COM2027 Software Engineering Student Guidebook

Version 2.0, updated 6 February 2017, by Norman Poh (n.poh@surrey.ac.uk)

This document will freeze by the end of Week 5 (10 March 2017), meaning that after this period it will not get updated except the sections that are volatile, e.g., FAQ and timetable.

Notes:

Grade descriptors and the assessment sections are subject to change.

Changes:

Contents

1. Introduction

The Software Engineering project enables you to apply your skills in Java programming as well as newly acquired software engineering and project management skills to deliver a piece of useful software, including game. You will work in a team and will use a series of programming and documentation tools, such as the version control system GIT, debugging tools, the LATEX typesetting system, Markdown and the JavaDoc source code documenting system, among others. You will work with the Android Studio as the main Integrated Development Environment.

Theme-based approach

Each year, we will select a problem theme for you to work on. This year, the selected theme is "Gamification for Dementia". Your software apps can tackle the theme from different perspectives and target different end-users. For example, for this year's theme, your end-users could be: social worker, carer, or patients with dementia. Your apps can be about assistive living, brain training, disease assessment (including prognosis and diagnosis), aide memoire, etc.

Because of the specific problem chosen (i.e., dementia), a talk has been scheduled to set you off in the right direction. This year, we are honoured to have the founder of MindMate app (http://www.mindmate-app.com), Mr Patrick Renner, to share his experience.

More description of this will be put in the Appendices once available. To get inspiration, please check out games available on MindMate.

Module Expectations

Implementing a set of requirements. You are expected to understand a range of requirements (e.g., functional, technical, legal ones) and use software design and development tools,

Working in team. The software engineering project is designed to work in team. A diverse set of skills are required to put a creative app together. For example, it is not possible for one or two students to do all the coding, whereas the others do everything else. A good team work should make use of skill sets available in such a way that there is a good distribution of effort. A team member is expected to be able to take up the work of another team member or support him or her where needed. Individuals who are team players or can demonstrate support to another team member (e.g., a CS student helping a CIT student to learn Android Studio) will be positively assessed through the individual assessment component, i.e., Individual Contribution Report.

Individual, self-learning. Lectures are intended to provide you with pointers information as a basis for you to prop further. You are expected to schedule in time on a weekly basis for yourself in order to learn tools that may be new to you, e.g., Android Studio, GIT repository, Latex, Doxygen (or another documentation tool such as JavaDoc), and/or Mark Down. The new skill sets that you will have acquired should be reported in both the Individual Weekly report (formative assessment) and the Individual Contribution Report (summative assessment).

> CS students who have taken COM1032 are expected to demonstrate learning beyond what was taught in COM1032. CIT students who have not taken COM1032 are expected to acquire the basic knowledge in COM1032. They are expected to learn from their CS peers[1].

[1] There is an individual unit of assessment, that is, "Individual Contribution Report", which has a weight of 20% of the entire module that specifically assesses a student's contributions with *respect to his/her baseline*. Thus, a CS student who has acquired COM1032 is expected to demonstrate that he/she has acquired new skills

Technical development. All students are expected to contribute to the technical development, which includes software writing or programming. You should document all your individual contributions on a weekly basis and then toward the end of the semester, use them to write your Individual Report.

Learning new skill sets. Besides the 'technical' aspect of the project we also expect you to demonstrate ability in project management and documentation, team working, and ability to work on your own. The intended learning outcomes are described in the next section.

2. Intended learning outcomes

Consistent with the high-level learning outcomes as described in the module description, the following is a list of more specific learning objectives that are related to a software engineering development lifecycle and project management.

Software requirements:

- To be able to identify software requirements agreed with the end user.
- To be able to negotiate inevitable changes and manage 'scope creep' effectively.

Design document / Functional Specification:

- To be able to define what is to be delivered and how different components are interfaced.
- To be able to document the design document professionally.

Project management

- To be able to devise a realistic project schedule which includes an appropriate task breakdown structure completed with milestones and deadlines.
- To be able to identify risks and devise contingency plans necessary to contain them.
- To be able to demonstrate and measure project progress by tracking them.
- To be able to manage change in case of slippage in schedule or early completion of tasks; and to document these changes.
- To be able to work as a team effectively, including clear division of responsibility and accountability, both at the individual as well as at the group level.
- To be able to manage change in project scope and to deal with this change effectively.

Test Plan

- To be able to develop test criteria that fulfill the customer's requirements.
- To be able to carry out the test rigorously following the established criteria.

Delivery of software

- To be able to develop application-layer software, ensuring that the final software works seamlessly.
- To be able to use the sensors from a smartphone/tablet effectively, e.g., camera, GPS, gyroscope, and accelerometer (where relevant).
- To be able to develop the required low-level software interface with the hardware components (where relevant).
- To be able to use additional services such as web API, Google Map, effectively (where relevant).

beyond COM1032 whereas a CIT student is expected to acquire the basics to the level equivalent to COM1032. The assessment is designed to be fair.

3. Requirements of the project

Your project that is intended to deliver a piece of software app, must fulfil a number of requirements, namely, the needs of the end user (user requirements), technical complexities, user experience, and security requirements, and organisation requirements.

In the sections below, requirements that are marked as mandatory (M), must be implemented. For those that are not mandatory, you may add your own requirements. The suitability of these requirements must be discussed and agreed with your Project Sponsor.

User requirements

The software must address the problem it states it would. You must identify at least two user requirements.

For example, based on recommendations in http://www.everydayhealth.com/longevity/mental-fitness/brain-exercises-for-memory.aspx, you have decided to develop a software app about brain training that evaluates the patient's ability to work out some calculation. So, the requirements might be:

U1	The app enables the user to do some basic calculations in their head; and then verify the results
U2	The app has several levels of difficulties
U3	The app can track the progress of the player

You have to (1) justify the origin of the requirements by doing some research into the related theme; (2) justify why the requirements fit the theme; and (3) ultimately, assess whether or not the user requirements are fulfilled.

Since the app is intended to solve the end-user's problem, you should seek to co-create the solution with the end-user, wherever possible.

Technical requirements

General principles. The technical requirements are put in place so that all groups will develop apps with comparable technical complexity. However, due to potential differences in user requirements and to allow rooms for creativity, we shall only provide examples of technical requirements rather than making them all mandatory. The suitability of requirements for your project must be discussed with your project sponsor. The marking scheme is designed that groups that challenge themselves with high technical difficulties and be able to implement these requirements will receive higher marks.

- The software app to be developed must be installed and run on the Android tablet provided by the University. Submitting a software app that runs off a website and displays the information through a browser of the device would not be satisfy this requirement.
- The app must make use of at least one of the many sensor functionalities available to the device, such as camera, GPS, gyroscope, or accelerometer.

