TEST AND INTEGRATION PLAN

MEng Year 3

Department of Electronics

University of York

Software Engineering Group 4

CODEV

Document Control

Version	Modified By	Date	Section Modified	Remarks
1.0	N Billis	2/3/20	All	Initial Document Creation
1.1	N Billis	5/3/20	All	Updating All parts of the document
1.2	Joe Butler	11/3/20	Test Criteria Iteration 1 User Story Testing	Updated A3,5,4 &11
1.3	L Newman, B Anderson, N Billis, James P	11/3/20	All	Updated all parts mainly test criteria

CODEV

Table of Contents

1. Introduction

1.1. Purpose and Scope

The goal of the Test and Integration plan is to ensure that various modules that CoDev are purchasing from other companies are correctly tested and work well with our product together with modules that are producing 'in house'. All of these modules are considered "Black Boxes" to each other and need to be adequately tested in order to ensure they integrate into the main product effectively. CoDev are purchasing two modules from FixPack for both Graphics and Digitised Sounds.

This document aims to cover:

- Integration Strategy; rules and procedures in adding our modules of the final product and how they will fit together effectively.
- Test Strategy; rules that the testing will be based on together with how the tests will be performed and the processes to identify and report defects and how to fix and implement these fixes.

The application ScranPlan aims to help users batch cook meals on a budget. The user can select and collate recipes that they plan to cook for that week and create an in app shopping list that is constructed from the ingredients that the user specified they do not own. With the added ability of creating a personalised profile, the user will be able to tailor the application to their tastes so that they can filter out any of the recipes that don't suit, save their favourite recipes and share them with their friends.

The whole project has been split into iterations in order to make things slightly clearer and break up the testing and integration for the entire application.

2. Test Strategy

2.1. Testing outcomes

CoDev undergo testing for their applications to get the following outcomes:

- Clean Code (Code which isn't full of unused functions and works well)
- Error free and functional classes
- Confidence in the code that is written

2.2. Testing Types

In this section the different types of testing methodologies at CoDev are described together with the types of testing.

2.2.1. Automatic Testing

Many of the elements in our projects will be tested automatically using the tools built into Android Studio IDE. They will automatically be verified as passing or failing without any user input.

2.2.2. Semi-Automated Testing

Semi-Automatic testing is similar to automatic testing but the verification of the results are often done manually. This together with manual tests are not CoDev's prefered test method.

2.2.3. Manual Testing

When automatic or semi-automated testing cannot be performed there will be manual testing, this will be kept to the absolute minimum. Any manual testing will have to be verified by the programmer and will be recorded in test reports. Manual testing will **only** be used when automatic or semi-automatic testing cannot be used.

2.2.4. Black-Box

Some of the tests that will be performed on user stories are black box tests, this means that to the tester they cannot see the inner workings of the story. They are just looking at the output and input to the functions are not looking at the inside workings. Much of the UI testing (using Espresso framework) will be tested in this way, where the testing will not look at the inner workings but instead will verify if the application functions in the correct way.

2.2.5. White-Box

The white box testing is where the coder is aware of the internal data structure and is testing the inner workings of the module. Testing in this way allows much more detailed testing into the individual functions of the user stories and finds more errors that might not be found during usual black box testing.

2.2.6. Grey-Box

In addition to white-box and black-box testing CoDev will be undertaking some grey-box testing, this is when in testing issues have been found or the code is being modified later in the development process. This is what a lot of CoDev's regression testing is, due to the coders already knowing the inner working and tests being written for that. Gray-box testing is testing the aspects that have changed and are not constrained to only the module as a whole or the inner workings.

2.3. Testing Method Overview

2.3.1. JUnit Testing (Unit Testing)

Unit testing core to CoDevs testing philosophy, it is testing each module independently of the others and ensuring that the functions exhibit the expected outputs. Unit testing is done to ensure code is functional and error-free and to present confidence in all of the modules.

There are two main methods to unit testing, class under test and system under test. CoDev will be mainly focusing on class under test when talking about unit testing. When unit testing classes CoDev will be testing the class and nothing else. All of the dependencies are 'faked' and tested without the real external dependencies.

2.3.2. Espresso Testing (Instrumentation Testing)

In addition to Unit Testing another key testing method is UI testing. For this we will be using the Espresso Framework. This allows CoDev to simulate users clicking on buttons in the User Interface or entering data. The espresso framework allows CoDev to produce reliable, automated, reproducible tests. Espresso testing is an example of instrumentation testing and runs on the device or emulator.

Instrumentation testing will be the second highest number of tests, with manual UI tests being the lowest amount of testing. Most of the instrumental testing will be automatic and executed in the IDE.

2.3.3. Travis CI (Build Testing)

In addition to all of these testing methods we use Travis CI [1]. Travis CI is a continuous integration tool which allows us as developers to integrate code and get early warnings if the code fails to build. Due to the agile methodology we are following developers may be working on different user stories or branches at the same time, Travis CI allows us to detect any integration problems early.

3. Test Criteria

Note. All Test are subject to change during the development process

3.1. Iteration 1

Outcomes:

- *Core Features of the Application*
- *Basic usability*

3.1.1. User Story Testing

User Story	Test Description	Test Objective	Success Criteria
Register Account (A1)	Create invalid account details	This test aims to check emails or/and passwords not reaching requirements don't register accounts	No account is created and an error message is shown to the user.
	Check username and password fields exist	This test aims to ensure that the fields for the username and password exist for creating an account	Fields are not null and exist in the app
	Editing password and/or username fields	This test aims to ensure that the fields can be edited	Field are edited
Login to account (A2)	Login with valid account	This test aims to ensure the login path flow works	App logs in successfully
	Login with invalid account details	This test aims to ensure that users cannot login with invalid account details	Application does not log in successfully
	Login screen loads	This test ensures that the login screen displays when the application is launched	Login screen displayed with no errors.
User Customisation (C20)	If the user hasn't used the app before or if they have not submitted their preferences the preference screen loads	This test ensures that the page is only displayed when the user has not submitted their preferences. Once they have, they won't seen the screen again	This is tested by simulating a new user registering for the first time and ensuring the screen is only shown until the user submits the preferences

	Verify that the preferences are stored in the user's profile	This test ensures that the preferences that are set on the initial screen are still checked within the users profile. Also ensures that the preferences are stored on firebase	This is checked manually by referencing firebase and checking the inputted preferences and the preferences stored on firebase are the same
	Verify the correct information is asked for and all elements can be selected	This test ensures that all the filters are displayed on the screen when the user sets them. It also ensures there isn't a limit of the preferences that can be selected	This is checked through espresso testing. Confirming that all elements are present and all boxes are able to select
Netflix Style Scrolling (C1)	Check that recipes can be viewed clearly on screen	This test ensures that all recipes can be seen clearly on a mobile device and that no content is loss due to being unable to see it	This is checked by graphical testing within the simulator and if it passes all virtual tests on the kanban. If all items are clear and information is what's expected
	verify that the scrolling is smooth and controlled	This test ensures the netflix style scroll isn't jumpy but works well clearly and the speed can be controlled by the swiping	Checked manually and ensuring behaves are expected
	Check that all recipes are available on the main home screen	This test ensures that the user can see all recipes that are on firebase	Checked by all recipes appearing with no errors
Main Screen Tabs (C2)	Check that all screens can be viewed by a swiping action	This test ensures that the user can access every tab by using a swiping action can can see the contents of that page	Tested manually by checking all pages are accessible.
	Ensure the actions takes the user directly to the next page	This test ensures that there isn't a delay and that the correct page is displayed	Graphically tested checking everything is displayed when an action is performed
View Recipe Contents (C3)	Ensure when recipe is selected the meal	This test ensures that when a recipe is selected	Tested through graphical testing within the simulator and

	is shown to the user immediately	a pop up opens with the recipe and details without a delay	checking it passes all visual tests on the Kanban.
	Check that all of the information is displayed correctly to the user	This test ensures that the recipe that is selected is the one shown to the user in the pop up	Graphical testing and ensuring all the details are correct for the selected recipe
Dietary Filters (C6)	Verify checkboxes in settings and start up can be modified	Tests functionality of check boxes to ensure users can translate their preferences into boolean variables to be saved to the server.	Check through espresso testing on in the settings menu and ensure these are changed.
	Check that these settings are saved	Test that these actions translate into the correct boolean variables on the FIrestore database.	Changing them and physically checking them. This is also tested in the settings with alteration and reopening the application in espresso on the settings page.
	Check that these settings carry across to search options	Changing personal dietary settings within the settings page and ensuring these are populated in search preferences by default.	Physically changing these settings within the settings page and checking this translates within the search filters tab.
Sort Recipes (C7)	Check that check boxes within search preferences are added to the search query	Changing sort parameters and checking results that are returned reflect these changes	Changing to the likes of vegetarian within search preferences and searching for ‘bacon’ to ensure nothing is returned that contains meat.
	Ensure user preferences are added to the queries within the home page	Changing user preferences within settings and ensuring we get relevant items returned.	Changing to vegetarian within user preferences and ensure that no meat based products populate on the home page
	Ensure all search possibilities are accounted for within the Firestore composite queries with the correct sorting.	Check that queries return correct results when adding a range of different queries	Changing the search parameters to different options and ensure that no errors are encountered from the Firebase for queries that aren't already set up.

