COMP0035 2023/24 Coursework 2 specification

Status: Approved. Version: 2 Date: 11th November 2023.

Previous versions: Version 1 27th September 2023

Summary of changes:

- moved use of AI from the tools and techniques section to the testing section.
- modified the test automation section and marking as a workflow was included in the template repository.
- removed tools and techniques from the assessment.
- updated the coursework 2 report headings, submission list, technology and marking criteria to reflect the above changes.

Contents

Introduction

This document specifies coursework 2 which is worth 50% of the assessment marks available for the module.

This coursework continues with the project you started for coursework 1, with greater emphasis on the software development aspects of the project.

You must work as you did for coursework 1, i.e. as an individual or if in a group then in the same group.

You must use the same data set as you used in coursework 1.

The purpose of this coursework is to document the design for two web applications (also referred to as 'application' or 'products').

- **App1**: A REST API that will allow developers to programmatically access and use your dataset in their apps.
- **App2**: A web app product that makes use of your dataset via the REST API (App1).

The detail of what you need to create is in coursework content.

Getting started

Continue to use the same repository as coursework 1.

Please contact the module leader (mailto:sarah.sanders@ucl.ac.uk) if you cannot use the coursework 1 repository for any reason.

Technology

The following must be used for this coursework:

Technology	*Notes*
GitHub for source code control	Use source code control for the test code. Source code control should be in GitHub, with repositories created in GitHub classroom to allow tutors and PGTAs to gain access. If you can't use GitHub for some reason please contact the course tutor to agree alternative source code control.
Python coding environment	Groups need a python coding environment. Guidance can be found on Moodle.
PDF or Markdown for text	Text created using any software such as Word, Powerpoint, Excel etc must be saved as `.pdf`. Markdown is a widely used format in software development so you may use this instead of .pdf. GitHub provides a markdown guide (https://docs.github.com/en/get-started/writing-on-github/getting-started-with-writing-and-formatting-on-github/basic-writing-and-formatting-syntax) though there are many freely available guides online.

As most of this coursework involves diagrams, audio is not a suitable alternative. If this prevents you completing the coursework, please contact the module leader (mailto:sarah.sanders@ucl.ac.uk) to discuss.

Note: If you choose to use markdown please pay particular care to make sure that any images you include are saved in the .zip file you submit to Moodle. Students occasionally link images via a GitHub image store which does not download the images when the code is downloaded and so these may be missing from the Moodle submission.

Use of AI

You **are permitted**, but not required, to use AI (e.g. GitHub Copilot, ChatGPT) for the test code only.

You **are not permitted** to use AI for any other section.

You must state and explain your use of AI in the 'testing' section of the course content.

Coursework content

Overview

The following subsections state what must be produced. You can choose to provide any additional relevant items that you believe are useful.

Variances for groups and individuals are noted within each section.

Use techniques included in the teaching materials, or other techniques that are typically used in software engineering and data science. Please provide a relevant reference if you use a technique not included in the course materials.

Add all the content (except for the .py code files) to a single file (.pdf preferred) for submission.

Please add the content in the following order within the file. We are marking 180 submissions and if you do not follow the specified sequence it is likely parts of your work will be missed during marking:

```
Requirements
   Explanation of the choice of techniques
   Prioritised requirements
Design
   Interface design
   Application design
   Database design
Testing (groups)
   (note: most of the evidence for this section will be in 'test_code.py')
   Test automation (explanation of results)
   Test creation (use of AI)
```

Do not change the names of the files given in the starter code and use the filenames given in the following section. Automated scripts will be used to separate files for marking purposes, if you change the filenames your coursework may be missed.

1. Requirements

The purpose of this section is demonstrate you can select and apply appropriate software engineering techniques for requirements analysis.

For this section, include all evidence in `coursework2.(pdf/md)`

Item	*Individual*	*Group*	*Filename*
1.1 Explanation of the choice of techniques	✓	✓	coursework2.(pdf/md)
1.2 Prioritised requirements	✓ App2 only	✓	coursework2.(pdf/md)

1.1 Explanation of the choice of techniques

- **Select** one or more relevant techniques to use to:
    - Elicit the requirements
    - Document the requirements
    - Prioritise the requirements
- **Explain** why you chose the selected technique(s) for this project.

