COSC2299/2428 Software Engineering: Process and Tools
RMIT University
Major Assignment, Part A – Semester 1 2017
Due: 9AM Monday April 10 2017

Assignment Overview
You are required to develop an application matching the specification below, using appropriate process and tools, in particular those supporting collaborative project and code development. The program may be implemented in Java or PHP.

This is a team assignment; Part 1 is worth 15% towards your final grade.
Progressive marks (5 x 1%) will be allocated in weekly demonstrations in lab/tutes:
team members must be present to be assigned the associated mark on each occasion.

For the final submission (10%), team members will be required to indicate how and how much each team member contributed to the project.
Marks for individual team members may be adjusted to reflect level and quality of contribution, as indicated by peer assessments and logs from collaboration tools.

Marks will be awarded for quality of the submitted system, quality of associated artefacts (e.g., test plans, test reports), and the proper use of the SE process and tools: a guide to the progressive weekly marks is below; a marking guide for the final solution will be published during semester.

Teamwork
Teams will be self-organised; support for this will be provided the first tutorial, and by the course coordinator if required. Teams should consist of 3 or 4 members; all team members must attend the same weekly tute/lab class. There are no bonus marks for a small team.

Anyone unable to form a complete team must contact the lecturer by March 13!!

Any issues that arise within the team should be resolved by the team if possible; if not possible, then this should be brought to the attention of the course coordinator as soon as possible.

Each team should discuss how to distribute tasks. **Different team members may contribute in different ways**: e.g., writing user stories; design and implementation; designing and running tests and bug reporting; writing documentation; etc; however, **all team members must make some contribution to coding/implementation**.
The tutors will facilitate weekly Scrum meetings in the weekly tute/lab sessions but it is recommended that each team appoint a Scrum Master who will be responsible for leading Scrum meetings and updating task plans in Trello or JIRA.

Academic Integrity
The submitted assignment must be your own team's work. No marks will be awarded for any parts which are not created by your team.
Plagiarism is treated very seriously at RMIT. Plagiarism includes copying code directly from other students (other than those in your team), internet or other resources ***without proper reference***. Sometimes, students study and work on assignments together and submit similar files which may be regarded as plagiarism. Please note that your team should always create its own assignment even if you have very similar ideas to other teams. Harsh penalties may be applied in cases of plagiarism.

What to do

Part 1 of the Major Assignment is to analyse a set of requirements, design and implement a basic system. For Part 1, you are only implementing a subset of the complete system that is to be implemented over the semester---the extra functionality will be explained further in later specifications (i.e., Part 2). Despite consisting of only partial functionality, you are expected to implement a working prototype, described below. This means that the program you submit must:

1. run correctly;
2. have an appropriate level of testing. This means you should have checked the functionality of your system against the specifications. Any errors discovered should be fixed. You should have a list of final acceptance tests that you perform whenever you build your software;
3. contain appropriate user documentation that states how to install and compile the software, as well as how to use it;
4. be easy to use (i.e. have an intuitive user interface).

Your submission will contain not only the program and associated documentation, but evidence (e.g., logs) of the use of appropriate collaboration tools, including a task management tool such as Trello or JIRA; a communication tool such as Slack or HipChat; a code management tool such as Git; and any other output as appropriate (e.g., the output of unit testing).

The Requirements are listed below (in Project Description).

The weekly demonstration schedule below provides demo and delivery milestones along the path of developing the system, assuming an Agile development approach with weekly Sprints.
Some of the things you need to do each sprint and include in the final submission include:

1. Maintain and update an Issue Register / Task Tracking system such us JIRA and Trello;
2. Maintain your code (and all other artefacts) in a repository such as Github or Bitbucket. You will obtain marks for the workflow you use, e.g., proper branching etc for new features (this will be discussed in lectures);
3. Identify User Stories for the system. You should document these and use them to form the basis of your task plans, as well as for designing Acceptance Tests.
4. Define a series of Acceptance Tests, i.e., tests designed to find any bugs or problems in your system, as well as evidence that you have performed testing (will be discussed in lectures). Acceptance tests should be designed to reflect the total (Part 1) functionality; if you do not complete the functionality by submission, then we would expect some of these tests to fail.

The final submission will also include:

1. A class diagram of the system, which includes appropriate attributes, methods and relationships.
2. Basic user documentation should be provided, including screenshots, making clear how to install and run your program.

