TE WHARE WĀNANGA O TE ŪPOKO O TE IKA A MĀUI

VICTORIA UNIVERSITY OF WELLINGTON

School of Information Management

INFO 226 APPLICATION DEVELOPMENT

Trimester 1, 2019

Student Guide

Version 1.2

Welcome

Dear Students:

Welcome to INFO 226. This Student Guide is the roadmap to your course. We hope this document will prove to be helpful to you.

Sincerely,

Pedro Antunes, Jesse Dinneen.

Introduction

The purpose of this document is to provide a comprehensive overview of INFO 226, including how it has been thought out and structured, requirements and recommendations on how to organise your work.

Course strategy

Application development has become a very complex endeavour mixing project management, software development methodologies, a myriad of programming languages, multiple technological architectures, development frameworks, and a countless number of libraries. Quite frankly, it is impossible to master all different concepts in just 12 weeks. Very realistically, we will grasp the following:

- Introduce you to fundamental application development concepts
- Introduce you to a modern management approach (Agile)
- Explore some fundamental concepts related to application architectures (client/server)
- Explore some core web development technologies (HTML, CSS and Javascript)
- Explore a modern web development framework (Angular.js 1.x)
- Experiment all these topics in a popular and friendly online environment (Plunker)

All in all, not bad for 12 weeks!

Agile

Nowadays, most companies have adopted the Agile Principles drawn in the Agile Manifesto, which states the following (our emphasis):

- Business people and developers **must work together** daily throughout the project
- Build projects around **motivated individuals**. Give them the environment and support their needs, and **trust** them to get the job done
- The most efficient and effective method of conveying information to and within a development team is **face-to-face conversation**
- Deliver working software **frequently**, from a couple of weeks to a couple of months, with a preference to the shorter timescale
- The best architectures, requirements, and designs emerge from **self-organizing teams**
- At regular intervals, the team **reflects** on how to become more effective, then tunes and adjusts its behaviour accordingly

We suggest you internalise these principles, as we will painstakingly use them.

The Agile approach favours a set of unusual and disruptive approaches to project management. A very important one is to avoid project planning, bureaucracy, methodology, and documentation. Instead, focus should be on delivering actual products that work. We will closely follow this view in INFO 226.

Planning is reduced to a minimum: you take responsibility for organising your work and do it in your own way.

Two other important ideas about being Agile are to be flexible and to focus on instant gratification. In INFO 226, we will strive to have a product that can be demoed on day one and everyday afterwards. Initially it will not have any significant functionality, but over time it will evolve towards the final product in an organic way.

The final important idea behind the Agile approach is to work in groups, a topic that is discussed next.

You can read more about Agile here:

- https://www.scrumguides.org/docs/scrumguide/v2017/2017-Scrum-Guide-US.pdf#zoom=100

Agile vocabulary

You should familiarise yourself with the following terms used by the Agile community.

- Product – A coherent and functional piece of software that has to be delivered to a client
- Vision – What we aim to achieve with a product
- Project management – An old way of looking at product development. Agile management is the new way
- Project plan – An obsolete technique. It has been superseded by the product backlog
- Scrum – An agile approach to product development, which uses sprints to iterate the development
- Scrum team – A team committed to develop a product. The scrum team is self-organised and self-managed
- Scrum board – A dashboard showing all elements related to the product development, i.e. vision, product backlog, user stories, sprints, sprint backlog, etc.
- Developer – A member of the scrum team with technical capabilities such as business analysis, design and software development
- Product owner – A member of the scrum team that is responsible for the vision and product backlog
- Product backlog – Prioritized list of features required by a product
- Product backlog item – A feature describing some desired functionality. A product backlog item may contain a user story, a user story card, or any other description of the desired feature. A product backlog item includes a definition of done
- Definition of done – A statement about what conditions are required to consider the product backlog completed. The definition of done may have one or more acceptance criteria
- Acceptance criteria – A set of tests used to check if a product backlog item is completed to satisfaction
- User story – Defines a product requirement from the user's point of view. A user story states exactly what a user would like to do with a product (a functional requirement) or would like to find in a product (a quality requirement)
- User story card – A card that describes a user story with more detail. It usually has a picture illustrating a user interface, and pieces of text explaining how the user interacts with the product
- Epic – A collection of user stories. Epics help organising stories, especially when products are complex
- Sprint – The functionality a scrum team commits to deliver during a time box. A sprint commits a list of product backlog items to be delivered within the time-box period. Each sprint has a sprint backlog
- Time box – A fixed period of time. In our case, we will use one-week time boxes
- Sprint backlog – A list of sprint tasks the scrum team will perform in a sprint
- Sprint task – Describes one step towards achieving the set of product backlog items committed by a sprint. A sprint task involves one day or less work
- Scrum master – A member of the scrum team that helps solving problems with sprints. The scrum master does not manage the team (as the team is self-organised and self-managed). The scrum master is a servant, not a manager. In our specific context, the scrum master will be responsible for maintaining the scrum board
- Pregame – A preparation phase, when a sprint is waiting to be developed. Usually this phase involves selecting a set of items from the product backlog and then defining the tasks necessary to develop them, i.e. defining the sprint backlog
- Game – The development phase, when a sprint is being developed. As the work moves forward, the team ticks off completed tasks from the sprint backlog