IMPORTANT: You must not involve anyone other than yourself or other students currently taking this course in the project. This means that for the 'elicitation' stage, you must not use methods such as online surveys and questionnaires; interviews; or inviting others to participate in a workshop as these activities require UCL ethics approval which you will not be able to obtain in time to complete the coursework.

1.2 Prioritised requirements

Apply your chosen techniques to result in a documented and prioritised set of requirements.

The requirements should be relevant to the problem statement, target audience (persona), and your prepared data set that was created in coursework 1.

There is no numerical limit for the amount of requirements as it will depend on the approach you use as to how numerous and detailed these are. For example, if you use agile user stories you *might* have between 8 and 20; if you create use cases you would likely have less, as one use case may incorporate more than one user story.

You do not need to provide evidence of the output from the elicitation techniques. For example, if you used brainstorming you don't need to show the brainstorm evidence.

All members of a group must be included in the requirements' analysis stage as you cannot work on the design without understanding the requirements. Groups are therefore expected to spend more time on this aspect than an individual.

Most students typically use the user stories technique to document requirements, however you may choose to use other techniques or even a combination of techniques. There are other models and techniques that were mentioned in the course that you may decide to apply (e.g. context diagram, UML models).

App1 requirements (Groups only)

Document the following requirements in an appropriate format. Make them specific to your app.

1. The REST API must allow a software developer programmatic access to:
    - Get existing data
    - Update data
    - Add new data
    - Delete data
2. The API must adhere to RESTful design principles.

You can add any other requirements you wish that are relevant to a REST API, e.g. if you have several 'get' requirements, if you want to authenticate users using the API, etc.

App2 requirements

Document the requirements for App2 in an appropriate format.

It is up to you to decide what App2 should do. Use what you produced for the 'product and project definition' section of coursework 1, and the type of app.

App2 is a web app that makes use of your data set via the REST API (App1). You selected the type of app in coursework 1. As a reminder the options you chose from were:

a. A data visualisation / dashboard app.
b. An app that deploys a machine learning model.

c. A web app that uses the data in some other way e.g. to generate content that has interactive features.

2. Design

The purpose of this section is demonstrate you can select and apply appropriate software engineering techniques to design an application including its interface, the application and data (database).

For this section include all evidence to `coursework2.(pdf/md)`:

Item	*Individual*	*Group*	*Filename*
2.1 Interface design	✓ App2 only	✓ App2 only	coursework2.(pdf/md)
2.2 Application design	✓ App2 only	✓	coursework2.(pdf/md)
2.3 Database design	✓	✓	coursework2.(pdf/md)

2.1 Interface design

App1 interface design (not required)

App1 does not have a user interface so interface design is not required. It is an API that is accessed programmatically.

App2 interface design

Draw wireframes that represent the interface for App2.

Points to note:

- Do not explain the wireframes or your design choices. Brief annotations can be added if really necessary to explain something that is not evident in the wirefame image.
- The design should be relevant to the requirements.
- Focus on the structure and information flow in the user interface. Aesthetic design elements such as colours, fonts, images etc. are not considered in the marking.
- Wireframes may be hand-drawn or digital. There are no additional marks for creating digital wireframes.
- Consider whether you are designing for mobile (portrait) or desktop (landscape). This will depend on your target audience.
- For charts within a dashboard you do not need to design these in detail as this will be covered in COMP0034. That is you don't need to decide on titles, axis labels, etc. Concentrate on the overall flow or layout.

2.2 Application design

App1 application design (Groups only)

Design the REST API including details of the:

- data (resources) to be provided
- data format
- URIs
- HTTP methods

Use the requirements as input to this.

App2 application design

1. Design the application. Use the requirements and wireframes as input to this.
2. Explain your design. Give reasons for choices made in the design, and any implications of those choices. This would include the use of any design patterns (design patterns can have benefits and tradeoffs); and the choice of the model/format (e.g. if you create a UML class diagram or other).