The requirements are listed below. Note that T1 and T2 are mandatory

T1 (M)	Data generated from the app must be able to be transferred to a web server when the WIFI Internet connection is available, or becomes available (i.e., if the connection is not available, the app may wait until connectivity is resumed to make the transfer)
T2 (M)	The app uses the sensors from the device, e.g., camera, GPS, gyroscope, or accelerometer.

T3	The app uses additional services or modules such as web API, QR Code reader, or Google Map
T4	The app uses Bluetooth to share information between devices

For T2, if your app uses the camera, GPS and gyroscope, your group is considered to fulfil three technical requirements.

User experience

General principles. Usability of a software app cannot be overly emphasized. Your app must be easy to use, does not drain the phone resources (e.g., battery, memory, speed), or causes any inconvenience to the user.

U1 (M)	The user must be able to pause, resume and end the app at any point in time, for example due to incoming calls, battery low messages, etc.
U2 (M)	The user must be able to consult a help screen anytime
U3 (M)	The app has useful and appropriate errors messages that guide the user. Informative error messages must, for example, be displayed when there is incomplete or wrong user input.
U4 (M)	The app can remember the preference settings of the user and other states (e.g., if this is a game, the app must be able to show historical progression of the high score)

Legal, Ethical, Social and Professional requirements

General principles. The legal, ethical, social and professional requirements of the project cover both technical and organisational (group) requirements related to: 1) compliance with regulations, standards and certification processes; 2) the management and enforcement of Intellectual Property Rights (including copyright and licensing issues); 3) the management of contractual obligations (including terms of delivery and liability); 4) the consideration of wider ethical and social issues that may arise in and around a particular project; and 5) ethical and professional issues related to the use of publicly available documents and code.

Compliance

On issues of compliance, we focus in this section on regulations, standards and certification processes related to data privacy and security, leaving general issues related to good project management and quality assurance to the organisational requirements. In particular, we will look for evidence that data privacy issues were considered and taken into account from the design phase ("privacy by design"), rather than after the application was implemented. Address 2 of the requirements below with respect to a recognised national or international standard, or regulatory framework.

S1 (M)	2 information risks related to the application's functionality are identified and addressed by the application's design and/or its implementation
S2	Identify and mitigate 1 risk related to the privacy of data when stored, displayed or handled locally
S3	Show evidence that good Information Security practices were put in place (e.g. through the definition of roles and access control rules within the organisation or group, and the appropriate distribution of roles to group members)

IPR Management

On issues of Intellectual Property Rights (IPR) management, we will focus on software licensing in terms of *using* externally licensed libraries, hardware or software, and in terms of *choosing* the right

licensing terms for an application, depending on the licensing terms of its dependencies and other external constraints.

L1 (M)	Choose appropriate licensing terms for the various components of your application
L2 (M)	Properly reference and acknowledge external libraries and resources used
L3	Provide evidence that your project and its delivery comply with its and its dependencies' licensing terms

Contractual Terms and Obligations

Requirements in this section cover the drafting of contracts for commissioned software development projects in some scenarios.

C1 (M)	Identify, in a use case, a customer who could have commissioned the development of your project, and draft *reasonable* contract terms
C2	Provide evidence that your development strategy is appropriate for the chosen contract terms

Ethical and Social Issues

Requirements in this section cover discussions related to the potential impact of the software project on society and individuals. Examples include the use of targeted advertising to support free applications; the sharing (voluntary or accidental) of personal information; issues related to legal data collection obligations; the potential effect of your app on its users' behaviour; or the use of collected data for medical research.

E1 (M)	Reflect on at least two of the ethical/social aspects of your project
E2	Justify some of your design, implementation or legal choices in light of these ethical and social aspects

Referencing and "Public Domain" Resources

The requirements in this section cover issues related to the use of "public domain" resources, their proper referencing, and ethical (and legal and professional) issues surrounding their use. We note that these requirements certainly overlap with avoiding Academic Misconduct proceedings. By making them explicit here, we emphasize that the projects are being assessed also on the "Good Citizenship" exhibited by its members. Note also that R1 overlaps with L2. However, R1 should also include non-code references (textbooks or documentation), and resources that were consulted during the development but are not used in its final product.

R1 (M)	List all resources and code used or referred to during project development
R2	Outline your organisation or group's strategy for "giving back" to the relevant development communities. This could take the form of a list of bug reports, patches or pull requests made to upstream libraries, or a *reasonable* strategy for donations to open source projects you relied on during the development of your application.

Organisational requirements

O1 (M)	Adhere to a clear and consistent coding style. That will make it easier for yourself, group mates, other groups and supervisors to understand you code.
O2 (M)	The project must be a standard Android SDK project. The project (and its Javadoc documentation) must compile in the SDK when checked out from GIT. The game must compile and run in SDK and the emulator. It must finally run on the provided tablet, currently a Samsung GT-P3110
O3 (M)	Write documentation of the code using Javadoc or similar software
O4 (M)	Use of the concurrent versioning system GIT is compulsory for code and all project documentation. Commit log messages must be significant and useful.

	(avoid putting binary files on GIT; put them on SurreyLearn)
O5 (M)	Minutes, Agendas and the other documentation must be kept on the GIT server and written in MARKDOWN format.
O6 (M)	4 project risks are identified

4. Delivery pattern

Breakdown of effort

The delivery of the module consists of a series of lectures, weekly supervised group meetings, unsupervised group meetings and individual work, according to the schedule below.

Table 1: Breakdown of effort

Topic	**Week number**	**Duration (hours)**	**Total effort (hours)**
Introductory Lecture	Week 1	2 hours (in total)	2
Security Lectures I & II	Weeks 1 & 3	3 hours (in total)	3
Project Management Lectures	Weeks 2,4, 5	3 hours x 3 sessions	9
Team working	Week 1	2 hour (in one session)	2
Technical Writing	Week 3	2 hour (in one session)	2
Supervised Group Meetings	Weeks 2–10	25 min weekly	3 ¾
Unsupervised Group Meetings	Weeks 1–11	3 hours weekly	33
Individual Work	Weeks 1–12	7 hours 20 mins weekly	88
End-user requirements (Q&A session)	Weeks 3, 5, 7 and 9	1 hour x 4 sessions	4
Software testing	Week 12	3 hours (in one session)	3
		Total	Approx. 150 hours

In total, you are expected to spend an average of 150 hours over the semester, which is commensurate with the 15 credits gained in this module.

We discuss the programme structure in greater details below.

Programme structure

Supervised group (progress) meetings

Each group will meet with their project manager, who is one of the module lecturers, in order to discuss the progress of the project and get advice.

The purpose of supervised group meetings is so that you provide updates at a weekly interval to your project manager who will give you advice and guidance on the technical and managerial aspects of your group project. In addition, since he will also mark your interim and final audits (or assess the performance of the team member in the real world, such as during an annual appraisal), engaging with him regularly is of absolute importance.

Note that the group manager is not the chair of the supervised meeting, but merely a participant. Group members are expected to drive these meetings, taking turns in roles of chair and minute-taker.