Search Recipes by keywords (C18)	Ensure all search possibilities are accounted for within the Firestore composite queries with the keyword searching parameters.	Check that queries return correct results when adding a range of different queries	Changing the search parameters to different options and ensure that no errors are encountered from the Firebase for queries that aren't already set up.
	Ensure relevant results are returned from searches	Check that the results returned match that of the search	Send a search for bacon within ingredients and check that bacon items are returned.
Change Password (A3)	Verify the change password dialog box is displayed	This test ensures that all valid information in the dialog box is displayed to the user	Checked through Espresso the dialog box must open and have 3 entry fields, old password, new password and new password (repeated)
	Simulate changing a users password in app	This test ensures that a password can be changed for the users account from the edit profile screen.	Password changes for the user and upon re-login this new password is required. (Tested on multiple accounts)
	Verify a successful password reset	This test ensures that a prompt is sent to the user to let them know their password has changed.	Android 'Toast' messages tested using Espresso must match with what is expected to be returned. IE a notification of a successful or unsuccessful password reset.
	Simulate sending a password reset request via email	This test confirms we can successfully send an email with the Firebase password reset link to the valid associated users email.	The link is clicked and the password is changed upon re-login.
	Test for invalid inputs	This test ensures that whatever is inputted into the password reset text input box is valid.	Espresso tests using returns from 'Toast' messages displayed to the user must be conducted to confirm the following: - Input field isn't blank - Password is longer than 6 characters in length - New password does not equal existing one

Retrieve User Info (A4)	Verify the profile settings screen loads	This test ensures the Profile settings activity used to edit a users preferences is loaded correctly with properly.	All required UI elements, including text input boxes, checkboxes, switches etc.. all need to be checked that they appear and are editable on the current device.
	Simulate creating fake user profile data	This test is included to make sure user data is properly stored and retrieved from a local class designed to hold the local users data as a serializable object.	Test associated user data, privacy options and preferences can be assigned through a fake test initialization and retrieved through valid getter/setter methods.
	Verify users profile information is correctly retrieved	Test all implemented associated XML Firebase profile information is returned and displayed (with the exception of a users email and password)	After editing the users associated data and saving it, this data can be retrieved upon a re-login. (for multiple accounts)
	Ensure all data is synced correctly	This test allows us to make sure no data is missed or isn't properly synced between the Firebase version of our user and the local (cached) version.	This is checked manually by editing user data for multiple accounts and checking this matches our cached (Serializable) data.
Delete User (A5)	Verify the delete profile dialog box is displayed	This test ensures that all valid information in the dialog box is displayed to the user	Checked through Espresso the dialog box must open and display a password confirmation Text input box which is editable.
	Verify users accounts can be deleted	This test ensures that a with a valid password all associated user data is deleted off Firebase.	All associated information must be confirmed manually to have been removed on Firebase and all collections of cached data are set to null. All activity screens are closed and destroyed. (Repeated for multiple accounts)

	Send feedback to the user on the current state of account deletion	This test ensures that a prompt is sent to the user to let them know the current status of there account deletion	Android 'Toast' messages tested using Espresso must match with what is expected to be returned. IE a notification of a successful or unsuccessful profile deletion
	Test for invalid password	This test ensures a user can't accidentally delete their own profile because the password confirmation box checks are not implemented correctly	Check the password the user inputs matches their current password by checking through Firebase authentication and detecting this through Espresso through valid/invalid 'Toast' return messages displayed to the user.
	Test for invalid inputs	This test ensures that whatever is inputted into the delete profile text input box is valid.	Espresso tests using returns from 'Toast' messages displayed to the user must be conducted to confirm the following: - Input field isn't blank - Password is longer than 6 characters in length
Profile Customisation (A11)	Verify the public profile screen loads	This test ensures all UI elements for the Public Profile activity used to show any valid users profile are displayed properly to the user.	All required UI elements, including checkboxes (non editable) are present on the screen for multiple devices.
	Check that a user can successfully hide parts of their own profile information	This test is essential in making sure private data is not in-correctly displayed based on the users own preferences.	Manual tests are used to ensure for every individual privacy setting the associated public profile is updated accordingly. For e.g. when hiding a username the username should by default display 'Anonymous' to other users instead. (Repeated for multiple accounts)
Enter Presentation (B1)	By clicking a button within the recipe information page, the XML	The test ensures that the XML presentation is successfully retrieved and displayed	No errors on button click and XML presentation successfully displays with the correct recipe information

	presentation will load		
Move within presentation (B2)	Using the buttons within the slide, the user can move to the next or previous slide	Test ensures each slide within the presentation loads correctly	The correct information is displayed when moving to each slide with no errors
View Presentation steps (B3)	Using a drop down menu, the user can navigate to a slide of their choosing	Test ensures drop down menu works correctly and user is able to traverse the presentation in any order	The correct information is displayed when moving to each slide with no errors
Comments to steps (B6)	Verify that the slide shows comments that have been added	Test ensures that the correct comments can be retrieved and shown on the correct slide	Comments are shown on the correct slide
	Verify that the slide shows how many comments have been added to the slide	Test ensures that the correct information can be retrieved as to how many comments have been stored for that particular slide	Correct number of comments are retrieved and displayed on the correct slide
	Verify that the comments are stored correctly and then can be safely retrieved on a certain slide	Test ensures that the comment data can be stored in the correct order and then retrieved to be displayed	Data is safely stored and retrieved then displayed in order of whichever sorting method is being implemented
Enter Presentation from meal planner (B4)	Ensure 'let's cook' button is only available when a plan is set up	Test that button only shows when meal plan set up	Check if button is available when the meal plan is set up and not when it is not.
	Ensure button triggers a meal preparation event	Test the button press of the button	On press of the button, a meal preparation event is triggered
	Verify if meal preparation event triggers a menu that shows user all meals from the plan	Test that meal preparation even gives user options from meal planner	The correct meal options are given to the user

	If meal finished cooking, meal taken out of cooking list and added to cooked list to trigger food life notifications	Test that meals taken from plan and notifications added.	Meal removed from the cook list and added to the cooked list with notifications set.
Create Meal Plan (C4)	Check that the meal plan created is saved to individual profiles	Ensure that when a meal plan is created it is saved to the user	Creating a meal plan enables user to use lets cook button and other meal plan features
	Check that the meal plan can be edited and the shopping list updates.	Check that adding more meals to the meal plan updates all other variables	Check that shopping list is updated with ingredients from further added meals
	Ensure that all ingredients are included in the list and duplicate ingredients are added together rather than separate ingredients	Check that when adding two meals with the same ingredients, these are added together and not duplicated	There are no duplicate ingredients
Build Shopping List (C5)	Check that the correct ingredients are adding to the list	Test ensures that the ingredients in the recipe are the ones shown on the shopping list	Manual testing that checks the ingredients added to the list are the same as the ones stored for the recipe in firebase
	Ensure the shopping list is saved and available to view.	Checks that the user is able to view the shopping list on their profile when they want to access it	Graphical testing to ensure the shopping list is clear and can be viewed at any time
Display typical expiry dates (C14)	Ensure that each recipe has the correct information for health and safety reasons	Test ensures that the correct information is displayed to the user	Manual testing that ensures the information displayed is the same as the information in the firebase and is the same as information available on trusted health and safety sites
	simulate selecting a recipe to check the information is present and correct	Tests that the information regarding how long the recipe will last is displayed correctly and visible to the user	manual testing ensuring information is displayed when the user selects