2.3 Database design

Use the requirements, wireframes and application design as input to this.

1. Document the logical design of your database

An ERD or a table style schema are likely to be the most suitable format to model the database design. Whichever format you choose it must show:

- the tables (entities)
- the attributes of those entities with data types and any constraints
- relationships between tables

Points to note:

- Include the data from your prepared data set data in the database design.
- App2 may contain features that require additional data to be created and stored, for example if you have a login feature then you will need to store users details.
- You may design either one database that is used for both apps, or separate database designs for the REST API (App1) and the additional data for App2 (if additional data is required). This is a design choice.
- You do not need to include the results of the conceptual design stage, though you are likely to start with this before completing the logical stages of database design. Do not create a physical design i.e. do not create the database itself.

2. Explain any decisions made when designing the database

For example, if you normalised the design explain the extent to which you chose to normalise and any implications of your choices.

Do not describe the contents of each table in the explanation as the diagram/design should show this.

3 Testing (Groups only)

The purpose of this section is demonstrate you can design appropriate unit tests and create the outline structure needed to run the tests automatically.

For this section you need the starter files that were included in the repository:

- `src/coursework2/employee.py`
- `src/coursework2/tests/test_code.py`
- `src/coursework2/tests/test_employee.py`

Add your work to the following files:

Item	*Individual*	*Group*	*Filename*
3.1 Test specification	✗	✓	test_code.py
3.2 Test automation	✗	✓	coursework2.py
3.3 Test creation	✗	✓	test_employee.py

3.1 Test specification

You will need the REST API design as input to this activity.

Write one test specification for each type of REST API route (GET, POST, UPDATE, DELETE) and add it to `src/coursework2/tests/test_code.py` .

- Write this as a docstring and include a suitable test case name.
- Do not implement the test code. If you want to, you can add `assert True` to avoid the linter reporting issues.
- You do not have to use the GIVEN-WHEN-THEN style syntax; use any appropriate format for Python pytest (e.g. doctest (https://realpython.com/python-doctest/)).

```
def write_an_appropriate_test_case_name():
    """
    GIVEN
    WHEN
    THEN
    """
    assert True
```

3.2 Test automation

You can start this activity at any time, you do not need to wait for the requirements and design.

1. A GitHub Actions workflow for Python was included in the template repository (hint: Go to the Actions tab, find the 'Python application' workflow, and press Configure). The workflow should:

    - Include a step to run pytest.
    - Run every time a commit is pushed to the main branch.
    - Run the tests you created in the step above (test specification) and the tests in `src/coursework2/tests/test_employee.py`.
    - Generate a report of the test results.

    You will need to edit this to achieve the above. You can also delete the provided workflow and replace it with another.

2. Export the test results in `coursework2 (pdf/m`

    - Include a URL hyperlink to your GitHub Actions workflow on GitHub.
    - Include a screenshot/image of the test report/results.
    - Explain any issues/problems that you were not able to resolve. State the issue and what you tried to do to resolve it.

Points to note:

- There may be errors in the `Employee` class. The sample test code is deliberately poorly written and does not adhere to test naming convention.
- Linting is not assessed in this coursework as you are not writing much code; however you should consider adding linting to your Actions workflow as you will be expected to do this in term 2.

3.3 Test creation

Add four tests to `src/coursework2/tests/test_employee.py`.

You are permitted to use code assistance AI (e.g. GitHub Copilot, ChatGPT) when creating the test code.

If you did not use AI, then please clearly state "AI not used" in this section.

If you did use AI then explain your use of AI in this section. For example, state which tool(s) you used then explain how you used them and to what extent the resulting code was used.

Note, if you do not explain your use of AI here, and it is found to have been used in the coursework, then it may be considered as plagiarism.

If you wish to challenge yourself, do this through showing your test skills rather than increasing the number of tests. That is, rather than add more tests, instead look at writing tests that evidence a range of skills e.g. test an error is raised, parameterize the tests, add fixtures (possibly in a `conftest.py`).