Supervised group meetings are scheduled weekly from weeks 2 to 10 and last 25 minutes for each meeting as listed in Table 1.

The supervised progress meetings are compulsory and **all group members must be present**. Since project sponsors are very busy and manage a lot of projects (as is also common in professional life), you need to prepare for these meetings well and decide the agenda in advance, which essentially cover the topics you want to discuss.

More specifically, the following needs to be available **at beginning of the progress meeting**, both in printed form – ensuring that one copy is handed to the project manager – and the original is pushed onto your GIT repository:

1. an agenda for the current supervised progress meeting
2. minutes of the previous supervised progress meeting
3. a report on the previous week's external lectures (if any)
4. the current version of the design document (only weeks 1–5)

Late submission of the above items will lead to the standard lateness penalty. Therefore, make sure that these items are complete *before* your supervised meeting slot.

Unsupervised group meetings

Purpose. Each group is expected to meet weekly with the aim to discuss and take decisions related to the project, such as distributing and coordinating tasks, evaluating task outcomes, and resolving any technical or managerial issues.

Meeting time/location. In order to ensure that meeting arrangements are acceptable to all members, unsupervised meetings are to take place on Campus. You are advised to plan ahead any extracurricular activity. Once a time is agreed, all members are expected to turn up punctually.

Remember that most of you are registered as full-time students and hence ought to be available at standard office hours throughout the week.

Attendance. Attendance at the unsupervised meetings is compulsory; and any apologies need to be communicated in advance of the meeting to all group members.

What to do in each meeting. In the three hours of weekly unsupervised meeting , group members take turns chairing these meetings, which includes setting the agenda and timekeeping.

You are supposed to meet up *face to face* and practice to practice what is normally considered a "daily scrum" session by answering three questions[2]:

- What did you do since the last meeting?
- What will you do today (and onward)?
- Are there any impediments in your way?

Use this session to seek helps and discuss problems.

[2] Read about daily scrum here: https://www.mountaingoatsoftware.com/agile/scrum/meetings/daily-scrum. Although we can't do a daily scrum due to the nature of the module, students are expected to get familiarized with Scrum as an important software development methodology (Read more here: https://www.mountaingoatsoftware.com/agile/scrum)

Individual work

Individual work is expected to be about 7 hours 20 minutes per week on average for each student. Use this time to learn a range of tools in your own time and at your own pace.

Lecture sessions

There are five external lectures in this module, namely three project management lectures and one technical writing lecture, as shown in Table 2.

Table 2: Lecture series

Week	Guest lecturer	Day/time	Topic
1	Russ Lisk (Library)	TUE, 11am-1pm	Working in a group
1	Norman Poh/Francois Dupressoir	THURS, 3-6pm	Welcoming & lecture on security (I)
2	Gaz Bloxsidge (IBM)	THURS, 3-6pm	Introduction to project management
3	Moray Bennet (Library)	THURS, 3-5pm	Technical writing
3	Francois Dupressoir	THURS, 5-6pm	Lecture on security (II)
4	Gaz Bloxsidge (IBM)	THURS, 3-6pm	Work breakdown structure & schedule
5	Gaz Bloxsidge (IBM)	THURS, 3-6pm	Risk, quality, change control

Note: Make sure you check the lecture time and location from your timetable by logging onto https://timetable.surrey.ac.uk.

Attendance to all lecture series is compulsory. For this reason, your attendance will be recorded. If you can't make it, please send an email to Norman to explain the reason for your absence.

Project management lecture series

The objective of the project management lecture series is to equip you with the skills required to manage projects including software projects. These lectures are delivered by experienced managers in industry, and are meant to provide 'real-world' perspective.

Lectures given by the Central Library

There are two talks – (1) working in a group and (2) technical writing. The aim of the first lecture is to ensure that you will be able to

- Develop your group working skills*
- Identify your existing skills and areas for development in terms of group working
- Participate in an activity to illustrate how we can work better in groups
- Identify your natural role/s in group work situations
- Become more aware of the issues that surround group work and how to mitigate them
- Compare the characteristics of two written management plans and conclude which is best and why

The aim of the second lecture is to enable you to communicate your ideas effectively through writing, which is useful to help you write the design document. Specifically, you will be able to:

- Recognise the key components of meeting minutes
- Consider one means for writing meaningful descriptions

- Use some sound structural characteristics of technical writing to assess the structural quality of any written documents e.g., the design document, the user test manual, and your individual report.

End-user requirements and Q&A session

Last but not least we have put in 4 session of 1-hour meeting to allow students to ask questions related to end-user requirements, choice of software library, and to advise students on final audit report and marking scheme. These meetings will take place in Weeks 3, 5, 7 and 9.

Timetable

To ensure that you are aware of all the scheduled lecture and meetings, refer to "Module timetable (updated as of 6 Feb 2017)".

5. Working as a team

This section covers topics that establish the best practices to work as a team. These include:

- How to run the weekly meeting?
- What should be included in the agenda?
- What should be included in the minutes?
- What you should write in your individual weekly contribution report?
- How you should communicate with each other?
- I am stuck with a technical problem. How do I get help?

This Guidebook is not intended to be comprehensive but provides the basis toward working together. Please refer to the external lectures on project management for a more comprehensive guide.

In the following, we elaborate on the meeting agenda, minutes, and individual weekly report, as well as briefly discuss the three project phases, namely, design, implementation and testing.

Agenda

In general, group progress meetings will have the following agenda:

1. Date, Time, Place of meeting.
2. Reading and approval of minutes from previous meeting (except, of course, for the first meeting, where there are no previous minutes to approve).
3. Any reports on external lectures (if there was in the week before the meeting).
4. Progress report – the group will be expecting each group member to detail progress in their responsibility area (and submit this weekly as their contribution report (see template in folder documentation\contribution.) , and may set targets for the following week with individual progress reports.
5. Any set items required for a particular week or period by this guidebook or your manager.
6. Any other set items the group wishes to discuss.
7. Group supervisor business (this will for example be used to give feedback, and about 5min should be reserved for it in the meeting)
8. Matters arising.
9. Attachments to the agenda are:
    a. Reports on external lectures of the preceding week (if any). **This report should state who was present in or absent from the lecture session.**
    b. Individual contributing reports of all group members.

Electronic submission to the group's repository is sufficient for these attachments. No hardcopy of attachments needed.

Agendas shall be produced and circulated to the group members and uploaded to the GIT repository in advance of the meeting. A printed copy shall be given to the project manager (and optionally to the group members) at the beginning of the meeting who keeps it and adds it to your group portfolio.

Agenda items shall be numbered for easy reference in the minutes in the Mark-down format.

Although it is up to the group to set the agenda for the supervised meetings, your project manager may ask you to take up specific issues and put them on the agenda from time to time.

