students with the fundamentals of software programming, laying a solid foundation in programming constructs, control structures, and data types.
2. Expose students to software engineering methodologies and best practices, helping them understand the processes and frameworks that lead to successful software development.
3. Introduce widely used software engineering paradigms, such as object-oriented and functional programming, to prepare you for diverse programming environments.

**ASSESSED LEARNING OUTCOMES (ALO):**

1. Employ fundamental programming constructs, such as control structures (e.g., loops, conditionals) and data types (e.g., integers, strings), to write efficient and readable code.
2. Describe widely used programming paradigms, including object-oriented programming (OOP) and functional programming (FP), and understand how these paradigms influence software design and development.
3. Select appropriate software development tools, techniques, and environments to support the implementation of simple yet effective software solutions.

**Overview**

This document contains all the necessary information pertaining to the assessment of *COMP1000 – Software Engineering 1*. The module is assessed via **100% coursework**, across two elements: *30% Report outline* and *70% Report*.

The sections that follow will detail the assessment tasks that are to be undertaken. The submission and expected feedback dates are presented in Table 1. All assessments are to be submitted electronically via the respective DLE module pages before the stated deadlines.

|  |  |  |
| --- | --- | --- |
|  | Submission Deadline | Feedback |
| Report Outline (30%) | **3rd November 2025 (15:00)** | **1st December 2025 (15:00)** |
| Report (70%) | **9th January 2026 (15:00)** | **6th February 2026 (15:00)** |

Table 1: Assessment Deadlines

All assessments will be introduced in class to provide further clarity over what is expected and how you can access support and formative feedback prior to submission. **Whilst the assessment information is provided at the start of the module, it is not necessarily expected you will start this immediately** – as you will often not have sufficient understanding of the topic. The module leader will provide guidance in this respect.

**Moderation**

This assignment brief has been moderated in line with the university policy.

|  |  |
| --- | --- |
| **Moderator Name** | **Moderation Date** |
| Dr. Rory Hopcroft |  |

**Useful Assignment Information**

Please see below some useful information regarding submissions for your modules.

**Good Coding Practices**

Like all things, no code is perfect. Just because your code compiles and runs does not mean it is perfect, there is always room for improvement. No code will achieve 100% of the available marks.

Where code submission is required by a module, it will be assessed against the following criteria:

1. **Functionality**
   1. Does your code meet all the specified requirements of the assignment?
   2. Does your code behave correctly across typical and edge-case scenarios?
   3. Does your code have appropriate error handling that does not leading to it crashing or undefined behaviour?
2. **Efficiency**
   1. Is your code optimised in terms of performance and resource use?
   2. Does your code handle functions, variables, data calls efficiently?
   3. Is there any unnecessary repetition, complexity or processing reducing efficiency within your code?
3. **Readability**
   1. Is your code logically structured and easy to read?
   2. Have you maintained standard formatting conventions like indentations, spacing and naming, within your code?
   3. Are variables, functions, and classes clearly named and purposeful?
4. **Documentation**
   1. Are there helpful comments explaining non-obvious parts of your code?
   2. Do your comments document your process, development or rationale clearly?
   3. Could someone unfamiliar with the code understand the approach you have taken?

**Use of Generative AI for Creating Code**

Each assignment element for each module will have a clear indication of the permitted level of use of AI (solo, assisted or partnered) including the generation of code. Please ensure that you **read and understand the permitted use**. If you are unsure of a particular use, please reach out to the Module Leader and ask.

We strongly encourage you to read and explore beyond the core content delivered within the modules. However, this must be done correctly following academic process. Wherever code is drawn from an outside source (e.g. you have not written it yourself), regardless of whether this is AI generated or from an online repository (GitHub, Stack Overflow etc) **you must reference the original source**. This can be done in your documentation as comments or part of your write ups. Students found to be utilising code without referencing could face an academic offence. **If in doubt speak to your Module Leader.**

**Versioning**

A critical part of a development (software or academic) is versioning. Keeping a number of iterations over the course of development ensures you always have backups to fall back on. It also provides clear demonstration of the development process you implemented.