Submission

Please refer to Moodle for the deadline date and time.

Submit your work on Moodle as a single .zip in the assignment submission (see Checklist in the next section). Moodle is used as the submission date/time and to authenticate students. GitHub is NOT an acceptable alternative for submission.

Check carefully that you have the correct files with the correct file names. Automated scripts will be used to separate elements for marking; if you change the file name your work is likely to be missed.

Files that are only provided as URLs that link to external locations will be excluded from marking consideration since they may be modified after the coursework is submitted.

GitHub provides an option to download repository contents as a zip file. If your dataset is too large, please remove it from the zip file before submission.

You can include other files in the zip; though marking will only consider the files above. **DO NOT add your .venv folder to the zip file**, this creates unnecessarily large zip files.

Submission checklist

You should have the following files in the zip file:

- `coursework2/coursework2.pdf` (or `coursework2.md`)
- `coursework2/employee.py`
- `coursework2/tests/test_code.py`
- `coursework2/tests/employee.py`

The coursework sections maps to the files as follows. Markdown is also accepted in place of PDF.

Other files can be included (and will be if you downloaded the zip from GitHub). Additional files will be ignored for marking purposes

Section	*Filename*	*Item*	*Individual*	*Group*
1. REQUIREMENTS	coursework2.pdf	1.1 Explanation of the choice of techniques	✓	✓
	coursework2.pdf	1.2 Prioritised requirements	✓ App2 only	✓
2. DESIGN	coursework2.pdf	2.1 Interface design	✓ App2 only	✓ App2 only
	coursework2.pdf	2.2 Application design	✓ App2 only	✓
	coursework2.pdf	2.3 Database design	✓	✓
3. TESTING	test_code.py	3.1 Test specification	✗	✓
	coursework2.pdf	3.2 Test automation	✗	✓
	test_employee.py	3.3 Test creation	✗	✓

Marking

Mark allocation

Marks will be allocated for the coursework as follows.

It is expected each person will spend 25 hours on the coursework.

Groups will have 100 hours in total, 76 hours to complete the deliverables and 24 hours co-ordination overhead. An overhead for co-ordination is expected for groups. This has been assumed at 1 hour per person per week for 6 weeks (total 24 hours). This allows for a 30-minute weekly meeting as well as review of work items.

While not exact, an indication of the hours of effort given the marking weighting is shown below.

Section	*Group*	*Individual*
Requirements	30% (23 hours)	35% (7.5 hours)
Design	45% (35 hours)	65% (16 hours)
Testing	25% (19 hours)	-

Mark calculation

Marks for individuals will be given for each section. Each section is then weighted using the % in the table above to calculate the mark.

A group's raw mark is calculated using the same method as for individuals. All members of a group are required to complete an IPAC assessment that includes their own and peers performance. The calculated IPAC rating will then be applied to the raw group mark to calculate the mark for each person within the group, i.e. `individual group member mark = group raw mark * individual's IPAC rating`.

IPAC guidance is given in the assessment overview (https://nicholsons.github.io/comp0034-5//asssessment/assessment-overview.html#ipac-peer-assessment-for-groups) .

Grading criteria and guidance

The coursework will be assessed according to the standards set in the standard UCL Computer Science grading criteria (see copy on Moodle in the Assessment section).

The criteria most relevant to this assessment are:

- 2 Understanding of relevant issues
- 3 Engagement with related work, literature and earlier solutions
- 4 Analysis: reflection, discussion, limitations
- 5 Algorithms and/or technical solution
- 6 Testing
- Writing, communication and documentation
- 9 Formatting, spelling, visuals, clarity, referencing

The following tables gives indicators that are more specific to this coursework where possible. Given the open-ended nature of the coursework it is not possible to describe every possible aspect that could be considered for each level. Please use the tables below as a guide and not an absolute.

Higher mark bands assume that the criteria from lower bands has been achieved.

Some aspects below relate only to groups, individuals, or both. Check the coursework content section of this specification for what each needs to include.