	Simulate selecting an ingredient and verifying the expiry date is listed and correct	Tests that the information regarding how long each individual ingredient is displayed correctly and visible to the user	Manual testing ensuring information is displayed when the user selects
Select Recipes for Breakfast, Lunch, or Dinner (C17)	Ensure each recipe categorically states the timeframe it is usually cooked in	This tests ensures that each recipe has a timeframe that it fits into and is stored in that category	Test that every recipe within firebase has a timecode present
	Simulate checking different options for recipes and verifying the recipes that display are of the type(s) selected	Ensures that when a time frame is selected, the appropriate recipes are shown	Manual graphical testing ensures that when an option is chosen that group of recipes is shown. No errors occur
Alerts for expiring food (C21)	Once a meal is cooked from cooked list check it has been removed from the to cook list and been added to the cooked list	Test ensures that when a user has cooked the meal it is moved to the cooked list and completely removed from the other	Graphical testing to ensure the recipe has moved lists and is only displayed on one list at a time
	Check the fridge and freezer life times are present in each recipe	Test ensures the relevant information regarding storage can be viewed on each recipe	Ensure recipes stored on firebase have the categories for the life expiration and is displayed on each recipe
	Ensure that these times are applied to timers and saved to the user profile	Tests that all times that items go out of date are stored	Test my manually inputting recipes and ensuring that the expiration is displayed and can be view by user
	Ensure once these timers expire a cloud function is triggered to alert user	Test that notifications are sent out when items go out of date	Manual test that the correct notification is sent out at the correct time
Recommended Reheat instructions (C22)	Check each recipe has this information given to the user once the cooking phase is complete	Test that when the cooking is complete the information is available for the user to see	Espresso tests to ensure all data is able to been seen and manual tests checking that all information on firebase and on the app match

	Ensure information page can be opened and closed without interrupting the functionality of the rest of the application	Ensures that the page opens without causing errors or issues in any other part of the code	Manual testing that page opens with no errors
Number of Portions per Recipe (C34)	Ability to change the amount of ingredients in accordance with the portion size	This test ensures that the ingredients change the correct amount for the portion size	Manual test checking results and correct and that there are no errors
	Changing the portion size should change anything else about the recipe	This test ensures the function works without any problems	Changing portion causes no errors or issues when testing
Digitised Sounds (Purchased Module)	Ability to import a sound file by either searching locally or by URL.	Check that an audio file can be played via local directory or by URL	Audio sound plays with no errors
	Ability to play the sound at the click of a button and to stop the sound with another click	Ensure that at the click of a button an audio track can be played	When the user clicks a button the audio track plays with no errors
	Ability to play the sound through a written function within the Java code.	Ensure that when a function is written, the application automatically plays the sound	Application can play the sound with no error when function calls audio player
Graphics (Purchased Module)	Ability to draw shapes such as circles, rectangles and triangles	Test ensures that each shape can be successfully constructed	Each shape can be called and placed on screen successfully with no errors
	Change the size of said shapes	Test ensures the ability to draw the shapes in different sizes	Each shape can be generated in different sizes without error

	Ability fill shapes with solid colour or shading	Test ensures that each shape can be filled with colour or shading	Each shape can be filled with a variety of different colours with no errors occuring
	Ability to draw lines	Test ensures that lines can be successfully drawn anywhere on screen	A line can be drawn giving different coordinates and will successfully appear on screen
	Set a duration for the graphics (Start time, end time)	Test ensures that graphics can be called at certain times and be removed at another	Graphics appear and disappear successfully given a start and end time

3.2. Iteration 2

Outcomes:

- *Social Aspects of the Applications*
- *Extended features*

3.2.1. User Story Testing

User Story	Test Description	Test Objective	Success Criteria
Post, Comment and Like (C16)	Ensure that comments can be shown objectively in the forum	Checking that comments can be left	Comment successfully saves to the Firestore
	Simulate user forum features and verify they are shown for other users on the forum	Check other users can view these comments	Comments correctly downloaded from Firestore
Send friends messages & recipes in the app (C26)	Ensure that any link sent goes to the correct recipe	Sent recipes bring up the correct recipe information	User sent to the correct recipe
	Ensure all messages sent are saved to both users collection so each user streams messages from their own collection	Makes sure all users involved see the messages	Messages on both users screens
	Check that messages sent are saved to both collection	Duplication through messages across collections	Messages are saved to both collections
	Check any messages deleted are deleted from both collections	Messages deleted by the application are deleted everywhere, individuals deleting the messages only allows that user not to be able to see	Data deleted from both collections when removed by app and single collection when deleted by user
	Ensure each user streams the messages from their own collection	Messages viewed by each user are taken from their own collection	Only messages on own collection downloaded to that user

Add and Invite Friends (C27)	Check button to view friends list registers a response on the client side	Downloading friends returns info from server	Data returned from server successfully
	When users click on this button, they are taken to the friends page	Checking friends list button operates	Client taken to friends list on button click
	This page populates with all of the users friends	Requesting list of friends returned that list	List of friends populates client side
	There is a button at the top to add friends that brings up a menu to enter an email address	Add friend button triggers an event	Event successfully triggered
	The email enters and is sent to the server to check if there are any users registered to this email	Request to add friend with inputted string sent to server	Server triggers an add friend event
	Client sends back a public user profile if there is a user profile linked to email entered	Checks that the client is returned public info on their request to add a new friend if friend exists	Client returned public info on their request
	If there is no user attached to the email, then an option to invite the user is given to the user.	Checks if there is a null returned from adding a friend and a new event is triggered to send an email to requested address	Response to null triggered and email set up
	The user has the option to add a personal message to the invitational email	When requesting to invite via email, checks personal message attached	Personal message attached to email request

	If a user selects to send email, the email is sent to the address.	Check that the requested email is actually sent	Email sent to requested email
	Email contains valid link to download the application	Checked link provided by email is valid	Link taking user to download app
	Email contains their personal message	Checks personal message present in email	Correct display of personal message
	There is a button to add user as a friend within each public user profile	Checks buttons operation to add friend section	Buttons launches add friend section
	Button sends notification to individual to add the user that pressed it as a friend	Checks notification sent to email requested	Notification successfully sent to user with given email
See trending recipes that have high star rating (C28)	Trending list is viewable from main menu	Check that user can see trending recipes	Trending recipes populate
	Filter button opens a menu, and this saves options client side but filters the request sent server side	Filter button opens up a new menu that can manipulate queries	New menu pops up on screen and filters given are applied to queries
	Pull to refresh	Pulling the list bring the newest query results	Refreshed recipes populate on screen
	Scrolling down the page to the end loads more recipes in from the server creating a potentially infinite scroll but limits	Test that more documents are loaded one user reaches the end	New recipes populate once getting to the end of the scroll view

	document downloads to 3 per load		
Social feed (C29)	News feed available from main menu	User can navigate to news feed	Successfully navigating from home page
	Page populates with posts from friend's activity within the app	Checks that users friends recipes are displayed	Successfully displaying the correct data from the server
	Ensure reviews, cooked meals and ratings populates here	Checking that all the relevant posts are displayed here	Successfully displaying all requested data from server
	Only downloads 5 items as a time & scroll to end downloads more posts from the server	Check that only 5 items are downloaded and there is an endless scroll	5 loaded from server successfully at a time and once the user reaches the end, the next 5 are displayed
	Pull down to refresh operates	Once the user pulls down, the query is run again to return the latest data	Newest data downloaded from server given a user swipe down
Share cooking tips (C31)	Check that when voting for the best answer occurs that the answers are ordered by highest vote	Comments ordered by highest votes	Highest voted comments at the top
	Ensure that questions and all answers save to the server and display correctly with all recipes	Check comments are saved and displayed correctly	Comments from server populate on the client
Display similar recipes by ingredient (C13)	Ensure the recipes that are searched by ingredients display the correct ingredients	Check that searching for ingredients gets the correct ingredients	Searches return recipes with searched recipes
Track eating habits (A13)	Check data can be inputted and stored privately	Ensure calories of meals cooked is recorded	Checks that this data is recorded