- Postgame – The closure phase, when a sprint has been fully implemented. All tasks in the sprint backlog have been ticked off. The corresponding items are then removed from the product backlog and archived
- Vertical slicing – A way of decomposing a product into smaller pieces, each developed separately. A typical approach is to start with an epic, breaking the epic into multiple user stories, and then develop each story separately
- Horizontal slicing – A way of decomposing a product in multiple layers, each layer tackling a specific aspect of the product development. A typical approach slices the product into client and server layers

Development language

Some development terms we are going to use:

- Trello – A project management tool popular in the Agile community
- Plunk – A piece of code developed in Plunker
- Plunker – An online software development tool
- Demo – A session conducted by the developers showing a piece of software running
- Review – A formal inspection activity that checks if some pieces of code have been properly developed
- Client/server – Typical approach to vertical slicing a product. The client does light processing and communicates with a remote server, which handles heavy processing
- Boilerplate – Pieces of code added by development frameworks
- Framework – A set of auxiliary libraries that support code development. Plunker includes several frameworks

Group work

Group work is very common in application development and Agile, suggested by specific Agile communities like XP (eXtreme Programming) and Scrum. The Agile Manifesto suggests that projects are more successful if less emphasis is placed on managerial control and more emphasis is placed on team collaboration, a can-do attitude, and self-organisation. We will closely follow these ideas in INFO 226.

However, experience has shown several roadblocks to effective group work:

- Probably the most critical one is that students, when they arrive to INFO 226, do not seem to have already developed collaborative practices
- Incompatible timetables and diversity of course enrolments also seem to make it difficult for students to work in groups
- Somehow students seem to consider that studying is a 9-to-5 job, with evenings and weekends dedicated to other things, like sleeping or having a life, which seems to make it difficult to work together outside regular classes
- For whatever reason, students do not share much knowledge and experience with other students
- There are few positive incentives to work together. Even though you already know that most jobs in most companies require working in groups, that seems to be a distant payoff
- Quite the contrary, there are many incentives to work individually. In particular, the VUW Assessment Handbook creates important constraints to assessments based on group work
- There are few negative incentives that can be associated to failing to work together
- Students can easily complain they may be treated unfairly by working in groups

And still… We will ask you to work in groups.

Reasons for working in groups

Considering the roadblocks, why would we insist on group work? Well, the first reason is that most jobs in most companies effectively require group work, so why not doing it here? But the most important reason is related with the main purpose of this course: learning application development.

Application development is notoriously stressful. It requires coding scripts that will not be accepted by computers if not 100% correct. It requires building code on top of libraries developed by others, which are difficult to understand and use. Furthermore, coding requires acquiring practical knowledge from the grapevine, instead of traditional classes. Coding also requires developing logical, structured problem-solving skills. And it also requires a lot of trial and error.

All in all, these activities are much better done with help and collaboration from others. This is especially true in a business school environment, where students do not have strong intrinsic motivation to spend too much time coding—they want to build solutions and move on—and therefore support from a colleague is very handy when a problem seems insurmountable and help from lecturers and tutors seems limited or even ineffective.

Another important reason is that in the past we have seen amazing groups working side-by-side on their projects and achieving great grades.

Group structure

Groups of two will be organised in the following way:

- One student will be the product owner, i.e. responsible for the vision and product backlog
- The other student will be the scrum master, i.e. responsible for the scrum board
- Both students will take different responsibilities for the sprint backlogs

Groups of three will be organised in the following way:

- One student will be the product owner, i.e. responsible for the vision and product backlog
- Another student will be the scrum master, i.e. responsible for the scrum board
- The third student will be responsible for the definition of done (including acceptance criteria)
- The three students will take different responsibilities for the sprint backlogs

Setting up groups

We (course coordinator, lecturer and tutors) do not like setting up groups and we **do not** take responsibility for setting up groups. We can facilitate. Groups are too important for us to set them up in a random way. If a group fails, we do not like being blamed for setting you with the wrong person. Choose carefully your colleague. The first workshop gives you the right time and place to look around for a partner.

Note that your best friend is not necessarily your best partner.

In general, groups have 2 elements. We may consider exceptions to this rule if a workshop has an odd number of students: the student left out will be allowed to integrate another group, which will then have three members.

However, it should be obvious that group members must belong to the same workshop.

Groups are setup in the first workshop, which occurs in **week 1**. Having a workshop in week 1 is unusual but the purpose is exactly to setup groups as soon as possible.

All exceptional cases, including new student arrivals and students moving to different workshops, must be dealt with in week 2. After week 2, no group setups will be considered and students with no assigned group will have to work individually. In our experience, groups formed too late or in opportunistic ways tend to break down and therefore we think it is better to work individually than to spend too much time roaming around in search for a group.

After a group has been setup, it cannot be modified. It can only break down. Breakdowns are discussed below.

Group breakdowns

Group breakdowns are unfortunately too common (we started with a 20% ratio but have improved to 10%, which is still too high).

Breakdowns may happen at different points in time, for which different mitigation and recovery procedures may be required. The following rules specify what happens when a group breaks down:

- The breakdown happens before any assignment has been submitted – You have to individually submit all assignments. All assignments will be assessed individually. Marking will take into consideration that everything was done in individual mode. Only project materials developed by the individual can be submitted. You cannot use any piece of work that was developed by others.
- The breakdown happens after some assignments have been submitted – The pieces of assignment already submitted are marked according to the predefined structure and type (see assessment section). The subsequent assignments will be submitted individually; and they will also be marked individually. All project materials already submitted belong to the group and can be reused by individuals.
- Marking always takes into consideration the workload defined in the course outline, i.e. 5 hours a week per person. This means that when a group breaks down the project has to be revised in order to accommodate the expected workload. That is, every project is expected to have a workload of 5 hours a week per person if done either by a group or an individual.