Using online/cloud-based storage solutions and repositories provide additional peace of mind, alongside being industry practice. The university provide OneDrive to all students which offers inbuilt version history for Microsoft products. GitHub is the university recommended repository for versioning code, which includes great integration with a number of IDEs.

**Submitting Code**

It is vital that you confirm that all code that you submit is in the **correct format** and **compiles correctly** or **runs without error**. If you clean up your code before submission, please ensure that all dependencies (libraries, functions and variables) are included so that the marker can compile your code.

We can only mark what has been submitted. If the code does not run correctly, it is not our responsibility to spend time error handling to award you marks.

It is also important to **show your working**. Tidying up your code is a critical part of coding practices, but if you remove the workings that provides you with the required output, we cannot see how you got there.

In a worst-case scenario, if you remove a key part of your working, and on compiling your code it gives an error or different result, we have no way of confirming what went wrong, or indeed if you fabricated those outputs.

**Please note**, just because a piece of code has given the wrong output, it does not mean it should be deleted. Seeing these attempts with comments documenting your attempt allows us to understand where you may have made an error and provide feedback that addresses this. In some cases, you could also be awarded marks for the attempt.

**Assessment 1: Programming Tasks**

**Task:**

This assignment evaluates your understanding of fundamental programming concepts in C++. You will write one integrated program containing multiple tasks, each implemented as a separate function.

In this assignment, you must:

1) Include personalized inputs using your Student ID.

2) Add short written explanations in comments for each task.

3) Complete ONE debugging task (Task 9) to test your problem-solving skills.

|  |  |  |  |
| --- | --- | --- | --- |
| **Task No.** | **Concept** | **Description** | **Personalization Requirement** |
| 1 | Variables & Output | Read two integers and print their sum. | Use the last 2 digits of your Student ID as inputs. |
| 2 | Arithmetic Operations | Multiply two numbers and print the result. | One number must be your Student ID’s first digit. |
| 3 | Loops | Print numbers 1–(ID % 20, min 10). | End value = max(10, Student ID % 20). |
| 4 | If-Else Conditions | Check if a number is even or odd. | Use your full Student ID number for testing. |
| 5 | Switch Case | Display the day of the week for input 1–7. | Input = (Student ID % 7) + 1. |
| 6 | Arrays | Declare an array of size 5, initialize with custom values, print the 3rd element. | One value must be Student ID % 50. |
| 7 | Functions with Parameters | Calculate Euclidean distance between two points. | Use the last two digits of your ID in one point's coordinates. |
| 8 | Static Local Variables | Track how many times a function is called. | Call function (Student ID % 5)+1 times. |
| 9 | Debugging | Fix a multi-bug function averagePositive() that averages positives until -1. | Fix and explain every bug in comments. |
| 10 | Global vs. Static Globals | Demonstrate difference between global and static variables. | Use Student ID in initialization. |

**Table 2: Assessment 1 Tasks, Descriptions, and Personalization Requirement**

**Program Structure Template**

Students must structure their code as follows:

#include <iostream>

using namespace std;

// Prototypes

void task1(); void task2(); void task3(); void task4(); void task5();

void task6(); void task7(); void task8(); void task9(); void task10();

int main(){ task1(); task2(); task3(); task4(); task5(); task6(); task7(); task8(); task9(); task10();

return 0; }

**Task 9 – Debugging Task**

**Intended Behaviour:** Write a function averagePositive() that:

* Reads integers until -1 is entered.
* Ignores all negatives (except -1 as the stop signal).
* Computes the average (double) of all valid positive numbers.
* Returns 0.0 if no valid positives were entered.