	Ensure data is saved and not lost and can show progress through the week	Data saved to Firestore	Data populated on Firestore
Target weight and calorie intake (C32)	Ensure the healthy living plan calculates BMI	Check that it takes user details and calculates BMI	Ensure the correct value is calculated
	Simulate making a target weight plan	Ensure user can input their own health goals	Check that these goals are saved the the server
	Check that calories can be tracked for the user and can be tracked for users weekly calorie goal	Ensure that meals cooked and eaten go into the tracker	Tracker populates with calories
	Check users get notifications about their goals and when they go over or under their goals	Check that once a limit has been gone over, an event is triggered	User receives notification telling them they have gone over on calories
User created recipes in the app (A12)	Check that all functionality of creating a recipe works	Recipes that are made, an xml is generated	check that recipe is made
	Ensure the recipe is saved to the server	Check that the recipe is saved to the server	check that XML is saved to server along with generating a document is on the Firestore
	Ensure that others can view the recipe and add additional comments	Ensure the recipe is downloaded for other users	As another user, recipe can be viewed
Gold Membership (A9)	Check that once a user has paid, this is saved with a date for which they have saved until	Check date paid until added to account	New date populates on Firestore
	Ensure all extra features are only available to users that have a date for gold	Ensure that premium features are only viewable with users with a date that is not in the past	Premium user able to view premium content, free is not

	membership in the future		
	Ensure that all new free users have a date set from the moment they make the account for gold membership	Ensure that the date is immediately updated and downloaded	Check that local and database info is updated immediately from payment confirmation
	Use: Test Google Play Billing to test paying	Check payments work and are recorded	Successful payments
Removal of Advertisements for Gold Members (C19)	Ensure advertisements are displayed in a user friendly fashion and do not move other elements out of place	Check date is in the past before displaying adverts	Check that adverst only display for free users
Price and nutritional value of recipes (C12)	Ensure all information is present on each meal	Check information on recipe page	Information present
	Check a supermarkets API to find the typical price for an ingredient, add all the ingredients in the meal and display the total typical price to user	Check that API is downloading up to date information	Information present

3.3. Iteration 3

Outcomes:

- *Final snagging and polishing*

3.3.1. User Story Testing

User Story	Test Description	Test Objective	Success Criteria
Change the unit of measurements for ingredients (C10)	Check the function that changes the measurements works correctly	Test to change unit in a list of measurements	Change to the wanted unit of measurement
	Ensure all units are displayed correctly	Test to make sure that unit shows correctly in several recipes including the changed ingredients	The unit in all chosen recipes edited correctly
Receive chef stars from ratings (A8)	Simulate multiple devices & therefore users rating a single user account to observe its effect on the chef star rating. The effects should fit the result of the simulated averaging algorithm	Test to ensure that rating a user account takes hold and effects the chef star rating accordingly	The rating will change in accordance to that of the ratings being given to that particular account
	Simulate testing the averaging algorithm used to rate users' recipes.	Test to ensure that an average of all of the ratings is applied and shown on the users profile	Average rating is displayed on the user profile
Rate recipes (C9)	Verify recipe rating is stored and saved for each user and the average rating is displayed to everybody	Test to ensure that the users rating is stored for the recipe and that when multiple ratings have been added, an average has been taken.	Rating can be stored with no error and when averaged it can be retrieved with correct value
	Ensure the rating is available to see on each recipe	Test to ensure that all recipes have a visible rating on viewing	rating can be successfully retrieved for each recipe

Add comments and reviews on recipes (C11)	Ensure each comment gets added along with information about who added the comment	Ensure that the comment is stored successfully on the backend the user who created the post has their user ID assigned to that post	Comment is successfully saved and the user ID is attached
	Check that the comment is only available on the selected recipe	Ensure that the comment is assigned to the recipe ID and will not get duplicated to other recipes	Comments left on a recipe can be viewed on that recipe and on no other
	check the comments are fully removed if the user chooses	Ensure that if a user clicks delete, the corresponding comment has been removed from the backend	Once comment has been deleted there should be no data associated with that comment and no trace can be retrieved
Reporting of content (C23)	Check that all sections where there is the ability for users to comment or write something, there is the ability for other users to report	Ensure that on each comment created there is an icon that allows another using to send a report	Report icon appears on every comment that is posted
	Ensure that action is taken immediately to tackle the issue and take appropriate action	Ensure that reported content is logged with the ID of the user who posted the comment.	Alert and user ID is logged and notification sent to developers
	Ensure that content is not removed and lost when it has been wrongfully reported	Ensure data is not removed from backend until approved	Data can still be retrieved until checked. Developers can resubmit or delete.
Post questions about recipes (C30)	Check that when voting for the best answer occurs that the answers are ordered by highest vote	Ensure that comments have incremental values that are compared when ordering posts	Posts are displayed in correct order depending on amount of votes
	Ensure that questions and all answers save	Test to ensure that data is saved on the	Data can be correctly retrieved and

	to the server and display correctly with all recipes	firebase and can be retrieved correctly with the correct recipe ID	assigned to the correct recipe
Favorite recipes (C8)	Check that the recipes the user adds are added and saved to their liked recipes	Make sure all recipes added by the user are shown in 'favourite recipe' page	The favourite recipes can be seen by the user
	Ensure it is easy and clear for the user to see the recipes and like them	Click the add/delete favourite recipe button for a recipe. If the recipe is already a favourite recipe, it'll be not a favourite one and vice versa. Move to 'favourite recipe' page to check it	The favourite recipe can be removed from the 'favourite recipe' page after clicking the button and vice versa.
Substitute meals disliked in meal planner (C15)	Ensure that the substitution meals differ from the deleted meals	Compare the integrations and cooking methods between the deleted recipe and substitution ones. Make sure that they are different.	The contents of the deleted recipe and substitution ones are quite different.
	Ensure that the substitution meals have almost the same calories as the meal before	Compare the calories of the deleted recipe and substitution ones. Make sure calories of them are almost the same.	The calories of the deleted recipe and substitution ones equal each other approximately after comparison
	Simulate deleting one meal in the week plan and the substitution choices will appear	Delete one of the meals in the week plan. Make sure that the list of substitution choices will appear	The substitution choices appear after deleting one of the meals in the week plan
Notifications (A6)	Simulate retrieving a notification from the server and displaying it to the users device	Retrieve a notification from the server. Make sure the user can receive it in his own device	Get the notification in the user device after retrieving it from the server
	Ensure that the notification is	Test to click the notification and the	The relevant recipe can appear correctly

	retrieved correctly and is of a valid recipe	relevant recipe appear	after clicking the notification
	Implement cloud functions for sending notifications to individual users	Test to use cloud functions of firebase to send notifications to the user	Users can receive the notification via cloud functions
	Produce a small survey or log general opinions over time for all existing team members who are testing the app with notifications enabled, if the notifications received are relevant to their experience	Make sure that users use the app with notifications enabled. They can receive the notifications relevant to their experience	Users can receive the interesting notifications based on the experience about using this app
Notifications Preferences (A7)	Simulate sending the notification to the users device and Check that the notification is received or isn't received based on the users preferences	Test to send the notifications to users based on the user preference setting. Make sure the notification is correctly relevant to the user preference	The user can receive the wanted notification if they enable the notification
	Verify that when the user enables/disables notifications the appropriate XML server entry is changed from "Enabled" to "Disabled" or vice versa	Test to choose to enable/disable notifications. The notification will start to work or not appear	The user can choose to receive the notification or not
Profile Badges (A10)	Check there are set standards that once a user has met they automatically receive badges	Test to check that badges will be given to users once a given criteria is met	Users will successfully receive a badge on completion of certain milestones
	Ensure the badges appear in the user's profile and are available to see	Test to ensure the profile gets correct data from the back end and displays the correct badges	Badges are shown on the correct user profiles successfully without error