- All group members should immediately notify the course coordinator about a group breakdown. The causes for the breakdown should be disclosed and justified.
- We will negatively consider causes for group breakdown looking rather superficial, as they reveal lack of commitment to the project. Obviously, we do not penalise students for special circumstances, which are described in the Assessment Handbook.
- We are committed to avoid punishing students negatively affected by their colleagues. We are also committed to avoid easing students that failed their obligations to the group.
- When there is a breakdown, we will hear from both parties.

Extensions

In case an extension is granted to a student, it will be automatically extended to the group.

Group work Q&A

My colleague is from CS and I'm from VBS. Should we set up a group?

Based on previous experience with this scenario, our recommendation is that groups should be homogeneous, i.e. VBS only or CS only.

I want to work alone from the beginning.

Provide a good justification and it will be considered. We will not enforce group work but strongly believe that group work is extremely valuable—for this course and your future career. There is a higher probability of gaining better marks if you utilise group synergies.

I am a code warrior and prefer to work alone.

Collaboration is a fundamental part of this course. If you decide to work alone you must be resilient and able to solve problems by yourself, something that every code warrior knows very well.

We are good friends; can we have a group of 3?

No, unless one of them is the last element in a workshop with an odd number of students. And by the way, friendship may not be the best criterion for selecting a group member.

We are close to the deadline and my colleague has not sent me what I need to submit the assignment.

You need to strategize in advance how to avoid this scenario—either by working together and having good communication, or by finding a way to avoid depending too much on your colleague. If this happens too close to the deadline, it is a good reason to report a group breakdown. From then on, you will work individually.

We cannot meet face-to-face.

Communication is a key issue in this course. Use online tools to communicate. No, we do not recommend any particular medium.

My colleague decided to leave this course.

From now on, you will work individually.

My colleague has a health issue and needs an extension.

Your colleague should apply for an extension, which, if granted, will be automatically granted to the group.

I have been doing most of the work; my colleague is not committed.

First, try to solve the problem with your colleague in a professional manner. If the problem is not solved, report to us so it can be taken into consideration when marking. You should also consider reporting a group breakdown.

We have done all the work together and my colleague achieved a better mark than me.

Keep in mind that some assignments are individually marked. Marks correspond to the quality of what was delivered so you need to be mindful about the quality of your pieces of work. Look for a win-win situation by giving positive contributions to the group and being attentive to what happens. If you avoid contributing and expect a win-lose, you may instead get the same treatment and be in a lose-lose situation.

How can I make it clear that group breakdown was not my fault?

Report to us. Provide evidence. Online tools can be helpful in documenting what happened.

Assessment

In this course, we consider three types of assessment:

- **Individual review**: Even though you work as a team, you will take individual responsibilities for some parts of the work. For instance, one developer may take the lead on a certain sprint task with support from the other colleague, and then will switch positions and provide support in another sprint. Even though the work is done collaboratively, there are clear responsibilities, which must be managed by the team. Individual reviews assess individual work taking into consideration the individual responsibilities.
- **Group review**: The team members are collectively responsible for some parts of the work. Group reviews assess the parts done by the group, assigning the same mark to the team members.
- **Exam**: The usual.

This course has 70% project assessment and 30% examination. The overall assessment structure is as follows:

- Week 3 review: (individual)
- Week 5 review: 15% (individual)
- Week 7 review: 15% (individual)
- Week 9 review: 15% (individual)
- Demo: 15% (group)
- Exam: 30% (individual)

The reviews and demo are done in the SIM laboratories during the designated workshop times. All group members are expected to attend and participate in the reviews/demo. In case some element or the whole group are unable to attend, the group may either request an extension or make special arrangements to attend another workshop occurring in the same day/week. Any special arrangements must be requested at least one day before the designated workshop.

Assignments

Piece of assessment	Type of assessment	Expected load	Default date [1]	Actual date [1]	Marks	CLO	What is assessed
Week 3 review	Individual	10 hours	Wks. 3		10%	2	Product backlog and scrum board
Week 5 review	Individual	10 hours	Wks. 5		15%	2	Evidence that some sprints have been completed
Week 7 review	Individual	10 hours	Wks. 7		15%	3,4	Evidence that some sprints have been completed
Week 8 review	Individual	10 hours	Wks. 9		15%	3,4	Evidence that some sprints have been completed
Demo	Group	15 hours	Wks. 12		15%	3,4	Working application

[1] We use a default calendar consisting of 12 consecutive weeks, named from week 1 to 12. As you know, the actual trimester is split in two parts with some free weeks in between, which we do not account for. Furthermore, sometimes trimesters spread through 13 weeks. So, **you** must map the default calendar into the real calendar. This is common practice in project management. A column has been added to the table for you to add the actual dates where pieces of assessment occur.

Weekly Overview

Weeks	Lectures	Workshops	Assessment (during workshops)
1	Agile. Key Agile concepts.	Induction to group work. Group work rules and best practices.	
2	Horizontal and vertical slicing. Architecture of web applications. Development patterns.	User stories. Breaking a project into user stories. Exercise: Create a product backlog. Exercise: Create a scrum board. Using Trello.	
3	Static web pages (HTML). Forms.	Development tool: Plunker. Plunks and project files. Exercise: "Hello world" plunk.	**Week 3 review.** Students will explain and discuss the product backlog and the scrum board.
4	Page layout (CSS).	HTML syntax.	

