COMP2931 Coursework

Semester 2 Team Project

1 Introduction

This project involves development of a system for managing training courses, further details of which are given in Section 4. The project idea comes from FDM Group, one of the largest recruiters of STEM graduates in the UK. FDM are collaborating with us on the Semester 2 projects this year.

Development is done in teams nominally consisting of 5 students. The teams have already been allocated; see announcements in the VLE for further information on this.

Project ramp-up week begins on Monday 6 February. You are strongly advised to hold your first team meeting during this week. You should use this initial meeting to exchange contact details, discuss the skill sets of team members and consider technology choices for the project. During ramp-up week, each team member should also check that they are able to access the team's GitLab project area (see Section 3).

Sprint 1 begins on Monday 13 February. Each sprint lasts for two weeks. You will have three full sprints before the Easter vacation, during which we expect the bulk of the functionality to be implemented. The remainder of the time—encompassing the Easter vacation and the first week back after the vacation—should be used to fix bugs, polish the implementation and finalise deliverables. The project concludes with a demo of the finished product.

2 Approach

2.1 Overview

The broad approach to be followed by all teams is a simplified and scaled-down version of the Scrum method already discussed in lectures.

The process is driven by the **Product Owner** (PO), who will provide each team with an initial prioritised list of requirements in the **product backlog**.

The primary period of development is broken down into a sequence of three two-week **sprints**. At the start of each sprint, team members will have a **sprint planning meeting** in which they select a subset of product backlog items to implement during the forthcoming sprint, identify the tasks required to deliver those product backlog items and allocate those tasks to team members.

In our simplified version of Scrum, the PO will not be present at these planning meetings. The team will therefore need to check the product backlog before each planning meeting and seek clarification from the PO if necessary, either face-to-face or via email.

Teams should hold two or three **status meetings** during each sprint. Status meetings are short—no more that 15 minutes in length—and involve each team member giving a brief description of what they have done since the team last met, what they will be doing until the team meet again and any issues they have that might prevent progress. Discussion of issues should be deferred to other meetings or online communication, not necessarily involving the whole team.

A sprint concludes with a **sprint review meeting**. This should begin with, or follow soon after, a brief demo of progress to the PO. All members of a team are expected to attend this meeting.

2.2 Scrum Master

The Scrum Master (SM) should organise and chair all of the formal meetings (sprint planning, sprint review, status meetings). The SM may wish to delegate note-taking duties during meetings to another team member. The note-taker, whoever they may be, is responsible for transferring their notes to the project wiki (see below). Formal notes are not required for status meetings.

The SM should keep attendance records for all meetings (including status meetings), using a page in the project wiki. The SM should also investigate sensitively any absences or missed task deadlines, via email or other means as appropriate.

In our modified version of Scrum, the SM is also a developer but is entitled to do less development work than others on the team because of the aforementioned duties.

The SM role can be rotated amongst team members, but changes in SM are only permitted between sprints, not during a sprint.

2.3 Team Members

Team members should attend all meetings if possible and provide apologies for any absences.

Team members should use the project's **issue tracker** (see Section 3.2) to record information about the tasks they are carrying out. Other project documentation should be stored in the wiki (see Section 3.3).

Team members should use Git version control for all programming activities (see Section 3.1) and should push the changes they have made locally back up to the remote project repository on a regular basis, to facilitate code review and sharing of the new code with others in the team.

Team members can work on tasks alone or in collaboration with others. Teams might like to consider a 'pair programming' approach such as that advocated by XP, for instance. This is particularly recommended for complex or critical parts of the system.

3 Project Tools

Project management is done using a remotely-hosted instance of GitLab, at `https://gitlab.com`. Project areas for each team have already been set up and team members have been added to these areas, using the GitLab accounts that you created last year for COMP1721.

A vital first task for every team member is to check whether you can successfully access your team's project area. Make sure you do this before Sprint 1 starts. Speak to Nick if you encounter any problems with access to your team's project area.

3.1 Version Control

All source code must be managed using the Git version control system.