	Ensure that all users receive the same badges for the same achievements	Test to ensure multiple accounts can log the same milestones and receive the correct data	All users gain badges when meeting the same milestones
Custom icons next to ingredients when viewing recipe (A14)	Ensure that the correct icon is attached to the correct ingredient	Test to ensure that enter into several recipes and the icons suit the ingredients	All the icons suit the ingredients in the chosen recipes
	Ensure that all ingredients have an icon	Test to ensure that enter into several recipes and all ingredients have their own icons	All ingredients in the chosen recipes have their own correct icons
Promoted similar recipes (B5)	Simulate to click the 'similar recipe' button this button brings up new menu populated with meals share ingredients with it	Test to make sure that the new memu including the similar will appear after clicking the 'similar recipe' button	The new memu including the similar appear after clicking the 'similar recipe' button
	Simulate selecting on these similar meals takes you to the page of this meal	Test to click any recipe in the new menu and make sure user can enter the chosen recipe page correctly	The user can click any recipe in the new menu including the similar recipes and then the user can go to the chosen recipe page.
	Ensure all meals returned have at least one ingredient in common	Make sure that the recipes in the menu of similar recipes have at least one ingredient same with the origin recipe	Compare the ingredients of the recipes in the new menu with that of the origin one. At least one ingredient can be found.
Suggested recipes (C25)	Check all recipes are categorised and ordered	Test to ensure that recipes are categorized by searching and receiving data from the back end	Correctly categorized recipes will be returned
	ensure the correct recipes are displayed	Test to ensure that queries are collecting	Correct recipes are displayed given a

	after running a test set of data of multiple categories	the correct recipes	certain query
View recipe history (C24)	Ensure users can go into their profile and see all relevant data they have created	Ensure user can retrieve all recipe data from the backend	User can see all recipes they have saved/viewed
	Ensure all data shows up in an understandable, readable format	Test to ensure no data is corrupt on the database and the received data is formatted correctly	User can see all data correctly formatted

4. Integration Strategy

4.1. Integration Overview

As a company we are using Git Version Control, due to this each user story being worked on is done on a separate branch as to not pollute the master branch. This allows us to quickly and effectively integrate all of the user stories to the final product without causing much slow down in the development process. On the master branch in GitHub there are multiple automated building bots which build the application and run any unit tests to ensure when integrating on GitHub there are no obvious build issues.

The application Scran Plan can be split into multiple key components (epics) and each of these are split into multiple user stories. Each of the epics are listed below and they will be what is integrated into the master branch. When integrating onto the master branch all modules are subject to additional integration testing.

As each of our epics are integrated onto the master branch, they will be tagged as a release on GitHub, and the progress will be tracked using our kanban boards. Before the user stories are integrated to the master branch they will be subject to testing which is set out below in the integration testing in each iteration.

Scran Plan are purchasing two modules from FixPack: Digitised Sounds (Purchased Module), Graphics (Purchased Module). These modules will be integrated into our project during the presentation integration stage (see below), Before these are integrated they are subject to the same entry criteria as all our user stories.

4.2. Entry Criteria

Before entering the integration phases of the modules life cycle. Each of the elements should have all Unit tests successfully passing, Instrumentation test passing and related documentation written including test reports.

4.3. Integration Sequence

Below is the basic iteration sequence these are subject to change as the project progresses:

Iteration 1

1. Launch Sequence and Login - Intergrated
2. Main Home Screen - Integrated
3. User Profiles and Privacy - Integrated
4. Launch Presentation - Due to be integrated 13/3/20

Iteration 2

1. Enabled Meal Planner
2. Social Aspect
3. Recipe sorting, filters and search functionality
4. Healthy Living System
5. User created recipes
6. Gold Membership
7. Recipe and ingredient information

Iteration 3

1. Measuring System
2. Recipe ratings and reviews
3. Favorite recipes
4. Notification systems
5. Chef stars and badges
6. icons for ingredients and recipes
7. Recipe suggestions
8. Personal interaction

4.4. Iteration 1 Integration Strategy

4.4.1. Launch Sequence and Login + User Profiles and Privacy

User Stories to be Integrated: *Register Account (A1), Login to account (A2), User Customisation (C20), Change Password (A3), Retrieve User Info (A4), Delete User (A5), Profile Customisation (A11)*

4.4.1.1. Integration Tests

The following integration tests will be executed before merging into the master branch:

- Test to ensure that users can register account and login after logging out
- Test to add user preferences and login to the account after logging out to ensure that preferences are maintained after log out
- Tests to change the users password and to ensure that login and user customisation remains
- Tests to delete a user and check login will not work and when recreating an account with the same name no preferences remain.
- Tests to ensure that when merging the branches the build still is successful

4.4.2. Main Home Screen

User Stories to be Integrated*: All previous stories + Netflix Style Scrolling (C1), Main Screen Tabs (C2), View Recipe Contents (C3), Dietary Filters (C6), Sort Recipes (C7), Search Recipes by keywords (C18)*

4.4.2.1. Integration Tests

The following integration tests will be executed before merging into the master branch:

- Test to login
- Test to scroll through the app going through each of the tabs to ensure scrolling is smooth and uninterrupted
- Test to add dietary filters to search recipes by dietary requirements
- Tests to search recipes by keywords and open up into the recipe contents
- Tests to ensure that when merging the branches the build still is successful

4.4.3. Launch Presentation

User Stories to be Integrated: *All previous stories + Enter Presentation (B1), Move within presentation (B2), View Presentation steps (B3), Comments to steps (B6) + Digitised Sounds (Purchased Module), Graphics (Purchased Module)*

4.4.3.1. Integration Tests

The following integration tests will be executed before merging into the master branch:

- Test to launch into the presentation from the main home screen
- Test to move through the presentation and view the steps
- Test to view the comments to the steps and adding comments to the steps
- Test to log out and back in to ensure comments remain
- Test to view comments from different accounts
- Test to change privacy preferences and ensure comments can or cannot be viewed depending on privacy settings
- Test to ensure digitised sounds are playable from the presentation and stop when exiting the presentation
- Test to display graphics in the presentation

- Tests to ensure that when merging the branches the build still is successful

4.5. Iteration 2 Integration Strategy

4.5.1. Enabled Meal Planner

User Stories to be Integrated: *All previous stories + Enter Presentation from meal planner (B4), Create Meal Plan (C4), Build Shopping List (C5), Display typical expiry dates (C14), Select Recipes for Breakfast, Lunch, or Dinner (C17), Alerts for expiring food (C21), Recommended Reheat instructions (C22), Number of Portions per Recipe (C34)*

4.5.1.1. Integration Tests

The following integration tests will be executed before merging into the master branch:

- Test to enter the presentation for meal from the meal planner
- Select multiple recipes for breakfast, lunch and dinner and check that their are reheat instructions and number of portions for all recipes
- Tests to ensure that when merging the branches the build still is successful

4.5.2. Social Aspect

User Stories to be Integrated: *All previous stories + Post Comment and Like on user forum (C16), Send friends recipes in the app (C26), Add and Invite Friends (C27), See trending recipes that have high star rating (C28), Rate recipes (C29), Share cooking tips (C31)*

4.5.2.1. Integration Tests

The following integration tests will be executed before merging into the master branch:

- Tests to add friends and open up a message friends recipes
- Test to open recipes and rate multiple recipes and check if friends can see the rating in their app
- Test to share cooking tips and to ensure they are viewable in others app
- Tests to ensure that when merging the branches the build still is successful

4.5.3. Recipe sorting, filters and search functionality

User Stories to be Integrated: *All previous stories + Display similar recipes by ingredient (C13)*

4.5.3.1. Integration Tests

The following integration tests will be executed before merging into the master branch:

- Test to search for recipes with different ingredients and add to meal planner
- Test to launch into recipe presentation from meal planner and check if similar recipes are displayed
- Tests to ensure that when merging the branches the build still is successful

4.5.4. Healthy Living System

User Stories to be Integrated: *All previous stories + Track eating habits (A13), Target weight and calorie intake (C32)*

4.5.4.1. Integration Tests

The following integration tests will be executed before merging into the master branch:

- Test to add target weights and track eating habits to ensure calorie intake is tracked
- Tests to ensure that when merging the branches the build still is successful

4.5.5. User Created Recipes

User Stories to be Integrated: *All previous stories + User created recipes in the app (A12)*

4.5.5.1. Integration Tests

The following integration tests will be executed before merging into the master branch:

- Test to add recipes and view later
- Test to view other user's recipes when they're added to the application
- Test to add comments to recipies once they're added to the server
- Tests to ensure that when merging the branches the build still is successful

4.5.6. Gold Membership

User Stories to be Integrated: *All previous stories + Gold Membership (A9), Removal of Advertisements for Gold Members (C19)*

4.5.6.1. Integration Tests

The following integration tests will be executed before merging into the master branch:

- Test to ensure advertisement disappears when accounts are added as 'Gold Members'
- Test to ensure extra features are not available to non-paid users and are removed from members who were paid and have unsubscribed
- Tests to ensure that when merging the branches the build still is successful