		HTML basic elements. HTML forms. Exercise: Login form. Developing basic layout and HTML elements for project.	
5	Dynamic web pages (Angular). Data binding. Dependency injection. Rendering.	CSS syntax. Exercise: Login form layout. Improving project layout.	**Week 5 review.** Students will explain the completed sprints and discuss the progress of the project.
6	Controllers. Event handlers.	Angular directives. Angular expressions. Exercise: Login with data bindings. Adding Angular to project.	
7	Accessing the backend (JSON).	Angular controllers. Angular event handlers. Exercise: Login with controllers. Creating controllers for project.	**Week 7 review**. Students will explain the completed sprints and discuss the progress of the project.
8	Javascript.	Angular collections. Angular HTTP. Exercise: Login with list of users. Interacting with project server.	
9	Javascript	Javascript. Exercise: Login with validation.	**Week 9 review**. Students will explain the completed sprints and discuss the progress of the project**.**
10	Document object model (DOM).	Angular visibility and enable/disable. Angular animations. Exercise: Login visibility and disable	
11	Mashups.	Project support. Demo preparation.	
12	Revisions.	**-**	**Demo.** The group will demonstrate the application.

Marking rubric

Week 3 review

Work	**Exemplary**	**Great performance but with low value results**. Still calibrating what can be done. The project can be delivered on time, but more control is needed.	**Great performance but could deliver more**. Reveals capacity to deliver the project on time. Reasonable results.	**Sprint was fully successful**. Delivers significant results. Excellent work. Reveals high capacity to deliver the project on time.
	4 - 4.5 - 5			
	Satisfactory	**Negligible sprint**. Performance is acceptable but is not delivering results. The project may be at risk.	**Sprint was partially successful**. Delivers reasonable results. Work can be partially tracked.	**Great sprint, but the project could be more structured**. Delivers important results. Reveals capacity to deliver the project on time. Reasonable work.
	3 – 3.5			
	Unsatisfactory	**Sprint was not successful**. Important work was not done. Project performance cannot be tracked. Project is at risk.	**Negligible sprint**. Performance seems inconsistent. Work cannot be tracked. Project may be at risk.	**Good sprint but performance seems erratic**. Still calibrating how to organise the work. The project can still be delivered on time, after improving performance.
	0 - 1 - 2			
		0 - 1 - 2	3 – 3.5	4 - 4.5 - 5
		Unsatisfactory	**Satisfactory**	**Exemplary**
		Results		

Week 5, 7, 9 reviews

Work	**Exemplary** 8 - 9 - 10	**Great performance but with low value results**. Still calibrating what can be done. The project can be delivered on time, but more control is needed.	**Great performance but could deliver more**. Reveals capacity to deliver the project on time. Reasonable results.	**Sprint was fully successful**. Delivers significant results. Excellent work. Reveals high capacity to deliver the project on time.
	Satisfactory 5 - 6 - 7	**Negligible sprint**. Performance is acceptable but is not delivering results. The project may be at risk.	**Sprint was partially successful**. Delivers reasonable results. Work can be partially tracked.	**Great sprint, but the project could be more structured**. Delivers important results. Reveals capacity to deliver the project on time. Reasonable work.
	Unsatisfactory 0 - 1- 2 - 3 - 4	**Sprint was not successful**. Important work was not done. Project performance cannot be tracked. Project is at risk.	**Negligible sprint**. Performance seems inconsistent. Work cannot be tracked. Project may be at risk.	**Good sprint but performance seems erratic**. Still calibrating how to organise the work. The project can still be delivered on time, after improving performance.
		0 - 1 - 2	3 – 3.5	4 - 4.5 - 5
		Unsatisfactory	**Satisfactory**	**Exemplary**
		Results		

Demo

	Exemplary	**Satisfactory**	**Unsatisfactory**
Product	**Feature rich.** Inspired interpretation of the project. It covers a wide range of features. May include features that were not required.	**Reasonable set of features.** Complies with the project. Restricted set of features.	**Incomplete.** Important features are missing. Not enough features have been implemented.
	5 - 4.5 - 4	3.5 - 3	2 - 1.5 - 1 - 0.5 - 0
Realisation	**Excellent realisation**. Great organisation. Consistent deliveries.	**Accomplished realisation.** Good organisation. On track.	**Poor project realisation.** Basic organisation. Weak track record.
	5 - 4.5 - 4	3.5 - 3	2 - 1.5 - 1 - 0.5 - 0
Demo	**Product works flawlessly.** Product is very realistic. All features demoed.	**Product works reasonably.** A good number of features were presented.	**Product is not realistic.** Major problems running the code.
	5 - 4.5 - 4	3.5 - 3	2 - 1.5 - 1 - 0.5 - 0

Technology and tools

Agile management

Trello – https://trello.com

We will use this online tool for Agile management. This is a very simple and flexible tool, which is inspired on the Japanese Kanban production management approach. Trello supports developing the product backlog and scrum board.

You should sign up on the Trello website. The functionalities you need to use are free.

Code development

Plunker – https://plnkr.co

This is an online platform for editing, developing, archiving, and sharing code for the web. In particular, the editor offers an as-you-type visualisation of your pieces of code that helps learning by doing. Other functionalities we will use include archiving and sharing your code, and managing versions.