**Task 9 – Description**

You are given a buggy version of averagePositive(). Your job is to:

1. Identify and fix all the bugs so that the program works as described above.
2. Add short comments explaining each fix (e.g., // Fix: Explanation).
3. Test with edge cases, including:
   1. No valid positives
   2. All positives
   3. Mix of valid positives, zeros, and negatives

**Buggy starter code students must fix:**

// Function: Computes average of positive numbers until -1 is entered

// Note: Code contains multiple logical errors for you to fix

double averagePositive() {

int sum = 0;

int count = 0;

int num = 0;

do {

cin >> num;

sum = sum + num; // May incorrectly count invalid values

count = count + 1; // Counting logic might be wrong

} while (num != -1);

// Potential issue: division correctness and edge cases

return sum / count;

}

**What you must submit for Task 9:**

* + A corrected function that meets the intended behaviour.
  + Inline comments in the form:
* // Bug 1: <what was wrong>
* // Fix: <what you changed and why>
  + A brief note describing how you tested edge cases (e.g., 'no positives', 'all positives', 'mixed values').

**Submission Requirements:**

* Single .cpp file submitted to DLE
* One PDF with completed Generative AI Declaration

**Submission of .cpp file criteria:**

* + Submit one .cpp file named: StudentID\_C1.cpp
  + Include a comment block at the top with your name and Student ID.
  + Each task must include a 1–2 sentence explanation in comments.
  + Your program must compile and run from the command line with standard input.

**Acceptable level of generative AI tool use in this assessment element**

The acceptable level of GenAI use for this element is detailed with the allowed uses listed below. This is split into three categories (**Solo Work** – work must be your own with no AI support, **Assisted Work** – some uses of AI tools allowed **Partnered Work** – AI tools integral part of the work). If you have any questions, please contact the module leader.

|  |  |  |
| --- | --- | --- |
| **Solo Work** | You must not use generative AI tools. |  |
| **Assisted Work** | You are permitted to use generative AI tools in an assistive role. |  |
| **Partnered Work** | Generative AI tool use is required as an integral part of the assessment, but transparency is required. |  |

Table 3: Acceptable Level of generative AI use

In line with the indicated acceptable level of AI use above, the following uses are acceptable:

**A3 – Code Architecture**

AI tools maybe used to help outline code architecture (e.g. suggesting class hierarchies or module breakdowns). The final code structure must be the student’s own work.

**A8 - Technical Guidance & Debugging Support**

AI tools can be used to explain algorithms, programming concepts, or debugging strategies. Students may also help interpret error messages or suggest possible fixes. However, students must write, test, and debug their own code independently and understand all solutions submitted.

Any use of AI in your work must be declared within your documentation. You **must also include a signed Generative AI Declaration as a PDF document to your submission**. The declaration form can be found on DLE page of COMP1000 under assessments brief. This form will not be included in any word count associated with this assignment.

**Assessment Criteria (Total 30%):**

The following marking scheme will be used for this part of the assessment:

|  |  |
| --- | --- |
| **Criterion** | **Marks** |
| Program compiles & runs successfully | 17 |
| Correctness of Tasks 1–8 & 10 | 33 |
| Task 9 (Difficult Debugging) – correctness + explanations | 23 |
| Personalized inputs using Student ID | 10 |
| Code clarity and output readability | 17 |

**Table 4: Marking Criteria for Assessment 1**

An illustration of the feedback is presented in Table 5.

