Computer Science – Software Development Life Cycle

# Mental Maths Skill Builder

The purpose of this task is to help prepare you for your major project and to get you thinking about the *Software Development Life Cycle* and how it applies to your projects.

It will assess Criterion 6.

## The Task

Your task is to create a simple maths game to help someone practice their mental arithmetic skills.

The person playing the game will be able to choose the type of problem (e.g. addition, subtraction, mixed…) *and/or* the level of difficulty (e.g. easy, medium, hard).

The problems will be displayed one at a time, and the user will be able to enter an answer for each problem. The answers will automatically be graded and the results displayed.

## ATTENTION!

Although this task involves writing code, *you will not be assessed on your programming skills*, at least not directly. Unlike all the pracs we’ve had so far, this task will not be assessing Criteria 2 or 3, instead…

**The focus of this task is on Criterion 6 – it’s all about the process and the documentation!**

## Time Frame

You will have two weeks of school time to work on this project, i.e. this week and the first week back next term.

The idea is that you will spend at least the first lesson planning (and documenting!) what you’re going to do, and *then* you write some code (if you’re keen and want to spend the holidays coming up with something awesome that’s nice, but remember it’s not the focus), and next term you’ll have a couple of lessons to finish writing up what you’ve done.

## Other Constraints

You can do this task individually or in a group of up to three people. If you’re not feeling confident about your programming, then team up with someone who is!

Hint: If you’re working with someone else, you can split up the work however you like, but it’s essential that you do the initial planning together and then stay in regular contact throughout the project. If one person does the coding and another person the documentation completely independently, it looks really bad!

You don’t have to use Java, but it’s probably going to be easier to get help if you do. If you choose to use something other than Java, be sure to discuss this choice in your documentation.

## Stages of the Software Development Life Cycle (SDLC)

Requirements

Design

Implementation

Testing

Release & Maintain

**Deliverables**

1. A Word document (or equivalent) that details the following information for each stage of the Software Development Life Cycle.
2. Source code (i.e. your \*.java files).
3. Screenshots demonstrating your program running.

**Requirements**

**Objectives**

* What are you aiming to produce? Be specific!

**Requirements**

* What does the user need the software to do?
* What does the software allow the user to do?
* What does the project need to be able to do so that the user's requirements are met?

**Design Documentation**

* How will the program do what it needs to do?
* It is important to discuss at least *two alternatives* within this phase and *give reasons* for final choices.
* Design the GUI (Graphical User Interface), including:
  + Layout of components (e.g. drop downs, text fields, buttons, display areas, etc.)
  + Data flow (how data links from one screen to another, or how information links.)
* Use images, dataflow diagrams, to give as much detail in this phase of the documentation so that it is visual and easy to understand for clients.

**Implementation - Technical Documentation**

Technical Documentation is documentation written during the implementation phase. It details how the program does what it does.

Included in this phase are:

* Comments in code – essential for self or for others
* *(Optional stretch goal: If a custom class is used, include a UML class diagram)*

**Test – Plan and Report**

Devise a testing plan to ensure you have thoroughly tested all aspects of your program and describe how you have done this.

If your requirements document is clear and comprehensive, you should even be able to specify your testing plan before writing code.

At the very least, create a systematic list of inputs and/or program states and the expected outcomes.

For example…

|  |  |  |
| --- | --- | --- |
| **Test Criteria** | **Date** | **Outcome** |
| Checking for incorrect input   * Negative numbers * Words rather than numbers |  | Allows negative numbers to be put in – this needs updating. |
| Calculations come out correctly |  |  |
| Works on different machines/platforms |  |  |
| Accessibility options |  |  |

**Release and Maintain - User Documentation**

User documentation is any documentation written for the end user, system administrator, or support staff. It typically contains instructions on:

* setting up the program
* how to use the program
* what buttons do what
* how to do basic functions, etc.

# Assessment

**Criterion 6: apply the software development life cycle to a variety of problems**

The learner…

|  |  |  |  |
| --- | --- | --- | --- |
| **Element** | **Rating C** | **Rating B** | **Rating A** |
| **1** | writes straightforward programs which meet specifications using appropriate programming standards and a range of programming constructs | follows the Software Development Life Cycle (SDLC) to write complex programs that meet the specifications using the appropriate programming standards | writes well designed complex programs using the SDLC which meet the specifications using programming standards with an appropriate user interface |
| **2** | accesses and applies core support resources to assist in writing programs | uses appropriate programming constructs and accesses relevant resources | evaluates possible programming constructs and resources, and chooses the most appropriate |
| **3** | tests programs against a plan and assesses how well the programs perform | specifies detailed testing plans before a program is written and makes some program revisions after testing | specifies comprehensive testing plans *before* a program is written, uses both hand and automated tracing to debug programs and refines programs in response to the testing |
| **4** | correctly follows a given set of design principles for the user interface. | explores options for the user interface regarding the specified problem. | tests and evaluates options for the user interface for a solution and selects the most appropriate. |