Each team has a shared repository, which can be accessed via the HTTPS or SSH protocols:

HTTPS	`https://gitlab.com/comp2931/`*`team`*`.git`
SSH	`git@gitlab.com:comp2931/`*`team`*`.git`

Substitute your team's name (in lowercase) for *`team`* above.

By default, any clone, fetch or push operation involving the remote repository will require authentication. This can be done by supplying your GitLab username and password when prompted.

It is possible to avoid manual authentication on every clone, fetch or push by using the SSH protocol and uploading an SSH public key to your account. For further information on this, see

`https://gitlab.com/help/ssh/README.md`

If you've not already set up SSH access, we recommend that you do this before Sprint 1 begins.

One other important decision that each team needs to make before Sprint 1 begins is their preferred Git workflow. Please refer to the lecture materials for detailed discussion of this. Your choice will determine how GitLab is set up. There are two options:

1. Selected individuals have Master permissions, everyone else has Developer permissions
2. Everyone has Master permissions

Option 1 supports a 'feature branch' style in which those with Developer permissions work in separate branches and then submit merge requests. Someone with Master permissions has to review and accept each merge request in order for it to be merged into the master branch.

Option 2 is an apparently simpler workflow in which any team member can push to the master branch of the repository—with the risk that somebody could break the system for everyone else!

Note: **You *must* make a decision about which model you want to use and then communicate this decision to Nick.** You won't be able to collaborate properly until you've done so.

3.2 Issue Tracker

Each team's project area includes an issue tracker, which can be accessed at

`https://gitlab.com/comp2931/`*`team`*`/issues`

where *`team`* is your team name in lowercase.

Your team should agree on some suitable issue labels and create them before recording any issues in the tracker. GitLab provides a convenient 'generate' shortcut that will create a reasonable set of labels, which you can then edit as you see fit. In addition to setting up some labels, you should also define milestones. 'Sprint 1', 'Sprint 2', 'Sprint 3' and 'Final Demo' are needed, but you can define others if you wish.

Use the tracker to record the tasks given to each team member. That team member should be recorded in the tracker as the 'assignee' of the issue. That team member is also responsible for closing the issue when the task has been completed.

Use the tracker also to record bugs found during testing.

Issues are written using a lightweight mark-up language called 'GitLab-flavoured Markdown'. Take some time to familiarise yourself with it. Help can be found at

`https://gitlab.com/help/markdown/markdown.md`

3.3 Wiki

Each team's project area includes a simple wiki, which can be accessed at

`https://gitlab.com/comp2931/`*`team`*`/wikis/home`

where *`team`* is your team name in lowercase.

The wiki should be used to host meeting notes, meeting attendance records and a list of team members along with their contact details. When you begin work, you should edit the wiki home page so that it identifies your team by name, describes the project briefly and provides links to the aforementioned information. Other content will be added over the course of the project (see Section 6).

Wiki pages are written using GitLab-flavoured Markdown, so take some time to familiarise yourself with the syntax.

4 Problem Area

The problem to be solved here concerns training centre operations run by a company such as FDM. The system you develop will need to manage information about the classrooms used for training activities, which can be in various training centres located in different cities. Classrooms can be of different types (seminar room, computer lab, etc) and can have different seating capacities. They can also be equipped with different facilities and have differing degrees of accessibility. You might find it useful to consult the University's teaching space pages at `http://ses.leeds.ac.uk/rooms` for ideas here.

The system will also need to manage information about the courses that are scheduled to run in these classrooms. Courses will need to have a title, a description and a maximum number of delegates (which will obviously need to match room capacity in order for scheduling to take place). Courses may require specific facilities and may have prerequisites—i.e., courses that delegates must have previously completed successfully before being allowed to book a place on the subsequent course.

Finally, the system will need to manage information on two types of people: trainers who are delivering courses, and delegates who are attending courses.

In terms of functionality, the system will need to support two kinds of user. The first is an administrator who creates a new course, assigns a trainer and schedules the course to run in a particular classroom at a particular time. The second is a potential delegate, who needs to be able to book a place on a course.