**COMP1000 – Assessment 1 – Feedback**

**Threshold Criteria (these are indicative only):**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Criteria** | **Fail (<40%)** | **3rd (40%+)** | **2:2 (50%+)** | **2:1 (60%+)** | **1st (70%+)** | **Grade** |
| **Program compiles & runs successfully** | Code does not compile or crashes. | Code compiles but some runtime issues occur. | Code compiles and runs for most cases. | Code compiles and runs for all standard cases. | Code compiles cleanly, runs efficiently, and handles edge cases. | **/17** |
| **Correctness of Tasks 1–8 & 10** | Most tasks incorrect or missing. | Some tasks correct but with major errors. | Most tasks correct with minor issues. | All tasks correct with good coding practice. | All tasks correct, efficient implementation, well-structured logic. | **/33** |
| **Task 9: Debugging + Explanation** | Debugging task not attempted or no fixes work. | Partial fixes attempted but solution incorrect. | Debugging mostly correct, explanation vague. | Debugging correct with clear explanation. | Debugging correct, explanation thorough, shows deep understanding. | **/23** |
| **Personalized inputs using Student ID & AI Declaration** | No personalization used. No AI Declaration | Minimal personalization with errors.  Inclusion of required AI submission document. | Personalization applied but incomplete. Inclusion of required AI submission document. | Personalization applied correctly for all tasks. AI declaration form correctly submitted | Personalization applied correctly and creatively enhances clarity. AI declaration form correctly submitted. | **/10** |
| **Code clarity & output readability** | Code unreadable, poor formatting. | Some formatting, minimal comments, output unclear. | Code readable with some comments, output acceptable. | Code well-structured with clear output and useful comments. | Code very well structured, professional-level clarity, excellent documentation and output. | **/17** |
|  | *Additional feedback* | | | | | **/100** |

**Table 5: Feedback Template for Assessment 1**

**Assessment 2: Game Project (C++)**

**Task:**

You will design and implement a text-based survival game in C++ called Zombie Apocalypse Survival. The goal is to survive a randomly generated zombie-infested city, collect resources, solve puzzles, avoid enemies, and escape before time runs out.

**Submission Files:**

1. Single .exe file – runs independently on Windows without extra installation.
2. Single .cpp file – your entire game code in one file.
3. Report – A 2000-word PDF document explaining your game design, implementation steps, testing, and reflections.
   * Include an Appendix within the PDF for the completed Generative AI Declaration Form.
   * The word count of the AI Declaration Form will not be included in the 2000-word limit for the main report.

**Submission Requirements:**

1. **Executable (.exe):**
   * Must run on Windows without requiring additional software.
   * Compile with: g++ game.cpp -o ZombieSurvival.exe
   * Submit only ZombieSurvival.exe
2. **C++ Source File (.cpp):**
   * Entire code in one file.
   * No external libraries except standard C++ libraries.
3. **Report (2000 words, PDF):**
   1. Title Page: Name, Student ID, Module Code.
   2. Introduction: Game overview and objectives.
   3. Game Design: Map generation, player movement, enemies, scoring system.
   4. Implementation Steps: Code snippets & screenshots.
   5. Challenges & Solutions: Problems faced and fixes.
   6. Testing: Evidence with screenshots.
   7. Conclusion & Future Work: Improvements possible.
   8. Appendix: Completed Generative AI Declaration.

**Game Requirements: Zombie Apocalypse Survival**

Your game must include:

1. **Random City Generation**: Grid-based map with roads, buildings, zombies, safe zones.
2. **Player Navigation**: Move using W/A/S/D keys.
3. **Zombies**: Random or chasing movement patterns.
4. **Resources**: Food, health packs, ammo randomly scattered.
5. **Timer**: Escape before time runs out.
6. **Scoring System**: Based on resources collected, zombies avoided, time remaining.
7. **Puzzles (Optional)**: Locked doors, codes, or traps to progress.
8. **Multiple Levels (Optional)**: Harder maps as player advances.

**Windows Setup Instructions (Initial Setup)**

**Step 1: Install VS Code**  
- Download from https://code.visualstudio.com and install.  
  
**Step 2: Install MinGW Compiler**  
- Download from https://sourceforge.net/projects/mingw/  
- Select g++ compiler during installation.  
- Add C:\MinGW\bin to Windows PATH environment variable.  
  
**Step 3: Test Compiler**  
- Open VS Code terminal and type: g++ --version  
- If version details appear, compiler works.  
  