4.5.7. Recipe and ingredient information

User Stories to be Integrated: *All previous stories + Price and nutritional value of recipes (C12)*

4.5.7.1. Integration Tests

The following integration tests will be executed before merging into the master branch:

- Test to add recipes and see if the price and nutritional value is added
- Test to see if recipes are updated with price and nutritional information as the recipes are updated
- Tests to ensure that when merging the branches the build still is successful

4.6. Iteration 3 Integration Strategy

4.6.1. Measuring System

User Stories to be Integrated: *All previous stories + Change the unit of measurements for ingredients (C10)*

4.6.1.1. Integration Tests

The following integration tests will be executed before merging into the master branch:

- Test to add recipes and check measurements have been added and change
- Test to change unit of measurements on existing recipes
- Tests to ensure that when merging the branches the build still is successful

4.6.2. Recipe ratings and reviews

User Stories to be Integrated: *All previous stories + Receive chef stars from ratings (A8), Rate recipes (C9), Add comments and reviews on recipes (C11), Reporting of content (C23), Post questions about recipes (C30)*

4.6.2.1. Integration Tests

The following integration tests will be executed before merging into the master branch:

- Test to review existing recipes
- Test to post comments/reviews on recipes that exist
- Test to ensure posts are deleted on account deletion
- Test to give chef stars to profiles and if they're removed on account deletion
- Tests to ensure that when merging the branches the build still is successful

4.6.3. Favorite recipes

User Stories to be Integrated: *All previous stories + Favorite recipes (C8), Substitute disliked meals in meal planner (C15)*

4.6.3.1. Integration Tests

The following integration tests will be executed before merging into the master branch:

- Test to add recipes to favorite recipes and ensure they are retrieved
- Test to substitute disliked meals in the meal planner and ensure they are added correctly
- Tests to ensure that when merging the branches the build still is successful

4.6.4. Notification systems

User Stories to be Integrated: *All previous stories + Notifications (A6), Notifications Preferences (A7)*

4.6.4.1. Integration Tests

The following integration tests will be executed before merging into the master branch:

- Test to push notifications to devices
- Test to block notifications on devices
- Test to ensure user preferences are updated with notification preferences
- Tests to ensure that when merging the branches the build still is successful

4.6.5. Chef stars and badges

User Stories to be Integrated: *All previous stories + Profile Badges (A10)*

4.6.5.1. Integration Tests

The following integration tests will be executed before merging into the master branch:

- Test to ensure stars and badges are added to profiles
- Tests to ensure badges are hidden/displayed when privacy setting allow
- Tests to ensure that when merging the branches the build still is successful

4.6.6. Icons for ingredients and recipes

User Stories to be Integrated: *All previous stories + Custom icons next to ingredients when viewing recipe (A14)*

4.6.6.1. Integration Tests

The following integration tests will be executed before merging into the master branch:

- Test to ensure custom icons are on new recipes
- Test to ensure icons are on existing recipes
- Tests to ensure icons appear throughout the app
- Tests to ensure that when merging the branches the build still is successful

4.6.7. Recipe suggestions

User Stories to be Integrated: *All previous stories + Promoted similar recipes (B5), Suggested recipes (C25)*

4.6.7.1. Integration Tests

The following integration tests will be executed before merging into the master branch:

- Tests to ensure suggested recipes are shown to the correct users
- Test to ensure similar recipes are displayed
- Tests to ensure that when merging the branches the build still is successful

4.6.8. Personal interaction

User Stories to be Integrated: *All previous stories + View recipe history (C24)*

4.6.8.1. Integration Tests

The following integration tests will be executed before merging into the master branch:

- Test to ensure recipe history is show to the correct users
- Tests to ensure that when merging the branches the build still is successful

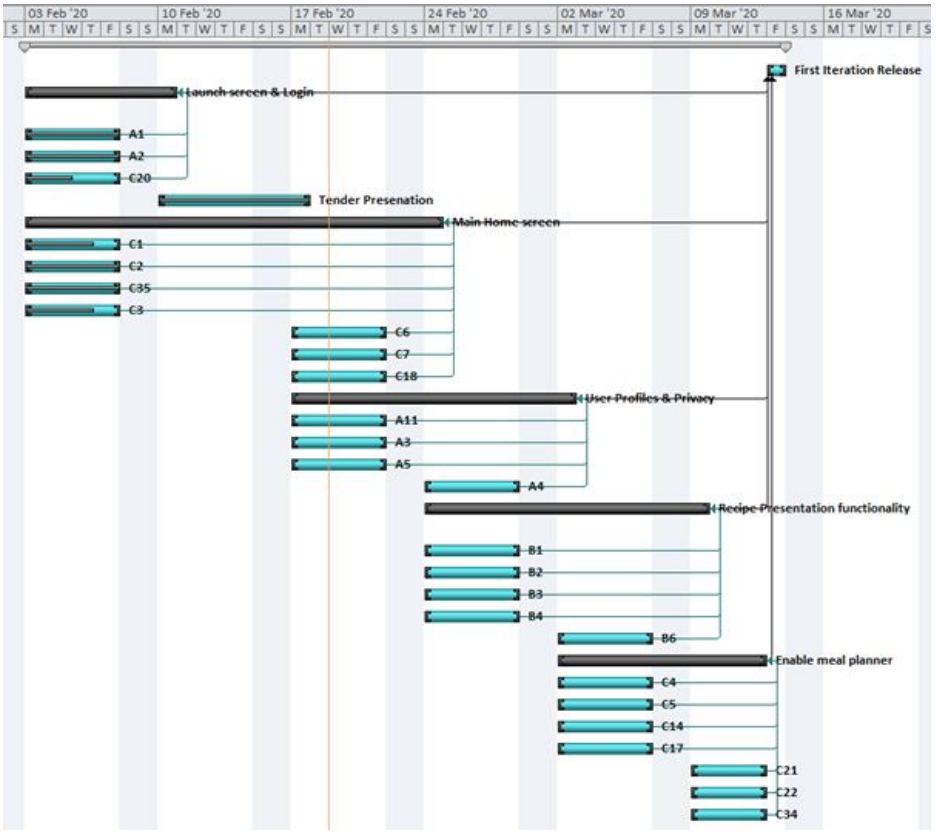
5. Responsibilities and Schedule

5.1. Roles and Responsibilities

Role	Responsibilities
Project Leader	• Oversee the project • Liaise with members of the team over any issues encountered
Testing and Integration Manager	• Oversee general testing of modules • Oversee integration of modules onto the master branch • Dealing with errors in conjunction with lead software developers • Verifying the quality of testing from testing reports and passing onto project leader if any issues occur.
Lead Software Developer	• Oversee development of user stories • Dealing with errors in the code with the developers and T&I Manager
Tester	• Write and execute relevant testing • Feedback problems to testing and integration manager • Produce test reports