5 Implementation Notes

A client-server solution with an underlying database is required, but otherwise you have a fair degree of freedom as to what technologies you use. The skills you've acquired from modules studied last year and this year will open up a range of options to you, and we will allow you to choose whatever technology suits your team's interests and abilities, subject to the constraint that it must be runnable on the School's computing facilities. Dependencies on external services are acceptable provided that you've fully documented how to get your system up and running with those services and that you have simplified and automated the process as much as is reasonably possible.

In choosing technologies, be realistic about your team's capabilities and level of motivation, and about what you will be able to get done in the time available. Keep in mind that a reasonably good mark is achievable for relatively modest implementations provided that the organisation and management of the project are good (see Section 6).

5.1 Server

We don't recommend that you make your server TCP-based, as working at this low level would require considerable effort. You are likely to find that an HTTP-based solution is easier and more flexible.

If using Java, you can create a reasonably powerful HTTP server with Jetty. You should also consider using higher-level frameworks such as Jersey or Spark, since these make it easy to create a nice REST-based API that your clients can use.

If using Python, a simple server can be implemented fairly easily with the `http.server` module from the standard library. For something much more powerful, take a look at Tornado. The popular web frameworks Flask and Django (see below) also come with simple HTTP servers included that would be sufficient for development and demo purposes.

5.2 Desktop Client

The user interface needed by the administrator could take the form of a desktop application. Doing this in Qt would be the most logical choice. You could use either C++ and Python here. Note that Python bindings for Qt4 are available on SoC machines; to activate a version of Python that has these bindings, use

```
module add anaconda3/4.1.1
```

If you'd prefer to implement a Java client, JavaFX or Swing will be your easiest framework options, as both are provided with the JDK. The former is likely to yield the best results. Remember that the textbook issued to you in COMP1721 last year contains some useful material on developing with JavaFX.

5.3 Mobile Client

The user interface needed by course delegates could take the form of an app for a mobile device. Developing for either Android or iOS is acceptable here, but if you choose the latter then please use Swift as the implementation language rather than Objective-C.

We will offer some optional sessions designed to help you get started with Android development, but keep in mind that considerable effort may be required to develop the level of understanding needed to produce a good app. The mobile option is one for confident programmers only!

5.4 Web Application

You could implement your entire solution as a web application, with the browser acting as the client, providing the user interface needed by both administrator and course delegates. Or you could implement a hybrid solution in which the web application serves the needs of the administrator only and course delegates use a mobile app to access the service (for example).

A web application solution would need to use HTML5, CSS3 and JavaScript for the user interface, and would preferably follow responsive design principles so that it works with a range of different screen sizes. Consider using a framework such as Bootstrap here.

The server-side parts of the web application are best written in Python or Java (although other languages may be acceptable—speak to Nick if you need advice on this). If you choose Python, we recommend the use of either Flask or Django to simplify development. Note that Flask is available on SoC machines as part of the aforementioned Anaconda Python distribution. Django can be accessed if you set up your `PYTHONPATH` to include `/home/csunix/scpython/lib`.

If you prefer a Java solution, you could create your application using Java Servlets or JSP and run the application with Jetty (mentioned earlier), or you could use Spark (also mentioned earlier). Another option would be the Spring framework.

5.5 Data Storage

The preferred option here is an SQL database of some kind. The simplest solution is an embedded database—e.g., using SQLite. This is provided as standard with Python, and SQLite drivers are readily available for Java; alternatively, you could use Java DB, which is included in the JDK.

A more sophisticated solution would make use of a separate database server, either running locally or as an external service. Java DB is a convenient solution here, as it will also run as a local server. If you want to use an external service running MySQL, PostgreSQL, etc, you will need to ensure that database set-up is as automated as possible. You will also need a fallback solution to handle cases where that external service is unavailable.

6 Deliverables

6.1 Software

This earns up to **40 marks**.