You should sign up to use Plunker. The tool is completely free. You sign up in Plunker using a Github account. So you should go to Github (https://github.com/) and sign up there as well. Github is also completely free.

HTML, CSS, Javascript

This is the classic trilogy of client-side coding for the web. We will only focus on the client-side in INFO 226.

Angular JS 1.x – https://angularjs.org/

This open-source framework helps building more sophisticated and modern web applications. Angular JS code is embedded with HTML and provides many useful extensions. Note we will use the version 1.x (at least 1.4 and eventually 1.6), not the 2.0.

JSON - www.json.org/

JSON is a data interchange format. We will use JSON in a very basic way, to interact with a server.

Web server communication (HTTP/JSON)

In order for you to develop a realistic client, we will provide a functional server that implements a set of data management functions. You do not need to do any development on the server-side, though you will need to know how to communicate with the server using HTTP functions such as GET and POST.

Bibliography

This course does not have a traditional bibliography. Considering the technologies being used, we will rely on online sources. Below we list some of the most useful sources.

Agile management

These documents describe the Scrum approach to application development:

- http://scrumreferencecard.com/scrum-reference-card/
- https://www.scrumguides.org/docs/scrumguide/v2017/2017-Scrum-Guide-US.pdf#zoom=100
- http://www.brianidavidson.com/agile/docs/scrumprimer121.pdf
- https://www.cpe.ku.ac.th/~jim/common/articles/Schwaber1995%20-%20Scrum%20Development%20Process.pdf

This document describes how to develop user stories, with examples:

- http://www.alexandercowan.com/best-agile-user-story/#Creating_an_Agile_User_Story

Code development

The infinite source of almost everything you need to know about coding for INFO226, including HTML, CSS, Javascript, Angular, and JSON:

- http://www.w3schools.com/

This is a highly recommended free book on Angular:

- http://www.angularjsbook.com/angular-basics/chapters/introduction/

The following guide also provides fundamental information about Angular:

- https://docs.angularjs.org/guide

Books

In case you really need to consult more traditional books, the following references are available from the library (full text available as electronic resources):

- MEAN web development: master real-time web application development using a mean combination of MongoDB, Express, Angular JS, and Node.js. Haviv, Amos. 2014.
- JavaScript. Crockford Douglas. 2008.
- Beginning HTML & CSS. Larsen, Rob. 2013.

Project assignment

You need to familiarise with the project as soon as possible. Do not delay reading this section.

Introductory remarks

This project was conceived to be as realistic as possible. This means the project description is brief, the requirements are not fully specified and may change over time, the documentation is scarce, the stakeholders are elusive, and – of course – the client wants the project done yesterday. Yes, you should know this is how most information systems projects are done. You have to deal with this reality, so please do not complain that something is missing, unclear, not be given to you in a proper way, etc. The project description provided in this document is perfect from the client's point of view. If you miss anything, fill the gaps and move on.

You – the scrum team – are responsible for providing a finished product by the deadline which fulfils the client's explicit and implicit requirements. Hint: an explicit requirement means the client stated "I want *this*"; while an implicit requirement means that *this* should work reasonably well, is fit for purpose, is user friendly, makes me happy, does what I had in mind but forgot to tell you, etc. In case pieces of information are missing or are equivocal, use imagination and common sense to fill the gaps and satisfy the client.

Context

NZTA is responsible for a large number of public roads in NZ. From time to time, roads need maintenance works such as repairs, resurfacing, alterations, modernisation, etc. NZTA seeks to develop the TRACK application to help tracking the maintenance works.

The idea behind TRACK is to oversee what is going on in the field and in real time. TRACK is expected to track when, where and what has to be done, who is going to do it, when works are expected to be completed, and what problems occurred.

Note that TRACK is not intended for planning or managing projects. TRACK is only focussed on due diligence: checking status, tracking progress, and identifying problems as they emerge.

If you think about the number of roads, each with different needs, number of maintenance works, each dealing with different situations, and number of maintenance companies doing the projects, it is definitely a big task to keep due diligence.

Project goals

TRACK manages information about road maintenance projects in a timely way. This means having updated information from anywhere (mobile) and for everyone (NZTA and contractors).

NZTA staff are constantly in the field overseeing projects: checking what has to be done, talking with contractors, and notifying any potential problems that may arise. NZTA has been using project management tools to solve this problem but cannot do it any more, as they do not provide the required levels of agility and flexibility.

Project organisation

NZTA is your client. TRACK is the project you need to develop. NZTA does not need a project plan, business analysis, or even project reports. Instead, they want to see the project rolling from day one—that means Agile. This will help building trust that you can deliver what they need. You are expected to deliver an application that works from day one, and to keep delivering functionality on a weekly basis.

System requirements

- The application will adopt a client-server architecture.
- On the server side, a web server will be provided with a fixed set of business-logic functions that can be accessed using HTTP and JSON. No work will be done on the server side. **No changes can be done either**.
- The project is centred on developing the client.
- The client will be a single-page application, i.e. running with a single page load on a browser.
- The client will run on modern browsers (Safari, Chrome, Firefox) on mobile phones.
- The client will be developed with standard web technology: HTML, CSS, Javascript, and Angular.js 1.x.

Data requirements

- A road list is the entry point for managing all data about road maintenance.
- The road list has the following data: identifier, official road code, type of road, section, named location, GPS location.