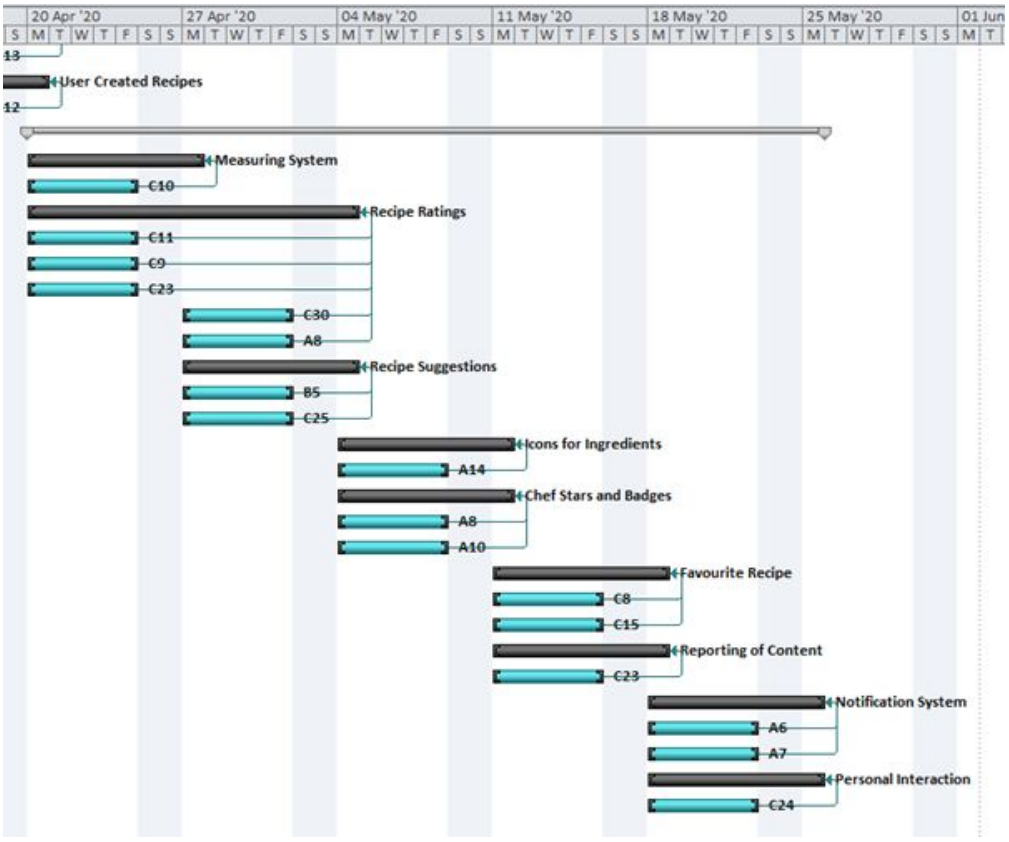
5.2. Schedule

5.2.1. Iteration 1 Schedule



5.2.2. Iteration 2 Schedule

5.2.3. Iteration 3 Schedule



6. Review and Retest Procedures

At Scran Plan every user story is peer reviewed by a developer who didn't write the story. When a user story is complete; a pull request is opened on GitHub and the user stories are peer reviewed by other members of the team who check the code for basic logic and check on our CI workflow TravisCI that it builds successfully and all tests run on their local machines too. Peer reviewers may find there are problems with the code which is when they need to record the problems and feed it back to the user story developers. This is done by filling in the notes section of the peer review checklist and passing it to the testing and integration manager to give a second opinion on. If the problem persists then it is commented on the Pull request with relevant lines of code and passed onto the user story developer who will refactor the code and fix the problems.

When it is necessary to re-open the branch after it has been merged into the master. CoDev will implement their review and retest procedures (Regression Testing). This is where new tests are written for additional code and all existing tests are re-run to ensure they're passing.

6.1. Problem Recording and Resolution

For recurring problems in code, or problems that a developer cannot resolve CoDev has recording and resolution. For each user story developers complete a testing document with summarises all testing done for the user story including any manual testing. When completing these documents there should be no errors with the user story and they should compile on the developers side.

6.1.1. Error Reporting

When an error is found in coding and it disrupts the development process the following steps will be taken:

1) Error should be reported to testing and integration manager, lead software developer who will discuss the most appropriate action plan
2) Explorationary testing must be done as soon as possible to find the root cause of the issues, if a root cause is found then the user stories development can resume
3) In the case of the error not being found a meeting with the group leader, t&I manager and lead software developer should be held to discuss an appropriate action plan be it pause the user story or discard it all together.

6.2. Regression Testing

When changes are made it is necessary to test again to ensure that all areas are covered. All tests must be rerun and necessary test reports written to cover the new code that has been written. Just like during the initial testing process these changes are subject to peer review to ensure that the code is to an acceptable standard and complies with no issues.

When code has been regression tested the reports are passed onto the testing and integration manager and the code is reviewed to be readded into the master branch on GitHub.

7. Suspension, Restart and Exit Criteria

Occasionally, it is necessary to stop integration of user stories to the master branch. If a developer feels it is necessary not to integrate a user story they will flag it to the testing and integration manager who will ensure that the team is aware that they shouldn't intergate said story to the main branch. The suspension of user stories would usually be done due to errors being found and no resolution found in good time (3-5 working days). If an error is not resolved within this time a meeting will be held with the developer, testing and integration manager, lead software developer and team leader who will create a plan of action for the user story.

If it is decided to restart the user story then the testing and integration plans will need to be updated to fit in the user story which has been moved to a later date. Integration reports will need to be done for that user story when it is individually added to the master branch.

7.1. Exit Criteria

The Exit criteria of the user stories from the testing and integration plan is when:

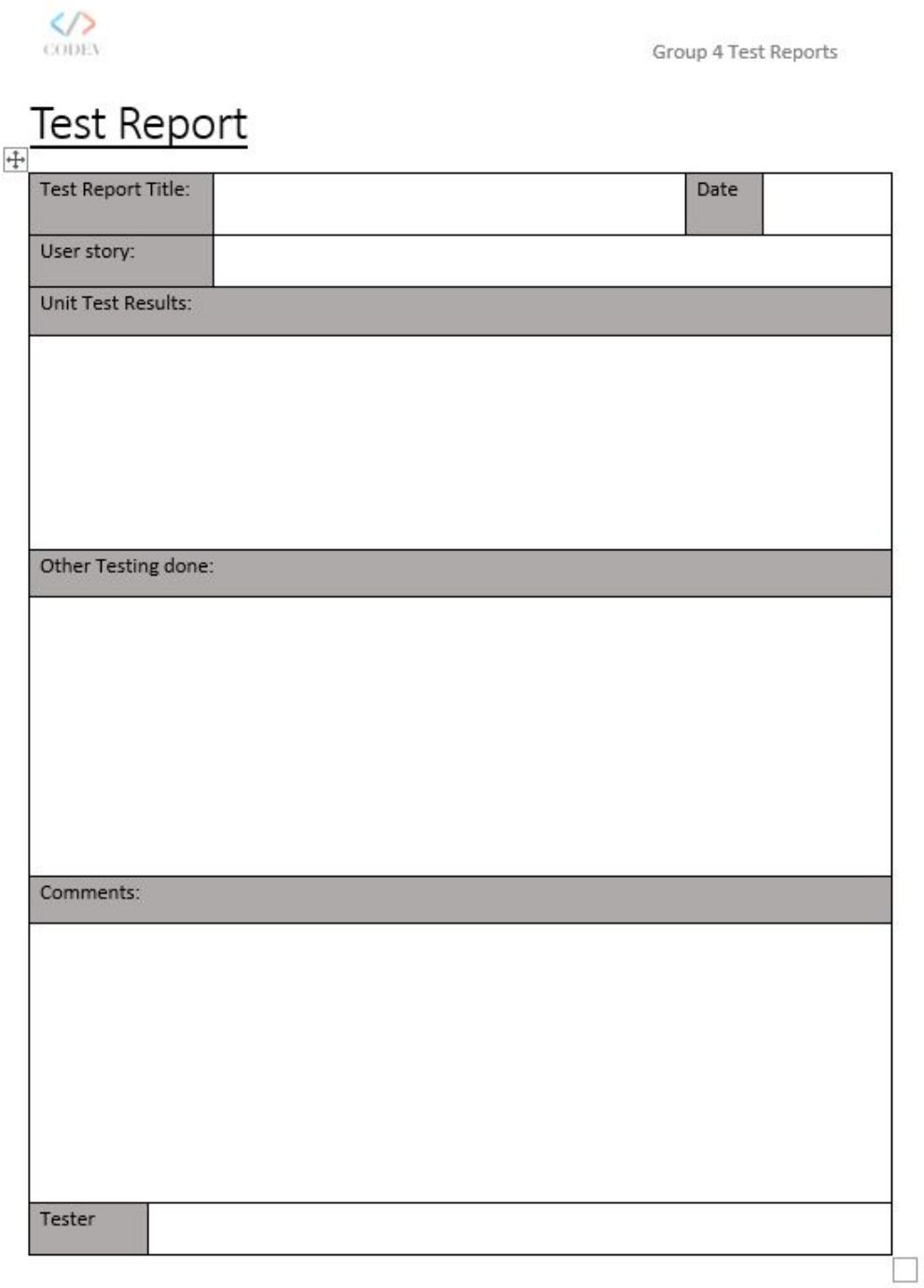
- All testing reports, Peer Review reports and integration reports are complete
- The user story is on the master branch and the stale branch deleted from github
- No Build errors are displayed in Travis

8. Bibliography

[1] 'Travis CI'. [Online]. Available: https://travis-ci.org/dashboard.