Minutes

Since minutes are a written summary of a previously held meeting, their structure should reflect well the order of the agenda. More importantly, any decision and actions should be recorded clearly. The structure of minutes should include at least the following items:

- Date, time, place and duration of meeting.
- List of attendees and apologies (if any) for the absent members.
- Approval of the minutes of the previous meeting.
- Summary of discussion about and conclusion for items on the agenda.
- Actions for each member of the group with deadline. Actions needs to be numbered in form x.y where x is the number of the week in which the meeting is held and y the running number of that action in that week. This makes it easier to refer back to actions from previous meetings for follow-up.
- Follow-up of agreed actions the last meeting.

It is common practice to assign new actions to a member only when that member is present. This is so that the member with the new action can have their say.

At each supervised group meeting, printed minutes of the previous supervised meeting need to be distributed to the project manager who will keep a copy in your group portfolio. Although it is not necessary to print minutes for the group members, it is expected that all group members should be able to access the minutes electronically during the meeting, if necessary.

Individual weekly contribution report

The purpose of this document is to enable each member records his/her weekly contribution to the group. It is meant to be brief, i.e., no more than five lines, and serves as an aide-memoire for you to write your final contribution report, which is formally assessed.

In addition, the report may be used as part of the supporting documentation in support of group scaling. Therefore all group members should read each other's individual report carefully on a weekly basis.

An individual report should address the following three issues:

- A list of main activities, tasks and assignments undertaken along with a note on the progress or the state associated with each of these activities;
- A list of your three most important commits/pushes (if any) to the GIT repository with git identifier and commit description/message.

This needs to be done on a weekly basis and each student's contribution needs to be submitted in Mark-down format onto the group's GIT repository before the next supervised meeting.

The instructions for writing the Final Individual Report (at the end of the module) will be discussed in a separate document entitled "Module Assessment".

Group Communications

Email communication

Important group communication shall take place using the members' University email accounts, and hence all members are expected to read and answer their emails **at least once each working day**. As a practice, you should respond to email requests within 24 hours for a working day.

> This is to prevent things like "Oh, I didn't know when the group meeting was because I never got your text." or "I did send you my apologies before the meeting, but my text message must have got lost." It also serves as evidence that communication has taken place in time, including all members, should a disagreement arise.

Files shall be shared exclusively using the GIT repository. This includes all source code, agendas, minutes, the design document etc. It is not acceptable that one member uploads files to GIT on behalf of a different member, or files being shared by email or any other means, as this cannot be audited.

The Appendix "GIT structure and " contains further instructions on how the GIT repository is organised.

Use of social media for communication

In addition to using your University's email account, you can use Facebook or Whatsapp, or any social media for communication. Such platforms enable you to discuss ideas outside the regular unsupervised group meeting or when members of the group work remotely.

Social platforms cannot and should not replace any face-to-face weekly unsupervised meetings.

Asking Questions

If you have a question regarding this module, we would prefer you to write or ask your group's project manager. There are several occasions where you can ask questions.

Questions related to the end-user requirements. For questions related to the end-user requirements, below are your opportunities to ask these questions.

- During the weekly supervised meeting with your project sponsor
- The Q&A sessions (currently scheduled in Weeks 3, 5, 7 and 9)
- If you are unable to ask questions during the above occasion, then, as a last resort, send an email with the heading 'COM2027 - Group XX' to your project sponsor.

General questions. Questions of general interest or those that can easily be answered within the community (i.e. by all students in this class) can be posted on the discussion forum on SurreyLearn. This could include questions about how to develop for Android on the Mac, general GIT questions, setting up Android Studio, etc. We encourage students to post questions there and to answer

questions there as well. This developers' forum worked well in previous years to exchange best practice. Lecturers will visit this forum occasionally.

> Go to SurreyLearn > COM2027> Tools > Discussions > General Questions to ask questions.

6. The Three project phases

Design phase

The first phase of your project which starts immediately is the design phase. In this phase you have to get an overview of Android Studio, decide on your app concept or problem to solve, and determine whether it is feasible to implement within Android Studio.

In order to deliver a good project you must know all the required tools well; so you are likely to try these tools in order to assess how hard or easy it is that you can implement your project concept.

Taking into account the requirements as stated in "Section 3. Requirements of the project", you shall develop a system design to guide the implementation. The design document (which forms part of the project documentation) would contain only a high-level description of your project during the first few weeks of the project, but it should be detailed enough to guide your implementation. For example, you should be able to identify which SDK components need to be used (and therefore learnt).

As implementation proceeds, certain design decisions will have to be refined and/or revised as your experience with the Android SDK grows. As a result, your design document will be continuously updated and this information is captured through the weekly supervised meetings (along with agendas and minutes).

During the design phase, you will produce an architectural design of your application as well as the infrastructure code necessary to interface your game with the Android SDK.

Good software engineering practices are discussed in (Pressman 2005, Sommerville 2011). The references of these books are provided in the Appendix.

Implementation Phase

Once the SDK components have been identified, you should already explore how their associated codes work by testing them. A good starting point is to follow a tutorial.

To deliver a good design, it is essential that as a group you have become acquainted with the tools you need to use and with Android Studio. It is particularly important that you understand how the infrastructure code works, i.e. the code that has to be there for any Android application to interface with the underlying OS of the mobile phone/tablet.

Do adhere to good coding standards. JavaDoc is essential for other group members and yourself to understand your code. Both complete and meaningful JavaDoc comments and appropriate single-line comments to document small-scale implementation issues are a requirement for the project documentation.

Test Phase

At the end of the implementation phase, do not forget to test your software application; and check it against the requirements. Also in good time before submission (we recommend to do so before the end of week 9), test the packaging and running your game as required for the final submission.

7. Module Assessment

Unit weighting and submission deadlines

The software engineering project is a group project. Therefore, assessment will be based on your group as a whole.

This coursework represents **100%** of the assessment for this module, with the following breakdown by assessment unit:

Table 3: Submission deadlines for different assessment units

Assessment	Unit weighting	Group or individual submission	Submission due	What to submit (in the GIT repository)	Scaling applied?
Interim Audit	Formative	Group	Week 5 (T)	Interim audit document	None
Final Audit (CW1)	40%	Group	Week 11 (M)	Project design document, your software package, User manual	Group scaling
User Acceptance Test (CW2)	40%	Group	Week 12 (T)	User Acceptance Test form	Instructor scaling
Contribution report (CW3)	20%	Individual	Week 12 (T)	Individual report	None

Note: M: Monday, T: Tuesday. The deadline is 4pm.

> Each submission must be done through SurreyLearn before the deadline. Your Project Sponsor will check the timing of the submitted item. The University's standard late submission penalties will apply.

The final audit assessment unit (40%) is subject to a peer-moderated marking scheme known as "group scaling". This is designed to take into account the effort of each group member. The details are further described in the Section "Group scaling for the Final Audit".