- Related to a road, there is a list of maintenance projects.
- A maintenance project contains the following data: identifier, name, status (closed, waiting, scheduled, opened, approved, rejected, closed), start date, finish date, contractor, list of works, list of problems, list of comments from contractor, list of comments from inspector.
- A road work contains: type of work, subcontractors, status (done, on-going, scheduled, postponed, cancelled).
- A comment contains: author (inspector, contractor, manager) and text.
- The server maintains a user list with: login name, password, and type of user (inspector, contractor, manager).

Users

- Each project has an inspector.
- Each project has a contractor.
- TRACK has managers to update road and project data.

Functional requirements

- All users have to login to access the application.
- The inspector can see a list of roads.
- The inspector can select a road from the directory and view the related information.
- The inspector can select a project and views the project data.
- The inspector can modify project data.
- The manager can open/close/add/archive roads and projects.
- The manager can add/delete/modify the project data.
- The contractor can view information related to own projects.
- The contractor can edit information about subcontractors.
- The inspector and contractor can open/close problems.
- The inspector and contractor can comment on a project.

Note: This is a minimal set of functions required by NZTA. Other good ideas that improve value are welcomed and will be rewarded. You are invited to add useful functionality to the application. Note however that you cannot change the server behaviour.

Other requirements

- Forms have good presentation.
- Forms are easy to use.
- Dropdown menus are preferred over empty fields.
- Forms have pre-selected options whenever possible.

Data structures

The data structures used by the server are illustrated as follows.

List of users:

```
user_list.json
  {"Users": [
          { "loginName": "bloggsjoe1", "password": "mypassword1", "userType": "inspector" },
          { ... }
  ]}
```

List of roads:

```
road_dir.json
  {"Roads": [
          { “ID”: “1”,
           "Code": "SH1 south",
           “Type”: "state highway"
           "Section": "kapiti",
           “Location”: "kapiti interchange ",
           “GPS”: “latitude xxxxx, longitude xxxxx"},
         { ... }
  ]}
```

Projects:

```
project.[number].json
  {"ID": "1",
    "Road": "1",
    "Name": "resurfacing.",
    "Status": "closed",
    "StartDate": "2019-12-12T00:00:00",
    "EndDate": "2019-12-14T00:00:00",
    "Contractor": "ABC Company",
    "Problems": [
            {"Author": "John D.", “Text”: “safety issues”},
            { … }
    ],
    "Works": [
            {"Type": "resurfacing", “SubContractors”: “Local Inc; Danny Do”, "Status": "done"},
            { … }
    ],
   "Comments": [
            {"Author": "John D.", "Text": "Work completed."},
            { … }
    ]
}
```

These data structures are managed by the server as flat **files**. In particular, projects are **not** managed by the server as a relational structure. For that reason, you need to carefully manage the relationship between roads and projects, generating unique “ID”s to identify projects and associating projects with roads using the “Road” attribute in each project.[number].json.

Client-server communication

The client will communicate with the following server:

https://track.sim.vuw.ac.nz/

The server will work with multiple clients based on usernames:

https:// track.sim.vuw.ac.nz/api/your_username

Note this is a simplistic approach, adopted just to test the client: If your colleagues know your username, they will be able to modify your data.

Client-server messages (HTTP)

Get list of users:

```
HTTP GET
https://track.sim.vuw.ac.nz/api/<user>/user_list.json
{
 "Users": [
  {
    "ID": "1",
    "LoginName": "bloggsjoe",
    "Password": "1234",
    "UserType": "student"
  },
  {
    "ID": "2",
    "LoginName": "smithtom",
    "Password": "4567",
    "UserType": "student"
  }
 ]
}
```

Get list of roads:

```
HTTP GET
https://track.sim.vuw.ac.nz/api/testuser/road_dir.json
```

```
{
 "Roads": [
  {
   "ID": "1",
   "Code": "SH1 South",
   "Type": "State Highway",
   "Section": "Kapiti",
   "Location": "Kapiti Interchange",
   "GPS": "Latitude 123, Longitude 456"
  }
 ]
}
```

Get list of projects:

```
HTTP GET
https://track.sim.vuw.ac.nz/api/<user>/project_dir.json
{
 "Projects": [
  {
   "ID": "1",
   "Road": "1",
   "Name": "Resurfacing",
   "Status": "Closed",
   "StartDate": "2019-12-12T00:00:00",
   "EndDate": "2019-12-14T00:00:00",
   "Contractor": "ABC Company",
   "Problems": [
    {
     "Author": "John D.",
     "Text": "Safety Issues"
    }
   ],
   "Comments": [
    {
     "Author": "John D.",
     "Text": "Work Completed."
    }
   ],
   "Works": [
    {
     "Type": "Resurfacing",
     "SubContractors": "Local Inc; Danny Do",
     "Status": "Done"
    }
   ]
  }
 ]
}
```

Delete a project:

```
HTTP DELETE
https://track.sim.vuw.ac.nz/api/testuser/delete.project.<projectid>.json
```

Delete a road:

```
HTTP DELETE
https://track.sim.vuw.ac.nz/api/testuser/delete.road.<projectid>.json
```

Update a project:

```
HTTP POST
https://track.sim.vuw.ac.nz/api/testuser/update.project.json
{
        "ID"        : "1",
        "Road"      : "2",
        "Name"      : "Resurfacing",
        "Status"    : "Closed",
        "StartDate" : "2019-12-12T00:00:00",
        "EndDate"   : "2019-12-14T00:00:00",
        "Contractor" : "ABC Company",
        "Problems"  :
        [
                { "Author" : "John D.", "Text" : "Safety Issues" }
        ],
        "Works"     :
        [
                { "Type" : "Resurfacing", "SubContractors" : "Local Inc; Danny Do", "Status" : "Done" }
        ],
        "Comments"  :
        [
                { "Author" : "John D.", "Text" : "Work Completed." }
        ]
}
```

NOTE: If "ID" does not exist in the server when you POST, a new file is created.