**Step 4: Run a Sample Program**  
Create hello.cpp with:  
#include <iostream>  
using namespace std;  
int main(){cout << "Hello, Zombie Apocalypse!"; return 0;}  
Compile: g++ hello.cpp -o hello.exe  
Run: hello.exe

**Starting Point – Basic Skeleton Code**

#include <iostream>

#include <vector>

#include <cstdlib>

#include <ctime>

using namespace std;

const int WIDTH = 10;

const int HEIGHT = 10;

char map[HEIGHT][WIDTH];

// Player position

int playerX = 0;

int playerY = 0;

void generateMap() {

for (int i = 0; i < HEIGHT; i++) {

for (int j = 0; j < WIDTH; j++) {

map[i][j] = '.';

}

}

map[0][0] = 'P'; // Player start

map[HEIGHT - 1][WIDTH - 1] = 'E'; // Exit

}

void printMap() {

for (int i = 0; i < HEIGHT; i++) {

for (int j = 0; j < WIDTH; j++) {

cout << map[i][j] << ' ';

}

cout << endl;

}

}

void movePlayer(char move) {

// Remove player from current position

map[playerX][playerY] = '.';

// Update position based on input

if (move == 'W' || move == 'w') playerX = max(0, playerX - 1);

if (move == 'S' || move == 's') playerX = min(HEIGHT - 1, playerX + 1);

if (move == 'A' || move == 'a') playerY = max(0, playerY - 1);

if (move == 'D' || move == 'd') playerY = min(WIDTH - 1, playerY + 1);

// Place player in new position

map[playerX][playerY] = 'P';

}

int main() {

srand(time(0));

generateMap();

while (true) {

printMap();

cout << "Use W/A/S/D to move. Reach E to escape!" << endl;

// Check win condition

if (playerX == HEIGHT - 1 && playerY == WIDTH - 1) {

cout << "You reached the exit. Game Over!" << endl;

break;

}

char move;

cin >> move;

movePlayer(move);

system("clear"); // For Linux/Mac to clear screen; use "cls" on Windows

}

return 0;

}

**Acceptable level of generative AI tool use in this assessment element**

The acceptable level of GenAI use for this element is detailed with the allowed uses listed below. This is split into three categories (**Solo Work** – work must be your own with no AI support, **Assisted Work** – some uses of AI tools allowed, **Partnered Work** – AI tools integral part of the work). If you have any questions, please contact the module leader.

|  |  |  |
| --- | --- | --- |
| **Solo Work** | You must not use generative AI tools. |  |
| **Assisted Work** | You are permitted to use generative AI tools in an assistive role. |  |
| **Partnered Work** | Generative AI tool use is required as an integral part of the assessment, but transparency is required. |  |

Table 6: Acceptable Level of generative AI use (see page 7 for more details)

In line with the indicated acceptable level of AI use above, the following uses are acceptable:

**A3 – Code Architecture**

AI tools maybe used to help outline code architecture (e.g. suggesting class hierarchies or module breakdowns). The final code structure must be the student’s own work.

**A5 - Language Refinement**

Used to check grammar, refine language, improve sentence structure in documentation not code. AI should be used only to provide suggestions for improvement. Students must ensure that the documentation accurately reflects the code and is technically correct.

**A8 - Technical Guidance & Debugging Support**

AI tools can be used to explain algorithms, programming concepts, or debugging strategies. Students may also help interpret error messages or suggest possible fixes. However, students must write, test, and debug their own code independently and understand all solutions submitted.

Any use of AI in your work must be declared within your documentation. You **must also include a signed Generative AI Declaration as an appendix to your submission**. The declaration form can be found on DLE (or Here on the Programme Page). This form will not be included in any word count associated with this assignment.

**Assessment Criteria:**

The following marking scheme will be used for this part of the assessment:

|  |  |
| --- | --- |
| **Criterion** | **Marks** |
| Program compiles & runs successfully | 10% |
| City generation & navigation | 15% |
| Enemy mechanics (zombie logic) | 20% |
| Resource management & scoring system | 15% |
| Timer & difficulty progression | 10% |
| Code quality & comments | 10% |
| Report quality & completeness | 20% |
| **Total** | **100%** |

**Table 7: Marking Criteria for Assessment 2**

An illustration of the feedback is presented in Table 8.