The User Acceptance Test (which is worth 40%) is subject to instructor scaling in order to ensure that marks are fair and appropriate. Read the Section "How the User Acceptance Test is marked?" to understand how the mark is calculated.

In the following sections, we briefly touch upon the four units of assessment.

Interim audit

The purpose of the interim audit is two-fold. First it enables you to document early on in the project what you will deliver and how you will go about doing this. Second, you will benefit from informal feedback from the instructors on areas that you can improve. This is so that you can take the feedback into account so that you are ready for the final audit (which is assessed; see the table above).

Your abstract should address the following aspects:

1. The description of the proposed product, which defines the project scope
2. A list of software requirements and specifications;
3. Important diagrams describing your software (see the box below);
4. Draft schedule and partition of work.

Refer to the document "COM2027 Software Engineering Assessment Criteria" for detailed guidance.

Important diagrams to be included. There should be a sufficient explanation accompanying each diagram so that you can communicate your ideas clearly to your Project Sponsor and to other team members through this document.

As a minimum, you should include **two** structure diagrams and **two** behavioral diagrams. For example:

- one structure diagram illustrating the overall system architecture
- one structure diagram such as the *class diagram* illustrating how you have used object-oriented principles
- one behavioral diagram, such as use-case diagram (particularly important for software applications other than games)
- one behavioral diagram such as a sequence diagram or a state machine.

Make sure that you choose only relevant diagrams to illustrate your ideas.

Further reading

- For a check list of what is needed for software requirements, refer to https://en.wikipedia.org/wiki/Software_requirements_specification.
- For a check list of software design, refer to https://en.wikipedia.org/wiki/Software_design_description.
- For diagrams to include, consider https://en.wikibooks.org/wiki/Introduction_to_Software_Engineering/UML

Your interim audit document acts as an early version of the final audit document, the assessment of which is described next.

Final Audit

The purpose of the final audit is to allow the Project Sponsor to assess the *process* you have followed in order to deliver the software. Although the structure for the final audit is similar to the interim audit, rigorous documentation will provide evidence that the learning outcomes of this module have been achieved.

The design document is a key part of this audit, which addresses four criteria. You should make sure that your design document fully satisfies each of them. These five criteria build on the interim audit and are:

1. Problem description and proposed solution (scope) – 20 marks

2. Software requirements – 20 marks
3. Quality of the design document – 20 marks
4. Quality of the source code documentation (naming convention, code structure and comment) – 20 marks
5. Legal, Ethical, Social and Professional issues – 20 marks
6. Project management – 20 marks

This gives a total of 120 marks for the Final Audit which is then renormalized to 100% and is given a weight of 0.4 for the entire module.

It is important to create a design document at an early stage and update this throughout the project. This is because the purpose of this document is to communicate the project idea and stages unambiguously among your team members. In addition, you also use this document to communicate with your Project Sponsor who may not necessarily have the technical background or the domain knowledge sufficient to understand your code.

Problem description and proposed solution (scope) – 20 marks. This section depends on whether you are developing a piece of software application OR a game.

- Software application: Explain the nature of the problem that your software addressed, and how you solve it. Justify why your software is important, unique, and/or potentially 'market leading'.
- Game idea: Explain how your game will be engaging and fun to play. You do so by explaining the overall game strategy and the elements that you have considered at the design phase.

Software requirements – 20 marks. Both functional and non-functional (technical) requirements should be clearly specified. For games, refer to the module guidebook; whereas for other software applications, refer to the criteria in the user acceptance test (section 4) for further explanation[3].

Prioritize your requirements so that you know which features are mandatory and which features are desirable.

Quality of coding and design documentation. In principal, good design and software documents allow a new team to take up a project with similar understanding to the current team, with minimal communication or support.

There are two types of documentation; one referring to the source code and another referring to your design document.

> **Design document – 20 marks.** The design document refers to a high-level architecture of the system that is intended to satisfy all the requirements. This document should be an expansion of the design from the Interim Audit document, which is continuously updated until the end of the project.
>
> **Source code document – 20 marks.** The source code document refers to any document explaining the software deliverable in the broad sense; and this includes not only the source

[3] While the requirements are clearly stated for the game, for a software application it is reasonable to expect that the functional and technical requirements will be different for different applications. For this reason, you are expected to work closely with your Project Sponsor in order to ensure that the software your group produces has a level of complexity that is expected of this module.

code but also any specifications you have entered in order for a piece of software to generate template code for you.

It is recommended that you adhere to the following good practices:

- Ensure that there is a good introduction to the design, documenting the system architecture (e.g., class structures and their purposes, complete with their associated properties and methods)[4].
- Make sure that you adopt a convention of coding and use this convention throughout the entire project. Either refer to and use coding guidelines from elsewhere, or write a short description of the convention applied, and make sure that everyone in your team follows the same convention.
- Use automatic tools to assist the documentation of source code, for example, Javadoc, and identify which tools you have used.

Legal, Ethical, Social and Professional issues – 20 marks. 80% of these marks cover the Mandatory requirements S1, L1, L2, C1, R1, and E1. 20% of these marks cover optional requirements, as illustrated in the relevant section, or discussed with the project sponsor during the supervised meetings. The breakdown of these 20 marks is as follows.

- **Compliance (10 marks):** *S1 – 8 marks;* S2, S3 – 2 marks
- **Licensing (3 marks):** *L1 – 1 mark; L2 – 2 marks;* L3 – 1 mark
- **Contract (3 marks):** *C1 – 2 marks;* C2 – 1 mark
- **Ethics and Society (2 marks):** *E1 – 2 marks;* E2 – 1 mark
- **Good citizenship (2 marks):** *R1 – 2 marks;* R2 – 1 marks

Note: The total number of marks stated for each of the sections above cannot be exceeded, even if additional optional requirements are fulfilled. (For example, if all three Compliance requirements are fulfilled, only 10 marks are awarded rather than 8.) However, optional requirements can be used to "catch up" failings in the delivery or documentation of mandatory requirements.

Project management – 20 marks. Project management covers work breakdown structure (WBS), risk, scope creep, communication and schedule. You will learn about these in the timetabled lectures.

If there is a change in the scope of the project, document the impact of this change is and how you have managed to accommodate this change.

User acceptance

While the Final Audit assesses on the *process* of your project (e.g., software documentation and project management), the user acceptance test assesses your software *product*. Submission comprises the software itself, as well as subsequent User Acceptance Test (UAT) evaluations for the software of 3 other groups.

How to submit your software package for the User Acceptance Test?

For the User Acceptance Test, you need to package a final, release version of your software.

[4] Make sure that you include this information in the design document and more detailed information in its Appendix (where applicable).

Submission. For submission, you will need to compile you program to its release version, thus generating an ".apk" file in the bin folder of your Android Studio project. The package file must be renamed to `group_XX.apk` where `XX` is your two digit group number (i.e. 01 for group 1)

The above installation file or URL text file must be committed in the folder `final_submission` in your GIT repository by Week 11, ensuring that you submit before the published deadline which is either a Monday or a Tuesday, as shown in "Table 3: Submission deadlines for different assessment units".