Update a road:

```
HTTP POST
https://track.sim.vuw.ac.nz/api/testuser/update.road.json
{
        "ID"      : "1",
        "Code"    : "SH1 South",
        "Type"    : "State Highway",
        "Section" : "Kapiti",
        "Location" : "Kapiti Interchange",
        "GPS"     : "Latitude 123, Longitude 456"
}
```

If "ID" does not exist in the server when you POST, a new entry is created.

Find/get a project:

```
HTTP GET
https://track.sim.vuw.ac.nz/api/testuser/project.{projectId}.json
{
 "ID": "1",
 "Road": "1",
 "Name": "Resurfacing",
 "Status": "Closed",
 "StartDate": "2019-12-12T00:00:00",
 "EndDate": "2019-12-14T00:00:00",
 "Contractor": "ABC Company",
 "Problems": [
  {
   "Author": "John D.",
   "Text": "Safety Issues"
  }
 ],
 "Comments": [
  {
   "Author": "John D.",
   "Text": "Work Completed."
  }
 ],
 "Works": [
  {
   "Type": "Resurfacing",
   "SubContractors": "Local Inc; Danny Do",
   "Status": "Done"
  }
 ]
}
```

Find/get a road:

```
HTTP GET
https://track.sim.vuw.ac.nz/api/testuser/road.{roadId}.json
{
 "ID": "1",
 "Code": "SH1 South",
 "Type": "State Highway",
 "Section": "Kapiti",
 "Location": "Kapiti Interchange",
 "GPS": "Latitude 123, Longitude 456"
}
```

How to organise the project

The whole project is organised using Trello.

Product backlog

The development of the product backlog requires understanding the product requirements:

1. Read carefully the project description and familiarise with the product **requirements**.
2. You will need to express **functional** requirements using **epics** and **user stories**.
3. Other **non-functional** requirements can be expressed using product **backlog items**.
4. Identify **epics** that reflect major functions the application will perform for the users. In general, each significant interaction (with a set of user inputs and associated functions) corresponds to an epic. Example: user login is an epic.
5. For each epic, identify the corresponding product backlog items (including in particular user stories). Example: user login has 1) input user name; 2) input password; 3) press login; 4) …
6. You do not need to follow a specific format to describe user stories. You just need to identify them in a simple, clear way and using verbs (e.g. user logs in the system). User stories are not literary writing, they are minimalistic and stereotyped phrases. Typical stories have the forms: "user [verb] [object]" or "user [type] [verb] [object]".
7. For each backlog item, create a definition of done.
8. You do not need many epics to describe the whole project, but you will have lots of stories. There is no fixed number for the number of stories or epics. Besides, you can create or delete them as you develop the project.

Scrum board

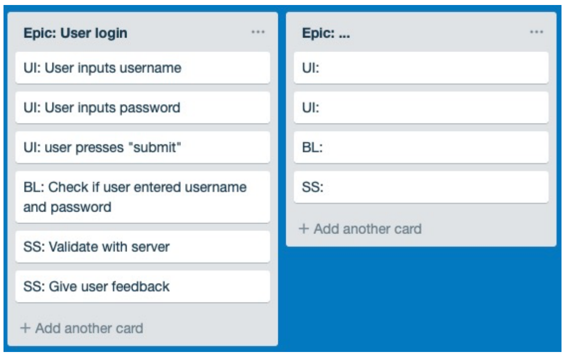
Using a scrum board requires slicing the project into several tasks:

1. There are two ways of slicing a project: vertical slicing and horizontal slicing.
2. Vertical slicing starts with an epic and seeks to identify the tasks that will implement the epic completely. Example: the user-login epic. This epic describes the user doing a login. With vertical slicing, we must define a set of tasks that develop everything related with a user completing the login, from the user input to checking the username/password.
3. Horizontal slicing also relates tasks to epics but divides tasks into three separate layers: user-interface (e.g. develop buttons and menus), business-logic (e.g. check if the user forgot the username or password), and server-side validation (e.g. check if the username/password are correct). Then, the development proceeds from layer to layer.
4. Vertical slicing is the preferred approach in the Agile world. However, for didactical reasons, we are going to adopt **horizontal slicing**.
5. First, for each epic, define tasks implementing the user-interface (UI) layer. Example: develop the login box, password box, and submit button.
6. Second, for each epic, define tasks implementing the business logic (BL), i.e. processing the users' inputs. Example: check that the password and login fields have been filled up.
7. Finally, for each epic, define tasks implementing the server-side (SS) functionality. Example: validate username/password on the server.
8. Tasks are always grouped by epic.
9. Within epics, organise tasks according to the development status: waiting, in progress, completed.

Using Trello

There are different ways to manage the product backlog in Trello. This is just an example.

1. Trello uses concepts such as **lists**, **cards** and **tasks**.
2. Turn the product backlog into a list.
3. The product backlog has a list of epics.
4. Each epic has a list of user stories.
5. Add acceptance criteria at the end of an epic. Create a card named acceptance criteria. Add tasks describing each acceptance criteria. These will be used to check if the epic has been properly implemented or not.
6. Epics should be listed by priority (more important first).
7. The picture below shows a product backlog with two epics (the acceptance criteria are missing).