Project Description

You are required to design, develop and test an application named "Appointment Booking System". This is basically a booking system that can be used by any business, allowing a customer to book a time slot for an appointment. The system can be for any kind of business, e.g., a hairdresser, a gym, or dentist. The application will display a customer's booking after it is entered and allow them to provide address, contact and other details. If the customer has made a booking previously and their information has been stored, the application will allow them to retrieve that information. A customer should also be able to track their booking status. Besides providing the required functionalities, your program should incorporate appropriate error handling, e.g., booking outside allowed times, or double-bookings.

In Part 1, you can focus on customer registration and implement the system for a single business. You may allow the system to define parameters such as allowed booking times (e.g., 9am till 5pm, Monday to Friday) in a config file or database (as below).
(Part 2 will allow a business to register itself and define those details when it registers.)

Functional requirements

1. The system should display a main page: When your program starts, a main page should be displayed as login/register pages. The program must authenticate and authorise users and based on user type/role, i.e., business owner or customer.
2. User registration is only for customers; business owner information can be accessible through a file called business.txt (or a db), including business name, business owner name, address, phone, username and password. Customers can register themselves via data entry; this information should be saved in customerinfo.txt (or a db).
3. If the owner can login successfully then the owner is able to add a new employee, add working time/dates for the next month, look at the summaries of bookings, new booking, show all workers' availability for the next 7 days.
4. Customer only view available days/time but not yet book a slot (Part 2);
5. A text file (customerinfo.txt as shown below) should be created; an alternative is to use a database which must be distributed with the application. Each line shows a customer's name, username, password, address, contact number. You should create at least 3 entries manually for testing purposes. (In practice, customer information would likely be encrypted, but this can be ignored for this assignment.)

You are required to use a recent version of PhP or Java (JDK 1.8 or later).
Code quality and clarity will be marked so make sure you properly document and lay out your code, avoid hardcoding, **use a logging framework**, etc. Analyse the requirements and think carefully about the basic class design before commencing coding.

You do **not** have to provide a GUI interface in Part 1 but you can if you prefer. If you use a console interface, the interface (and what values to be entered) should be very clear and should be error-checked to avoid crashing!

More details and clarifications of Requirements will be posted to Blackboard.

Final Submission of Part 1 --- 10%

Part 1 of the assignment is to be submitted by **9AM Monday 10th April 2017**.

Include a README file in your submission, containing:

- Names and student number for all team members;
- A short description of the contributions of each team member to the submission, including a % contribution of each team member (these should add up to 100%) and a statement of ***what*** part of the submission each team member was responsible for.

Marking criteria for the assignment will be discussed in class and published on Blackboard, but will include:

- how well your software meets the requirements at the demonstration;
- quality of the software produced, including quality of documentation/comments and adherence to programming standards;
- adherence to process and use of the planning and collaboration tools;
- how consistently you work throughout the project, based on weekly meetings/demos and details found in your code repository.

Marks for individual team members may be adjusted to reflect level and quality of contribution, as indicated by peer assessments and logs from collaboration tools.

All assignment **submission is electronic** – submission instructions, including **what to submit,** will be posted to Blackboard.
Assignments must be submitted by due date/time-–**late assignments** may not be accepted at all.

BONUS: You may also decide to use a full Test-Driven Development approach for your project, whereby comprehensive automated tests are written before development commences: Proper use of a **Test-Driven Development process will attract** a **15% BONUS** for Part 1, but this needs to be implemented from the start of the assignment and should be discussed with your tutor.
TDD and Unit Testing will be discussed in lectures.

You are required to manage your time, and required to work consistently until submission.
If you are ill, and are unable to progress on your work, you should submit a request for special consideration. We will then calculate the amount of time you were unable to progress on your work as a proportion of the time available for the assignment, as well as assessing the work you have submitted. In general teams should be able to cope with the temporary loss of a team member for a short time. If a team member is absent for any length of time, you should consult your lecturer or head tutor. We may then make an appropriate adjustment to your team's result. Illnesses of 1 or 2 days are unlikely to affect your work greatly, and are thus unlikely to be considered justifiable reasons for your work not progressing.
If you are ill and unable to work for either a major period (>50%) or the entire period of the assignment part, you should contact your student advisor to discuss your situation.

A consequence of not submitting an assignment/part on time is that your team may get 0 marks for that assignment/part.