Instructions for testing. For games, note that the only way to tell potential users how to play the game is by using the help screens in your game. Similarly, for software other than games, appropriate help screens will be needed.

Peer assessment. After the submission of your final software package by Week 11, the package will be distributed to three other groups who are asked to evaluate it. In addition, your work will also be reviewed by the module instructors.

Reviewer group versus reviewee group. When you review the other group's work, your group acts as the *reviewer group*. When your group's software product is reviewed by others, you are a *reviewee group*.

The reviewer group members must act together in the review, with attendance of reviewee members. So, make sure that you agree in advance the day to meet up as a group in order to test the three pieces of software (or game). This happens in Week 11. Each group member should report this in their Individual Contribution Report (to be discussed in the subsequent section).

How to perform the user acceptance test as a reviewer group?

After the deadline (refer to Table 3), you will find three installation files on

"SurreyLearn > COM2027 > Content > Software apps for User Acceptance Test "

The submission will have file names such as `group_XX.apk` and `user_acceptance_test`. You are to test three apps by installing the software using the provided Android Tablet.

You will then fill in a User Acceptance Test form which contains a list of criteria as shown in Table 4. For each of these criteria, you will provide a justification of how you arrive at the score.

The quality of your of your review, which is determined by the Module Instructor will be used to scale your mark as a reviewee group. This instructor scaling is described later.

Evaluation will be carried out against each of the following criteria, weighted for importance with **Feedback** provided for each of the criteria which will offer justification for the score and suggests how you could have improved your product.

Table 4: User acceptance criteria

Commented [NP1]: To revise

Criteria	Importance
Ease of use: The software is very easy to use. It is extremely streamlined with all functionalities easily accessible.	3
Help screen: I can access the help screen at any time and the information provided is relevant and useful	1
Resource consumption: The software uses only a reasonable amount of memory/space. An indication of this is that the software is responsive so, for	1

example, the user can switch to a different application and then come back to it; and it is still working.	
Glitches and bugs: The software runs smoothly; it does not crash or freeze.	1
Connectivity: The software interacts with a server or another copy of the same software, e.g., via peer-to-peer connection. (A peer-to-peer connection is obviously more technically demanding and should be rewarded with a higher score.)	3
Error handling: The software displays useful error messages when presented with wrong input	1
Persistence: The software 'remembers' the user's last setting, preference, or state (e.g., high scores for games). For games, the user can pause and resume the software at any time and then pick up from there.	3
Interoperability: The software works on a mobile and a tablet Android device	2
Functionality: The software does what it is designed for. A list of requirements should be provided by the reviewee group.	5

Note that Importance is not weighted equally. For example, the functionality of software is given an importance value of 5 out of the total of 20. This is because software functionality is deemed the most important factor in the user acceptance test.

Instead of using grade descriptors, a Likert scale (Carifio and Perla 2007) is adopted due to its ease of use. For each of the criteria shown in Table 4, the following Likert scale is used, ranging from strongly disagree to strongly agree, each of which is mapped to a mark from 0 to 4, respectively.

Likert Scale				
0-2	2-4	4-6	6-8	8-10
Strongly disagree	**Disagree**	**Undecided**	**Agree**	**Strongly agree**

Table 5: User-acceptance test

How the User Acceptance Test is marked?

Calculation. The total mark for this unit of assessment is 200 (which is then weighted to 40% for the overall module). It is calculated by taking the weighted sum between the importance of each criterion and its associated rating.

Instructor scaling. While the review you give to other groups will feed into their final mark, the *quality* of your review will also become part of your group's assessment; and this is done through an "instructor scaling" scheme. This scheme serves two purposes. First, the quality and accuracy of your feedback as reviewers yields a score, which is used to scale your raw mark. Second, reviews that are unrealistic or inaccurate (and subsequently also receive a low score) will be excluded from the reviewee's raw mark. We will elaborate next how this calculation is done in the next few sections.

Example. If a group receives the worst Likert rating (strongly disagreed) across all user-acceptance criteria, it will receive a total of 0 points out of 80. Conversely, if a group receives the best possible rating (strongly agreed) across all the criteria, it will receive 200 out of 200.

An Excel sheet with the name `CW3_user_acceptance.xlsx` containing the above form is available on SurreyLearn. This spreadsheet will do the calculation for you.

Calculating the mark received by a reviewee group in conjunction with instructor scaling

Averaging. Your group, as a reviewee group, will receive three reviews. Your raw mark for your project will be based on these reviews, *after* excluding unrealistic and inaccurate ones. The average of these marks – up to three marks, if they are all valid – is your *raw* mark for the User Acceptance Test.

Instructor scaling. Your Module Instructor will assign a score between 0.0 and 1.0 that reflects the quality of your review of three other groups' projects. Let's refer to this score as instructor's coefficient.

Your group's final mark is simply the product of the raw score and the instructor's coefficient.

Example. Assume the following: Your group produced an excellent piece of software that is useful (or fun to play for games) and meets all requirements. Your reviewers (groups A, B and C) all realise this and state this in their review form, and accordingly you get a very good raw mark for your project of, say, 70 points (out of 80). However as reviewers of the projects of groups X, Y and Z you do the following:

> **Group X** You are a bit lazy and fairly lenient and write a positive review, so you miss to realise that group X's project doesn't run under some emulator settings and crashes when there is an incoming call. Furthermore there are no comments that state your reasons for your rating of group X's project.
>
> The lecturers check your review and see that your review has missed severe flaws of group X's project and therefore is inaccurately positive. Your review is ignored for forming group X's raw mark. And you yourself receive a low score of, say, 0.5 for this review.
>
> **Group Y** You think your own project is better than this project and this is reflected in your review. However the other reviewers of group Y (and the lecturers) think group Y have done a good job and your criticism is not justified. As a consequence you receive a score of 0.6 for this review and your review is ignored for forming group Y's raw mark.
>
> **Group Z** Your review of this group is largely correct, however you missed to realise that group Z's application doesn't have a help screen in one of its ten forms (while it does for all other forms). The lecturers judge this is only a minor omission on your side and therefore you get a score of 1.0 for this review.
>
> In total, you have an average score of (0.5 + 0.6 + 1.0)/3 = 0.7. Hence your raw mark of 70/80 will be scaled to 0.7 × 70/80 = 61.25%. We assume that reviewers will make a good job in general, and therefore that 1.0 will be the most likely score. Scores lower than 0.8 are normally given only when reviews have to be excluded to contribute to the reviewee's raw mark because of major omissions, severe inaccuracy or simply lack of engagement.

Individual contribution report

Description

The purpose of this individual report is to enable students to reflect upon their individual contribution to the group and what he/she has learnt from both the process of working in a team and carrying out the project.