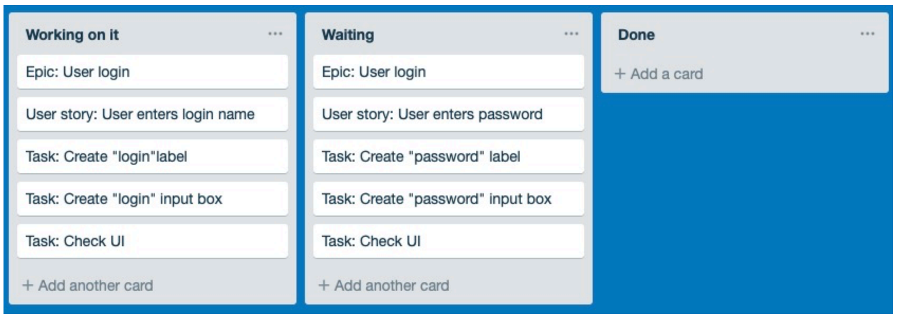


Epic: User login
- UI: User inputs username
- UI: User inputs password
- UI: user presses "submit"
- BL: Check if user entered username and password
- SS: Validate with server
- SS: Give user feedback
- + Add another card

Epic: ...
- UI:
- UI:
- BL:
- SS:
- + Add another card

There are also different ways to manage the scrum board in Trello. This is just an example.

- The scrum board is a tactical tool – use it to manage your work, not to satisfy any managerial theory.
- The scrum board identifies which sprints have to be done **this** week (but, as you move the project forward, may also include the completed sprints).
- You do not need to define all sprints from weeks 1 to 12. You may start with just week 1 and develop as you go.
- Divide sprints in categories, e.g. "current week", "next week" and "done".
- Copy some user stories from the product backlog into the scrum board and put them in the "current sprint". These are the user stories under development this week.
- Each sprint must include the list of tasks that have to be done.
- When starting the project, pick up a small number of backlog items that are easy to develop and add then to the sprint. We suggest you start with user stories related to the user-interface (UI). For instance, you can start with the user login story.
- After adding the backlog items to the sprint, identify the list of tasks required to implement them. These tasks describe what you have to do to finish the sprint. For instance, to implement the login user-interface, you need to create a "login" label and a text box.
- The picture below shows a scrum board with the sprint the team is working on this week. The sprint identifies the epic and user stories the team is working on, as well as the tasks that have to be fulfilled.



Working on it
- Epic: User login
- User story: User enters login name
- Task: Create "login"label
- Task: Create "login" input box
- Task: Check UI
- + Add another card

Waiting
- Epic: User login
- User story: User enters password
- Task: Create "password" label
- Task: Create "password" input box
- Task: Check UI
- + Add another card

Done
- + Add a card

Identify who does what

- This is important for marking.
- You can identify responsibilities either for sprints or tasks.
- For instance, you can define that a group member will complete a certain sprint by taking responsibility for all assigned tasks.
- Or you can define that a sprint will be done by the team, and then identify who is responsible for each task.

Additional comments about the project organisation

1. Do not forget that you will work with weekly time boxes, which means that you need to finish a sprint by the end of the week.
2. Use the scrum board to actively manage the sprints. The scrum board shows what tasks have to be done to finish a sprint. As soon as you finish a task, tick off the corresponding box. Track your progress by checking the tick boxes.
3. Note that the scrum board is supposed to deal with reality, not with a fictional project. So, if you need to change anything, just do it – you can change a user story, sprint, task, etc.
4. Initially, students tend to be too ambitious and cannot finish sprints by the end of the week. Learn to calibrate what you can deliver each week and adjust the scrum board and product backlog accordingly.
5. By the end of each week, the group should review if the sprints have been completed or not.
6. The scrum board should be used by the group to track how the project is evolving and discuss problems, especially delays. In many software houses these discussions are done every day, in the morning (daily scrum meeting).
7. Time boxes should never be extended. Define a specific time for starting, finishing and reviewing the sprints, e.g. Monday and Sunday.
8. Unfinished sprints should raise significant attention, as they indicate something is wrong with the project. Discuss the issues.

Code development

These comments are related with how to use Plunker:

1. It is better to work with Plunker if you sign-in using Github.
2. Plunker offers you the option to build plunks. Consider each plunk an iteration of you project, which creates a direct correspondence with the sprints.
3. Share your plunks with your team.
4. You can share plunks by sharing webpage links.
5. Do not make your plunks public until the end of the course. Otherwise other students will benefit from your hard work. **You have been warned!**
6. Work with Plunker in **very small iterations**. If something works, save it in a plunk and fork a new version. Constantly forking plunks is highly recommended. Use the "description" and "tags" fields to describe each forked version, e.g. noting what is working or not.
7. Note that Plunker saves versions of everything you do on a plunk – but you need to press the save button!
8. Plunker has a menu on the right-hand side that allows you to add external libraries to your project. Initially, you only need to add angular.js to the project. We will be working with version 1.x. Do **not** add 2.x libraries. Depending on the functionalities you would like to use, you can add libraries from versions 1.4 to 1.6. Consider using the highest 1.x version available (1.6.x).
9. Note that Plunker adds a lot of code to the project, which people like to call boilerplate. Do not mess with that code and especially do not remove it!