Requirements

Grading	*Requirements*
Distinction 90+	Refer to the generic CS descriptors: "Exceptional solution (in this case the requirements). Exceptional thought and awareness of relevant issues. Sophisticated sense of conceptual framework in context."
Distinction 70-89	Uses appropriate analytical tools and/or software engineering techniques, with evidence that goes beyond the basic techniques covered in the teaching materials. Clearly demonstrates mastery of many areas of the curriculum. The requirements are comprehensive and appropriately prioritised. More complex/interesting aspects of the requirements are analysed more fully through the application of additional techniques such as use cases or relevant UML models.
Merit 60-69	Employs appropriate analytical tools and/or software engineering techniques. Clearly demonstrate mastery of some areas of the curriculum.
High pass 50-59	Uses some analytical tools and/or software engineering techniques from the course.
Low pass 40-49	Uses some analytical tools and/or software engineering techniques from the course. The requirements include the mandatory features (varies for group/individual).

Design

Grading	*Coursework-specific indicators*
Distinction 90+	Refer to the generic CS descriptors: "...Advanced technical design. Exceptional thought and awareness of relevant issues. Sophisticated sense of conceptual framework in context."
Distinction 70-89	Uses appropriate analytical tools and/or software engineering techniques, with evidence that goes beyond the basic techniques covered in the teaching materials. Clearly demonstrates mastery of many areas of the curriculum. The interface design is well presented and the data and screen flow are consistent with other materials (e.g. user stories, use cases). Database concepts such as entity relationships and normalisation are understood and there is evidence of their application in the database design. Application design shows innovative solutions to the technical challenge of the project, e.g. application of relevant design patterns
Merit 60-69	Uses appropriate analytical tools and/or software engineering techniques. Clearly demonstrates mastery of some areas of the curriculum. The interface design is clearly presented and is consistent with other materials (e.g. user stories, use cases). Application design is appropriate, clearly presented and it can be seen how this relates to the requirements. Database design shows understanding of relationships. The tables, data types and constraints have been considered and are relevant to the project.
High pass 50-59	Uses analytical tools and/or software engineering techniques from the course. The interface design is complete but may lack clarity of completeness. The application design is too simplistic meaning that it does not adhere to software engineering principles and/or does not relate well to the requirements. The database design does not consider all of the data suggested by the requirements and/or there are issues with the design that would cause problems if the design were to be implemented.
Low pass 40-49	Uses some analytical tools and/or software engineering techniques from the course. The interface design may be incomplete or not appropriate for the requirements. The application design is too simplistic meaning that it does not adhere to software engineering principles and/or does not relate well to the requirements. The design covers the mandatory features given in the specification (varies for groups/individuals).

Testing

Grade band	*Coursework-specific indicators*
Distinction 90+	Refer to the generic CS descriptor: "Exceptionally comprehensive testing, extremely thorough approach to testing."
Distinction 70-89	Test code quality is high throughout and consistent with relevant Python test standards (or commonly used styles) There is a range of tests. May provide evidence that error and/or edge cases have been considered. There is evidence that the tests have been run and the results provided. Evidence that test set-up goes beyond the core requirement and shows sophistication in the use of testing techniques e.g. effective use of fixtures, coverage reporting, parameterised tests, error conditions, etc
Merit 60-69	Test code quality is good and consistent with relevant Python test standards (or commonly used styles) The tests show evidence of more than one technique, or complexity. Complexity can be in many ways e.g. effective use of setup/teardown or fixtures, tests with multiple values, etc. Further, the tests would show variation, that is the tests would not be almost identical e.g. two tests that use an assert that tests for a straight forward success condition would not be considered to be showing range or complexity. There is evidence that the tests run successfully (this does not mean the tests pass as that implies there are no errors in the code being tested which may not be the case).
High pass 50-59	Tests show evidence of more than one technique. These may contain errors though the general structure and test approach is appropriate for the software with some further work the tests could be made to run.
Low pass 40-49	The tests are simple/straightforward and/or inappropriate. The tests are likely to contain errors that prevent them from running.

Appendix

The appendix contains supplementary guidance and is common to all assessments on this module.