Progressive / Weekly Assessment Tasks --- 5%

An important aspect of this course is the learning and practice of modern Software Development techniques. Agile Software Engineering in particular requires regular meetings, short sprints of development, as well as testing-focused processes (e.g., Test Driven Development). Tools that facilitate agile practice include:

- task management tools, such as JIRA or Trello;
- team-communication tools, such as Slack;
- code management repositories, such as Github or Bitbucket;
- automated unit testing frameworks, such as JUnit or PHPUnit;
- test management frameworks, such as leantesting.com;
- automated build tools, such as Ant, Maven or Gradle.

This assignment requires you to use an Agile development methodology (Scrum) and associated tools for each aspect of the process. Note: **you may leave the use of Build tools until Part 2**.

Each week, your tutorial/lab session will involve a Scrum meeting, including a short (3-5 min) presentation to the "client" (i.e., your tutor). For Weeks 3-7, there will be a 1-mark progress for achieving the previous weeks tasks, as defined below.

- **Each weekly mark is 0.5 for Team and 0.5 for Individual Contribution for that week.**
- **Only team members present at the team presentation qualify for the weekly mark**.

The schedule for tasks and associated demonstrations are as follows:

Week	Tasks to work on this week	Delivery based on previous week's tasks	Marks
1	form groups, set up required tools, define second week tasks in task manager		0
2	develop user stories for main functions such as user management	names of team members and roles; tasks for following week; github ready; basic class design	0
3	start development for the first designed function; test and submit. design next function such as reading input file & display stored data to owner	mock-up of login in console; user story for log in and registration; tasks for week three	1
4	review previous task; show user story of this week's task; start development, test, submit; define next task; design next function, such as another main menu function	mock-up of for the next tasks; user story for next tasks; acceptance test for the previous task; updated task register; updated github	1
5	review previous task; show user story of this week's task; start development, test, submit; design next function such as store details	mock-up of next tasks; user story for next tasks; acceptance test for previous task; updated task register; updated github	1
6	review previous task; show user story of this week's task; start development, test, submit; design next task next/remaining function/s; make a complete build for **Part 1 submission**	mock-up of the next tasks, user story for next tasks; acceptance tests previous task; updated task issuer; updated github	1
7		demonstrate working submission, all acceptance tests; show tutor updated task register and updated github	1

Proposed activities each week:

It is recommended that each group have at least 2 scrum meetings each week; meetings can be face-to-face or online. Online communication must be documented through Slack or some other collaboration tool. It is very important for team members to be in touch every working day based on your project schedule.

Week 1:
Form your team and send an email about your team members name and student number to your tutor. You also need to setup your collaboration tools, and submit a fake app into the code repository; every member should make at list one submission. You also need to have early team meetings to clarify member roles, and issue tasks for the next week. All meetings must be documented and signed.

Week 2:
Start project documentation, with writing the project scope, class design, and planning work. You also need to start writing the main user stories of this project, e.g., login and/or registration functions. Create mock-up prototypes for these functions. You can define all user management functions validations and limitation after creating their user stories.
If you **plan to use Test-Driven Development**, then you should commence writing the associated automated tests by end of Week 2 / start of Week 3. Otherwise, team members can commence designing Acceptance Tests, based on User Stories as they are written.

Week 3:
In this week, teams should start development of functions that build on the previous week, finish the task, debug, test and submit. You need to also make new tasks based new user stories for new functions such as loading and storing data on users and their selections.

Week 4:
Review previous tasks and fix problems if needed. Develop all assigned tasks for this week, debug, test and submit. Make user stories for new tasks and clarify tasks for the next week.
Note: as you write code, you should be adding documentation and **logging** statements as you go; in particular, logging statements will help you debug your code as it gets complex. You should also be maintaining a bug reporting tool to document / assign / manage bugs.

Week 5:
Review previous tasks and fix problems/bugs if needed. Develop all assigned tasks for this week, debug, test and submit. Write user stories for new functions and clarify tasks for the next week.

Week 6:
Review all previous tasks and fix problems if needed. Develop all assigned tasks for this week, debug, test and submit. Try to polish your code, fix any remaining problems and prepare your full submission, including basic User Documentation.
Part 2 specifications will be available and you can start planning new tasks for Part 2.

Week 7:
Submit by 9am Monday; demonstrate your Part 1 submission to your tutor; commence working on Part 2.