Mind-set in writing an individual report. Think of this exercise as a project post-mortem. This is a process for evaluating the success (or failure) of a project's ability to meet software requirements.

You can also take the exercise as an individual performance appraisal. A performance appraisal, also referred to as a performance review, performance evaluation, (career) development discussion, or employee appraisal is a method by which the job performance of an employee is documented and evaluated.

Following a performance appraisal in the real world, a Project Sponsor may decide to use the member for another project or not. If a team member is hired through a short-term consultancy contract, this could mean that the member may or may not be subcontracted to work in another project, depending on his/her performance.

Sections. Using the above mind-set to write your individual report, you are requested to address the following issues:

1. **Individual contribution**: What contributions have you made to the group? Put differently, without your contribution, would the project output be similar or slightly worse off than it currently is? And how?
2. **Self-improvement:** What new skill sets have you acquired through this process?
3. **Process improvement**: If you are to participate in another project that is similar to this one, what aspects will you like to change in order to make it better?

For each of the points above, you should provide as much evidence as possible. Evidence must be tangible. Consider the following aspects: number of bugs found and handled, the timeliness of module delivery, the number of suggestions made, or the amount of effort spent on writing the design document, minutes, and other forms of documentation.

Criteria	**Feedback**	**Rating: Poor, Weak, Acceptable, Very good, Excellent**
Individual contribution: What contributions have you made to the group		
Self-improvement: What new skill sets have you acquired through this process?		
Process improvement: If you were to do a similar project, what aspects would you like to change in order to make the project better?		

Grade descriptors for individual report

You can find the grade descriptors in the section “Grade descriptor for the Individual report”.

8. Marks calculation

What is my final mark?

Your final mark is calculated as follows:

$$CW1\times\left(\frac{8+s}{8}\right)\times 0.4 + CW2\times F\times 0.4 + CW3\times 0.2$$

Noting that CW1, CW2 and CW3 are in percentage. s is the factor for group scaling and F is the coefficient for instructor scaling.

Group scaling for the Final Audit

Purpose

The purpose of group-scaling is to encourage team spirit and create responsibility and accountability in teamwork. It is certainly not intended, and nor should it be used, to create competition among group members.

A group-scaling form can be found at the end of this document

In order to differentiate the contributions of each member in a group, we will use "group scaling". The contribution value listed for each group member indicates the individual contribution to the group work according to the following rules:

- A score of 0 indicates that the group member contributed to the group as expected, doing no more or no less than was agreed.
- A value less than 0 indicates that the group member did less work than expected, with a more negative number indicating less effort.
- A value greater than 0 indicates that the group member did more work than expected, with a more positive number indicating more effort.

Note that the sum of the individual contributions for the group members must be 0. This means that if one or more people have negative scores, then there must be one or more people having positive scores to balance this out and vice-versa. Each member's score will be used to scale the overall mark the group achieved for that member by fractions of ten.

$$Individual\ mark = Group\ mark \times \left(\frac{d+s}{d}\right)$$

You need to decide and debate on the value of the (severity) denominator, d. Currently, $d = 8$.

Example

The following examples are given on the basis that the final project audit is awarded 60 marks:

- A member gets a score of -2, then that member will get a final mark of 60×(8-2)/8=45.
- Another member gets a score of +2, then that member will get a final mark of 60×(8+2)/8=75
- All other project members will receive a mark of 60.

All marks are to be considered provisional and subject to approval by the Board of Examiners.

Constraints

1. The mark that is attributable to the group, after applying group-scaling, cannot be greater than 100. Example: if the group receives 85 and a member of the group gets a score of +2, then the final mark is not 85x(2+8)/8=106.25, but 100.
2. Similarly, if the final mark after applying group-scaling is below the pass mark, then the module coordinator reserves the right to judge whether the project member concerned should fail the unit of assessment or not.

Scale descriptor

The following table illustrates how the group-scale marks are given.

Group-scale mark	Descriptor with examples
0	A student is fully engaged in the group work
0	A student is fully engaged in the group work but has failed to deliver the task allocated to him/her initially. When support is sought, this has allowed the task to be executed. Rationale: Students who are "weaker" in the group should not be penalized for seeking help. On the contrary, learning from peers is one of the purposes of doing the coursework in a group. (Students who helps others will be rewarded based on evidence given in their Individual Contribution Report)
-1	A student fails to deliver a sub-task allocated to him but did not seek help from others. The rest of the group members believes that this failure has significantly impacted the group's result.
-2	A student has not been actively engaged with the group (e.g., not responding emails in a timely manner, e.g., within 24 working hours) and did not allow the group to have sufficient time to re-allocate the task assigned to him/her to someone else. As a result, the software lacks some important features that cause the group to get an overall a lower mark.
-2	A student only prepares some documentation work and has not delivered any coding or software deliverables in the general sense, e.g., classes, functions, XML resources, IT infrastructure, or graphic resources.
-3	A student "bosses around" and does not actually contribute to other software to deliver some of the results. The rationale is that simply giving instructions and not doing the actual work is not in line with the team spirit.
-3	A student fails to deliver any task given to him. Whenever a task is allocated, there is always an excuse not to complete the task. Effectively, there is no tangible contribution from the student.
-8	A student has failed to attend at least half of the group meeting and has not engaged with the rest of the group. The remaining students will share a maximum of 4 group-scaled mark (not 8) in equal proportion, unless agreed otherwise.
-8	A student drops the module halfway. As a consequence, the remaining students will share a maximum of 4 group-scaled mark (not 8) in equal proportion, unless agreed otherwise. The rationale for this is that the remaining students have to share the work load that has been allocated to the student mentioned above.

Grade descriptor for the Individual report

Your Project Sponsor will rate your individual contribution report and will check that what you have written is consistent with the weekly report. The general grade descriptor as shown in Table 6 is used.

Table 6: General grade descriptor

Mark range	Descriptor	Description
[70-100]	Excellent	Surpasses expectations; significantly better than requirements
[60-70]	Very Good	Meets all requirements; generally exceeds expected level
[50-60]	Acceptable	Generally meets expected level; some deficiencies may exist but none of major concern
[40-50]	Weak	Generally fails to meet expected level; significant deficiencies

[0-40]	Poor	Significantly below requirements; many deficiencies

The total points for this unit of assessment is 300, which is then rescaled to 20% for the final mark of the module.

9. Appendices

Software list

The software tools you need are:

Recommended software/tools	Functions	Where to get them
Android Studio	Programming environment	http://developer.android.com/sdk
GIT system	Version control	http://info.eps.surrey.ac.uk/IT/FAQ/gitlab-faq.php
LATEX	For the design document	http://miktex.org
Notepad ++	Editor	https://notepad-plus-plus.org
Markdown	For minutes, agenda, individual report (weekly and final)	https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet

Although the necessary software is installed on computers in FEPS UNIX labs, it is recommended that you install them on your PC/laptop.