**COMP1000 – Assessment 2 – Feedback**

**Threshold Criteria in the following table is indicative only:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Criteria** | **Fail (<40%)** | **3rd (40%+)** | **2:2 (50%+)** | **2:1 (60%+)** | **1st (70%+)** | **Grade** |
| **Program compiles & runs successfully** | Code does not compile or crashes on execution. | Code compiles but major runtime errors prevent normal gameplay. | Code compiles and runs with some minor runtime issues. | Code compiles and runs smoothly with no critical errors. | Code compiles, runs flawlessly, handles errors gracefully, and demonstrates efficiency. | **/10** |
| **City generation & navigation** | No map generation or navigation; static layout only. | Map generated but mostly static; limited movement features. | Map generation and navigation functional but limited randomness or features. | Dynamic map generation with functional player movement and basic interactivity. | Highly dynamic map generation, smooth navigation, multiple features (e.g., obstacles, safe zones). | **/15** |
| **Enemy mechanics (zombie logic)** | No enemy implementation or static enemies. | Enemies present but lack meaningful interaction or AI. | Basic enemy movement or random patterns implemented with partial functionality. | Functional zombie AI with chasing patterns or difficulty scaling. | Sophisticated enemy AI with adaptive behaviours, multiple difficulty levels, and smooth integration. | **/20** |
| **Resource management & scoring** | No resources, scoring, or win/loss conditions implemented. | Minimal resource or scoring system; poorly integrated with gameplay. | Basic resource collection and scoring system partially functional. | Functional resource management and scoring; integrated with game objectives. | Well-balanced resources, scoring system, and win/loss conditions; multiple resource types with strategic impact. | **/15** |
| **Timer & difficulty progression** | No timer or difficulty scaling implemented. | Timer present but difficulty progression absent or minimal. | Basic timer and difficulty scaling partially functional. | Functional timer with increasing difficulty levels integrated into gameplay. | Highly polished timer and difficulty scaling with clear impact on player strategy and engagement. | **/10** |
| **Code quality & comments** | Code poorly structured, no comments, lacks readability. | Some structure but inconsistent commenting and formatting. | Basic structure with some comments; partial modularity and readability. | Well-structured code with clear comments, modularity, and naming conventions. | Exceptionally well-organized, modular, reusable code with professional-level documentation and clarity. | **/10** |
| **Report quality & completeness** | Report missing or far below word limit; no explanation of design. | Report present but lacks structure, depth, or screenshots. | Basic report covering game steps, some explanations, limited screenshots. | Complete report with explanations, screenshots, testing evidence, and reflections. | Comprehensive, well-written, professional-quality report with thorough explanations, evidence, and future work. | **/20** |
| **Feedback/Overall** | *Additional feedback* | | | | | **/100** |

**Table 8: Feedback Template for Assessment 2**

**Acceptable levels of AI use:**

The table below provides the acceptable use categories for GenAI. Each assessment element may allow different uses. Please check the brief for each element carefully to see what uses are allowed.