9. Appendix

9.1. Test Report



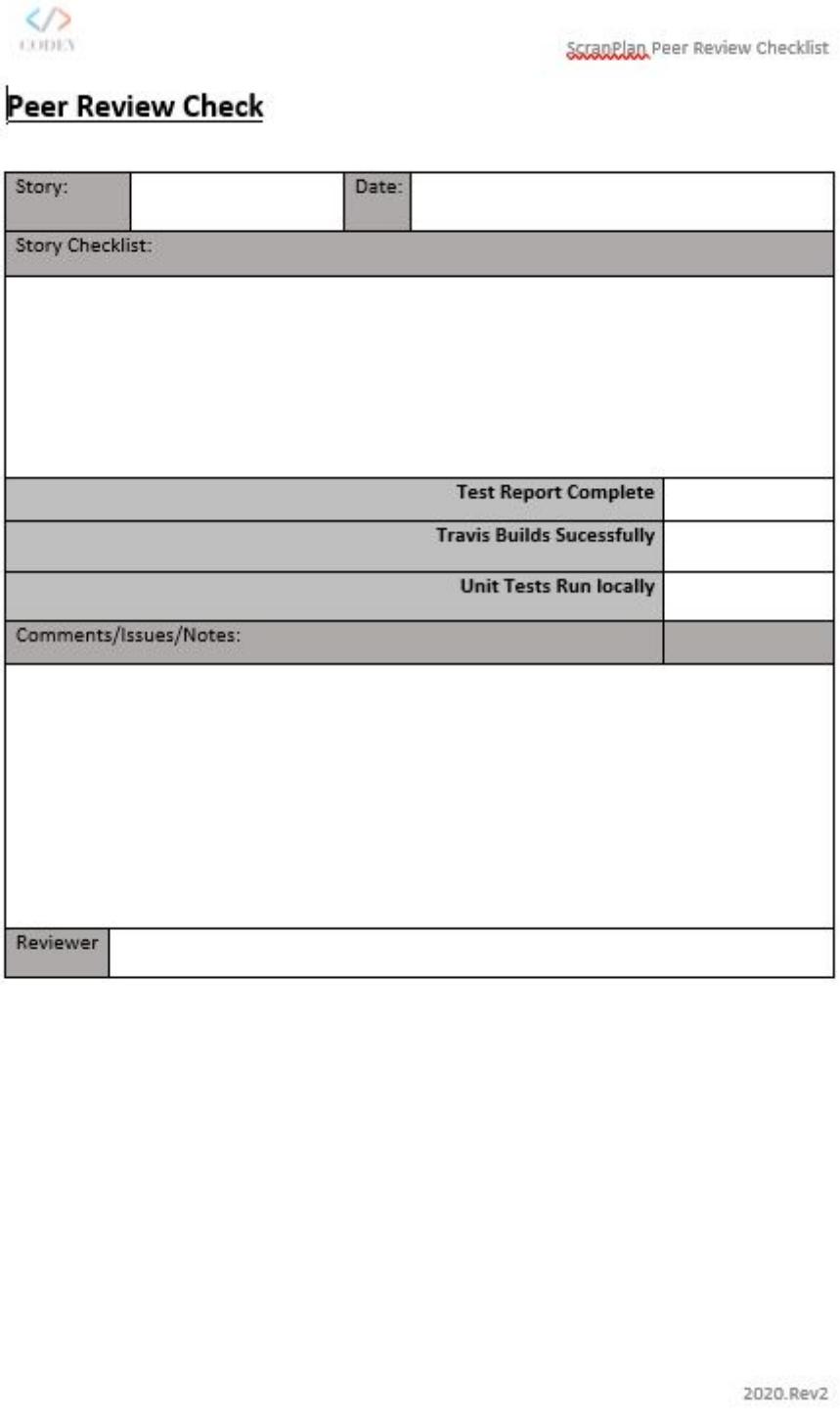
CODEV

Group 4 Test Reports

Test Report

Test Report Title:		Date	
User story:			
Unit Test Results:			
Other Testing done:			
Comments:			
Tester			

9.2. Peer Review Report



CODEV

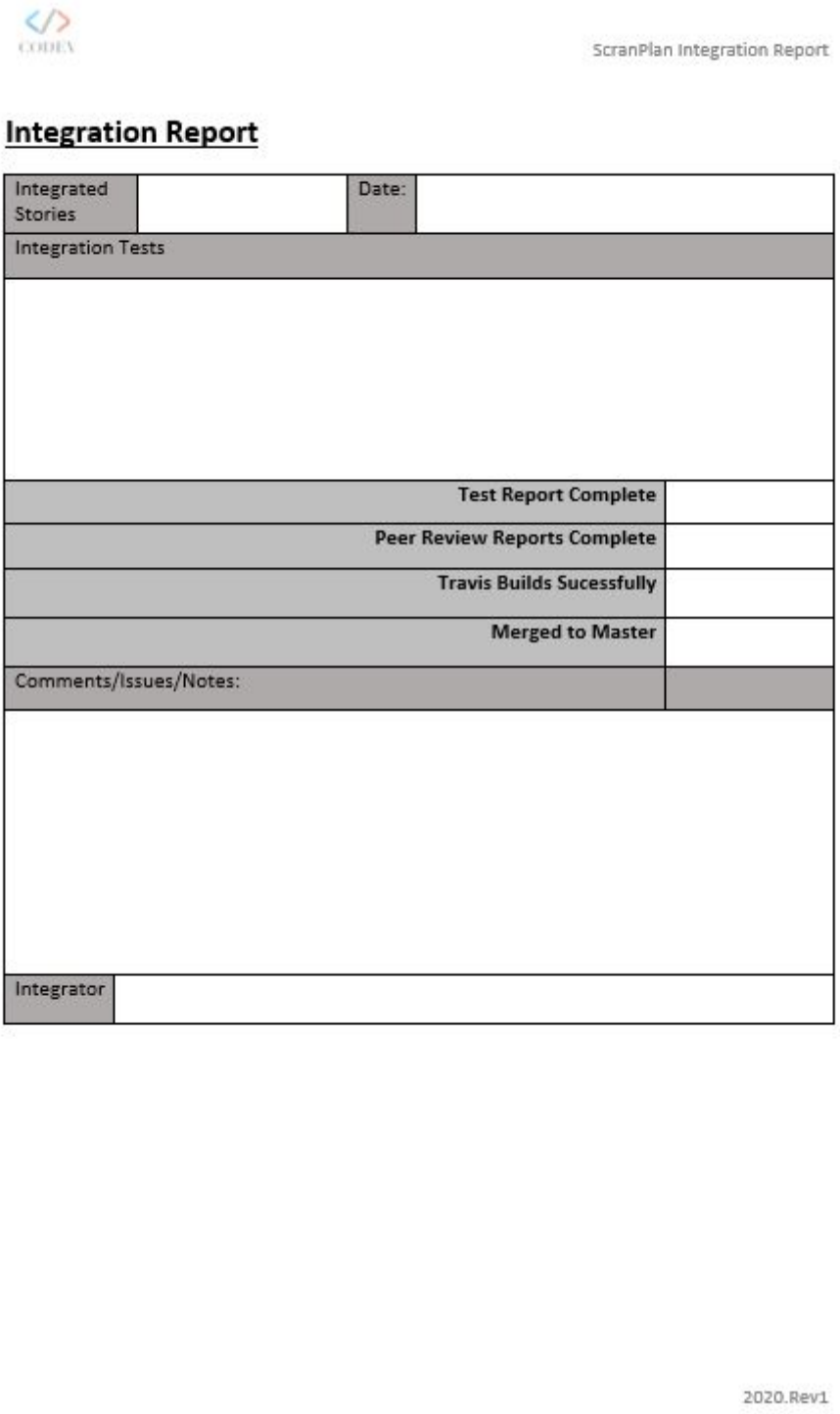
ScranPlan Peer Review Checklist

Peer Review Check

Story:		Date:	
Story Checklist:			
		Test Report Complete	
		Travis Builds Sucessfully	
		Unit Tests Run locally	
Comments/Issues/Notes:			
Reviewer			

2020.Rev2

9.3. Integration Report



CODEV

ScranPlan Integration Report

Integration Report

Integrated Stories		Date:	
Integration Tests			
		Test Report Complete	
		Peer Review Reports Complete	
		Travis Builds Sucessfully	
		Merged to Master	
Comments/Issues/Notes:			
Integrator			

2020.Rev1