Check list for submission and due date

Note that the cut-off deadline is *before* the meeting for the minutes of the previous meeting, the meeting agenda for the current week and the individual weekly report (that each member has to maintain).

Week	**Items due**			
	Minutes of previous meeting	**Agenda for the meeting**	**Individual weekly report**	**Other submissions**
Week 1				
Week 2		Required	Required	
Week 3	Required	Required	Required	
Week 4	Required	Required	Required	
Week 5	Required	Required	Required	Draft design document (for interim feedback)
Week 6	Required	Required	Required	
Easter break				
Week 7	Required	Required	Required	
Week 8	Required	Required	Required	
Week 9			Required	
Week 10	Required	Required	Required	
Week 11	Required			Finalized design document, software package, user test manual

Week 12				User acceptance test, Individual contribution report

GIT structure and file-naming convention

You should create the file directories on GIT.

You can access your group here: https://gitlab.eps.surrey.ac.uk/dashboard/groups. Your group should be https://gitlab.eps.surrey.ac.uk/COM2027-Group01 for group 01 and so on for other groups. If you can't find your group, please email to Norman (n.poh@surrey.ac.uk) to be added manually.

Directory name (all lower case)	What it contains
`design_doc`	Latex source with the file that the project manager can compile using the following command line. `pdflatex design_group01.tex` Project schedule (Gantt chart), design figures (data flow diagram, system architecture, use-case diagrams or other relevant diagrams) should be stored in this directory. You can create a `figure` directory or other directories so you latex files are well-organised.
`minutes`	Mark-down text file such as `minutes_week01_group01.md`
`indi_report`	Individual report in mark-down format which you update every week using the *same* file. You should only maintain one single weekly report which you update regularly. Use the filename: `firstname_lastname.md`
`src`	This is where you store all your source codes
`resources`	This is where you store additional resources required by your project, e.g., sprites, images, logo, files that your program will need.
`assessments`	All assessment forms and feedback, e.g., the interim and final audit forms, should be stored here.
`final_submission`	Where you store your APK files (where relevant)
`user_acceptance_test`	You will find the installation files of three other groups whose work you will review.
`agenda`	Markdown text file such as `agenda_week01_group01.md`

Module timetable (updated as of 6 Feb 2017)

For your convenience, a timetable of the different events are listed in Table 7. Since time table may change, please refer to https://timetable.surrey.ac.uk for the latest information.

Table 7: Lecture schedule

Thu 9 Feb 2017	15:00 – 17:50	COM2027 Talk (MindMate) and Security lecture - 04 AZ 01
Tue 14 Feb 2017	13:30 – 17:50	COM2027 Group meeting - LTA. TB22, 15 AC 03, 20 AC 03, or 35 AC 04 (check https://timetable.surrey.ac.uk)
Thu 16 Feb 2017	15:00 – 17:50	COM2027 Management lecture (Gaz) - 04 AZ 01
Tue 21 Feb 2017	13:30 – 17:50	COM2027 Group meeting - LTA. TB22, 15 AC 03, 20 AC 03, or 35 AC 04 (check https://timetable.surrey.ac.uk)
	15:00 – 15:50	COM2027 End-user requirements meeting - 35 AC 04
Thu 23 Feb 2017	15:00 – 16:50	COM2027 Technical writing and referencing - 04 AZ 01
	17:00 – 17:50	COM2027 Privacy and Compliance - 04 AZ 01
Tue 28 Feb 2017	13:30 – 17:50	COM2027 Group meeting - LTA. TB22, 15 AC 03, 20 AC 03, or 35 AC 04 (check https://timetable.surrey.ac.uk)
Thu 2 Mar 2017	15:00 – 17:50	COM2027 Management lecture (Gaz) - 04 AZ 01
Tue 7 Mar 2017	13:30 – 17:50	COM2027 Group meeting - LTA. TB22, 15 AC 03, 20 AC 03, or 35 AC 04 (check https://timetable.surrey.ac.uk)
	15:00 – 15:50	COM2027 End-user requirements meeting - 35 AC 04
Thu 9 Mar 2017	15:00 – 17:50	COM2027 Management lecture (Gaz) - 04 AZ 01
Tue 14 Mar 2017	13:30 – 17:50	COM2027 Group meeting - LTA. TB22, 15 AC 03, 20 AC 03, or 35 AC 04 (check https://timetable.surrey.ac.uk)
Tue 21 Mar 2017	13:30 – 17:50	COM2027 Group meeting - LTA. TB22, 15 AC 03, 20 AC 03, or 35 AC 04 (check https://timetable.surrey.ac.uk)
	15:00 – 15:50	COM2027 End-user requirements meeting - 35 AC 04
Tue 28 Mar 2017	13:30 – 17:50	COM2027 Group meeting
Tue 2 May 2017	13:30 – 18:00	COM2027 Group meeting - LTA | 20 AC 03
	15:00 – 15:50	COM2027 End-user requirements meeting - 35 AC 04
Tue 9 May 2017	13:30 – 18:00	COM2027 Group meeting - LTA | 20 AC 03
Tue 16 May 2017	13:30 – 18:00	COM2027 Group meeting - LTA | 20 AC 03
Tue 23 May 2017	13:30 – 16:20	COM2027 Software testing - TBD

Note: The table above is likely to change so please check https://timetable.surrey.ac.uk for updates.

Past-year student projects

To find out what last year's cohort of students did, go to the link below and download the game:
http://personal.ee.surrey.ac.uk/Personal/Norman.Poh/dropbox/COM2027_2014

References

Carifio, J. and R. J. Perla (2007). "Ten common misunderstandings, misconceptions, persistent myths and urban legends about Likert scales and Likert response formats and their antidotes." Journal of Social Sciences **3**(3): 106-116.

Pressman, R. S. (2005). Software engineering: a practitioner's approach, Palgrave Macmillan.

Sommerville, I. (2011). Software Engineering, Pearson.

FAQ

About your computer tablets

If you still have the University's tablet, please keep using it.

Q: Must I use the University's computer tablet?
A: No, you don't have to. However, you must be able to find an alternative, compatible device that you use to develop and test your codes, as well as the codes produced by your team members.

Q: I don't have a tablet; where can I collect it?
A: Please go to see Denise at 29 BB 02 during the end of the first week 1, i.e., before 10 February 2017.

Q: My tablet is faulty. What should I do?
A: Please discuss with Dr Norman Poh for record. If a replacement unit is needed, please go to see Mrs Denise Myers (D.Myers@surrey.ac.uk) in 29 BB 02 with the faulty one in order to get a replacement unit.

Grouping

Q: Can I change my group?
A: No, you cannot. In real life, you are not likely to be able to select your team members. This is consistent of one of the intended learning outcomes of this software engineering project, i.e., to help you develop your ability to work in a team.