|  |  |
| --- | --- |
| **Solo Work** | **S1 - Generative AI tools have not been used for this assessment.** |
| **Assisted Work** | **A1 – Idea Generation and Problem Exploration**  Used to generate project ideas, explore different approaches to solving a problem, or suggest features for software or systems. Students must critically assess AI-generated suggestions and ensure their own intellectual contributions are central. |
| **A2 - Planning & Structuring Projects** AI may help outline the structure of reports, documentation and projects. The final structure and implementation must be the student’s own work. |
| **A3 – Code Architecture**  AI tools maybe used to help outline code architecture (e.g. suggesting class hierarchies or module breakdowns). The final code structure must be the student’s own work. |
| **A4 – Research Assistance**  Used to locate and summarise relevant articles, academic papers, technical documentation, or online resources (e.g. Stack Overflow, GitHub discussions. The interpretation and integration of research into the assignment remain the student’s responsibility. |
| **A5 - Language Refinement** Used to check grammar, refine language, improve sentence structure in documentation not code. AI should be used only to provide suggestions for improvement. Students must ensure that the documentation accurately reflects the code and is technically correct. |
| **A6 – Code Review**  AI tools can be used to check comments within the code and to suggest improvements to code readability, structure or syntax. AI should be used only to provide suggestions for improvement. Students must ensure that the code accurately reflects their knowledge and is technically correct. |
| **A7 - Code Generation for Learning Purposes** Used to generate example code snippets to understand syntax, explore alternative implementations, or learn new programming paradigms. Students must not submit AI-generated code as their own and must be able to explain how it works. |
| **A8 - Technical Guidance & Debugging Support** AI tools can be used to explain algorithms, programming concepts, or debugging strategies. Students may also help interpret error messages or suggest possible fixes. However, students must write, test, and debug their own code independently and understand all solutions submitted. |
| **A9 - Testing and Validation Support** AI may assist in generating test cases, validating outputs, or suggesting edge cases for software testing. Students are responsible for designing comprehensive test plans and interpreting test results. |
| **A10 - Data Analysis and Visualization Guidance** AI tools can help suggest ways to analyse datasets or visualize results (e.g. recommending chart types or statistical methods). Students must perform the analysis themselves and understand the implications of the results. |
| **A11 - Other uses not listed above**  Please specify: |
| **Partnered Work** | **P1 - Generative AI tool usage has been used integrally for this assessment**  Students can adopt approaches that are compliant with instructions in the assessment brief.  Please Specify: |

**General Guidance**

**Extenuating Circumstances**

There may be a time during this module where you experience a serious situation which has a significant impact on your ability to complete the assessments. The definition of these can be found in the University Policy on Extenuating Circumstances here:

<https://www.plymouth.ac.uk/student-life/your-studies/essential-information/exams/exam-rules-and-regulations/extenuating-circumstances>

**Plagiarism**

All of your work must be of your own words. You must use references for your sources, however you acquire them. Where you wish to use quotations, these must be a very minor part of your overall work.

To copy another person’s work is viewed as plagiarism and is not allowed. Any issues of plagiarism and any form of academic dishonesty are treated very seriously. All your work must be your own and other sources must be identified as being theirs, not yours. The copying of another persons’ work could result in a penalty being invoked.

Further information on plagiarism policy can be found here:

Plagiarism: <https://www.plymouth.ac.uk/student-life/your-studies/essential-information/regulations/plagiarism>

Examination Offences: <https://www.plymouth.ac.uk/student-life/your-studies/essential-information/exams/exam-rules-and-regulations/examination-offences>

Turnitin (<http://www.turnitinuk.com/>) is an Internet-based 'originality checking tool' which allows documents to be compared with content on the Internet, in journals and in an archive of previously submitted works.  It can help to detect unintentional or deliberate plagiarism.

It is a formative tool that makes it easy for students to review their citations and referencing as an aid to learning good academic practice. Turnitin produces an ‘originality report’ to help guide you. To learn more about Turnitin go to:

<https://help.turnitin.com/new-links.htm?Highlight=guide>

The university also had Draft Coach available on the online version of Microsoft Word. Draft Coach is a Turnitin plugin and provides instant feedback on how to address citation issues, grammar mistakes and matches with Turnitin’s database.

**Referencing**

The University of Plymouth Library has produced an online support referencing guide which is available here: <http://plymouth.libguides.com/referencing>.

Another recommended referencing resource is [Cite Them Right Online](http://www.citethemrightonline.com.plymouth.idm.oclc.org/); this is an online resource which provides you with specific guidance about how to reference lots of different types of materials.