Data sets

Only use the ethics approved data set

The data sets (https://moodle.ucl.ac.uk/mod/page/view.php?id=4787524) allocated to students on this course have been approved by the Computer Science Ethics Committee and signed by the Head of Department for use in this course. These data sources comply with GDPR and UCL ethics and data protection policies. You must not use any other data set for the coursework.

The approved data sets do not require data protection registration and since you will not be working with participants in your project there is no requirement to complete any further training relating to data protection and GDPR.

Data set selection was guided by the following principles:

- does not contain data related to, or collected from, humans. There are some very limited exceptions to this and these also demonstrate:
    - the data was gathered lawfully and in accordance with GDPR, and that this can be verified
    - consent was given by the human participants and this consent can be verified
    - the data is wholly anonymous
- is available in the public domain (and further does not require a password to access it)
- is not sensitive
- grants permission for its use (i.e. there is an stated license permitting you to use it)

Complete the mandatory ethics training

UCL computer science provides a series of videos aimed at introducing students to the ethical implications of using secondary data in computer science projects, and in particular the processes that must be followed at UCL.

You must complete the two mandatory ethics training videos as shown on Moodle.

Assessment

Examples of previous coursework

Partial extracts from previous submissions are given on Moodle as examples.

Full coursework examples are not provided since the coursework remains similar to the previous year, and it would then be too easy to simply copy and adapt the coursework from a previous student. Turnitin plagiarism detection is used.

You are encouraged to start work early on your coursework and use the tutorials to gain formative feedback.

Time estimates for assessment

An *average* module is expected to take around 150 learning hours (https://www.ucl.ac.uk/module-catalogue/glossary-terminology#learning-hours).

To help you plan your time, the coursework is designed with the expectation you will spend the following time on it:

- 2.5 hours per week (all weeks except reading week)
- 7 hours during reading week
- 10 hours during Christmas/Easter periods

Each coursework should take 20-25 hours.

All assessments are based on projects that start in week 1 and run throughout the module. You are expected to make progress each week. There are weekly activities and checkpoints in Moodle to guide your progress.

Support and guidance

Module staff will be available to provide support and guidance in weekly lab sessions.

The tutor will be available during a weekly module office hour (see Moodle).

You can ask questions at any time in the Moodle Q&A forum where anyone in the course is able to reply.

IPAC peer assessment for groups

Any assessment for groups includes peer assessment using the UCL IPAC assessment software and process.

Please read the IPAC student presentation (https://moodle.ucl.ac.uk/mod/resource/view[illegible]4928189) if you are not familiar with IPAC. All IEP students should be familiar with this as it was used in ENGF0001 and other IEP modules.

For each assessment all members of a group must complete an IPAC evaluation that includes their own and peers performance. This determines an individuals IPAC rating. The calculated IPAC rating will then be applied to the raw group mark.

For example, for a coursework graded at 60%:

- a student with an IPAC of 0.9 would receive a mark of 54%
- a student with an IPAC of 1.0 would receive a mark of 60%
- a student with an IPAC of 1.1 would receive a mark of 66%

The criteria used in the IPAC are:

Criteria	*Definition*
Attendance	Attending scheduled tutorial sessions, meetings with the TA and other meetings arranged outside of these
Effort	The student's attitude to work in and out of the class, reflects their attentiveness and contribution to discussion and work load
Quality of work	A reflection of how complete, accurate and informative the work the student produced was
Teamwork	Demonstrates a cooperative and supportive attitude

Note that:

- the self-promotion score, which is provided by the IPAC software, quantifies how a student has valued his/her contribution in regard to the contribution of the group (ratio of averages). If the self-promotion score is too high (over 1.25), then the software ignores the scores given by that student from the calculations.
- 0.5 is the minimum IPAC score a student needs to have before being allowed to take his/her peer marks into account. The reasoning is that those students that do not contribute to the group cannot accurately rate their peers.
- student comments are moderated both automatically by the software and by the course tutor
- different students will be more generous or harsher when assessing their team, the IPAC software applies bias correction

Requests to change coursework submission dates

Coursework submission dates are set centrally in Computer Science and not by the course tutor.