Your software should be provided as source code and in executable form if appropriate. You should edit the `README.md` file at the top level of your repository, adding a brief explanation of how to run your system. Make sure that your solution is packaged in such a way that it exists and runs independently, outside of any IDE. You should provide scripts of some sort that take care of all build, installation and setup issues; ideally, it should be possible to run the server side of the system by typing a single command (and likewise for the desktop application, if you solution includes one).

Your system should have a ‘demo mode’, in which it runs with sample data suitable for demonstrating functionality. Demo mode should run fully automatically, with no dependencies on any external services. It should be as foolproof as possible, such that it can be run via a single command by a person who has no prior knowledge of your project or the technologies that it uses, at any point in the future. Note that we fully intend using this feature of your solutions at open days and applicant visits.

We will assess your work by considering the following questions:

- How well does it meet the requirements?
- Does it exceed the requirements in useful ways?
- Is performance good enough?
- Does the system manage data sensibly?
- Does the system have suitable user interfaces?

- What is the level of technical complexity?
- Has the system been tested with sufficient thoroughness?
- Is the code structured sensibly?
- Is coding style uniform and reasonable?
- Have comments been used appropriately?
- Has the software been packaged correctly?
- Are the instructions for installation/execution sufficient?
- Is installation/execution scripted and sufficiently simple?
- Does the system have a suitably straightforward demo mode?

6.2 Version Control Repository

This earns up to **10 marks**.

We will assess it by considering the following questions:

- Has version control been used correctly by all members of the team?
- Are the commit messages sufficiently informative?
- Are commits distributed sensibly throughout sprints?
- Has there been an effective process for combining the work done by team members?

6.3 Issue Tracking

This earns up to **10 marks**.

We will assess it by considering the following questions:

- How effectively have labels and milestones been used?
- Have issues been created to record tasks allocated to team members?
- Have issues been created for other reasons–e.g., to track bugs?
- Is there discussion around issues (where appropriate)?
- Have issues been resolved appropriately?

6.4 Wiki

This earns up to **20 marks**.

Your wiki should be used to record information about your team and the meetings that you have. You should use it to record evidence of your design and testing activities. You should also use it to host documentation on how to use your software.

We will assess it by considering the following questions:

- Is there a suitable home page identifying your team and its members?
- Are there records of all meetings and are they detailed enough?
- Is there good information on design decisions and the design process?
- Is there good evidence of user interface design (sketches, wireframe mockups, etc)?
- Has suitable evidence of a systematic approach to testing been provided?
- Is the user documentation sufficiently clear and detailed?

6.5 Individual Report

This earns up to **20 marks**.

It should be reflect on the successes and failures of the project and on the specific contributions that you made to it. It should be no more than three A4 pages in length. You will be given more detailed information on the expected format and content of the report later, nearer the time of submission.

7 Submission

The deadline for deliverables is **10 am, Thursday 4 May 2017**.

We expect to see a single submission of software from each team. This should take the form of a single Zip archive. It should contain everything required to build and run the system (both normally and in 'demo mode'). It should include the `README.md` file from the top level of your repository. It should *not* include the project history (contained in the `.git` directory of your team's repository). The Zip archive should be submitted via the link provided for this purpose in the *Semester 2 Project* folder in the VLE.

Each team member should submit their own individual report, as a PDF document, via the link provided in the *Semester 2 Project* folder. Reports will be checked for plagiarism using Turnitin.

8 Marking

A group mark (with a maximum of 80) will be awarded for your software and project area. This means that every active team member[1] will normally receive the same mark for these deliverables, **unless the team has requested an unequal distribution of marks via the form provided for this purpose in the VLE**. This form must be signed by all active team members.

Note that reserve the right to impose an unequal distribution of marks for the team deliverables ourselves, should we feel that the situation warrants it.

The group mark is combined with the individual mark for your report. Your final individual mark for the project will therefore be on a scale of 0–100.

[1] An active team member is one who has made a non-negligible contribution. Inactive team members will simply be awarded a mark of 0 for the project.