These have been planned carefully to fit with the teaching schedule.

Requests to change these for all students must be made to the course tutor (mailto:sarah.sanders@ucl.ac.uk) who will then forward the request to the relevant authority in the computer science department. Requests must be supported with appropriate evidence of the need for change.

SoRA and EC

SORA: alternative formats

If you need the assessment or any teaching materials in an alternative format than has been provided, please contact the course tutor directly <sarah.sanders@ucl.ac.uk>.

If you need/prefer to submit the assessment in a way that minimises use of the written word, audio recordings instead of markdown text are accepted. Please don't use video, the files are too large, and you are more likely to be recognised (marking should be anonymous).

If you choose to use markdown and are concerned with spelling and grammar, there are spell checkers available for VSCode (https://marketplace.visualstudio.com/items?itemName=streetsidesoftware.code-spell-checker) and included in PyCharm. The mark scheme does not focus on spelling and grammar.

SORA and EC implications for groups

Where a group member has an approved statement of reasonable adjustment (SoRA) or extenuating circumstances (EC) that allows for 1-2 weeks delay to submission, then the approved revised submission date will apply to the whole group. This is because it is not possible to separate an individual's contribution from the group's submission.

In the rare situation where a group member has an approved EC that extends beyond 1-2 weeks, e.g. into LSA, then the impact on the group will need to be considered on a case by case basis. It is more likely in this event that the student concerned would need to submit a new individual assessment and the group would continue to the original submission date.

Late submission penalties

Late submission rules apply to all assessments. Any penalties for late submission are applied by the computer science teaching and learning team when marks are entered in portico. The mark you see in the Moodle gradebook, therefore, is the mark before moderation or any penalty has been applied.

If you have an approved extension resulting from a SoRA or EC, these will be carefully checked by the teaching and learning team before marks are entered in portico. However, the deadlines in Moodle may not be modified for individuals, so it may appear on Moodle that your submission is late even though you have an approved extension to your late period.

Referencing

Book, journal, web article etc.

For book, journal, and web article references, follow the referencing style you use in your own department (e.g. Harvard, APA). Different departments in UCL use different styles. Include these in a 'References' section in your markdown.

Code

When you copy code from an external source, whether you are copying a snippet of code or an entire module, you must credit the source. This includes where you copy and then adapt the code. External sources include:

- user forums e.g. stack overflow (https://stackoverflow.com/)
- open source repositories e.g. GitHub (https://github.com/) repos where an explicit open source license is stated
- communities associated with a library e.g. plotly forum (https://community.plotly.com)
- course teaching materials
- official library documentation and tutorials of the python libraries and frameworks

It is not appropriate to copy from other students on the course, past or present.

For COMP0034 and COMP0035 you do not need to cite code from the course teaching materials or boilerplate code from the official library documentation of the required python libraries and frameworks that are expected to be used in the coursework (used in the courses e.g. pandas, flask, dash, bootstrap).

UCL has no single method for referencing code. For the purposes of this coursework use the following.

- Give the author, URL and the date of retrieval. If you adapted the code, indicate "Adapted from" or "Based on".
- Include citations within your code using code comments close to the affected code.

An example:

```
def play_sound(button_text):
    # Adapted from code from 'remdog' on the Plotly Community Forum at
    # https://community.plotly.com/t/linking-scatter-plot-elements-to-audio-
files/11908/6
    # Accessed 01/02/21
    ...some code here...
```

Academic terms

It is assumed that you are familiar with essay terms such as 'describe', 'explain', 'discuss' and 'justify'.

There are numerous sources that provide guidance on these and academic writing in general:

- Essay question words (https://www.oxbridgeessay.com/article/explain-evaluate-other-essay-question-words/)
- UCL Guidance for academic writing (https://www.ucl.ac.uk/teaching-learning/study-advice/writing-centre/)
- UCL students union academic writing guide (https://www7.studentsunionucl.org.uk/advice-and-support/academic-writing-guide)