



Send A: Project Deliverable 1

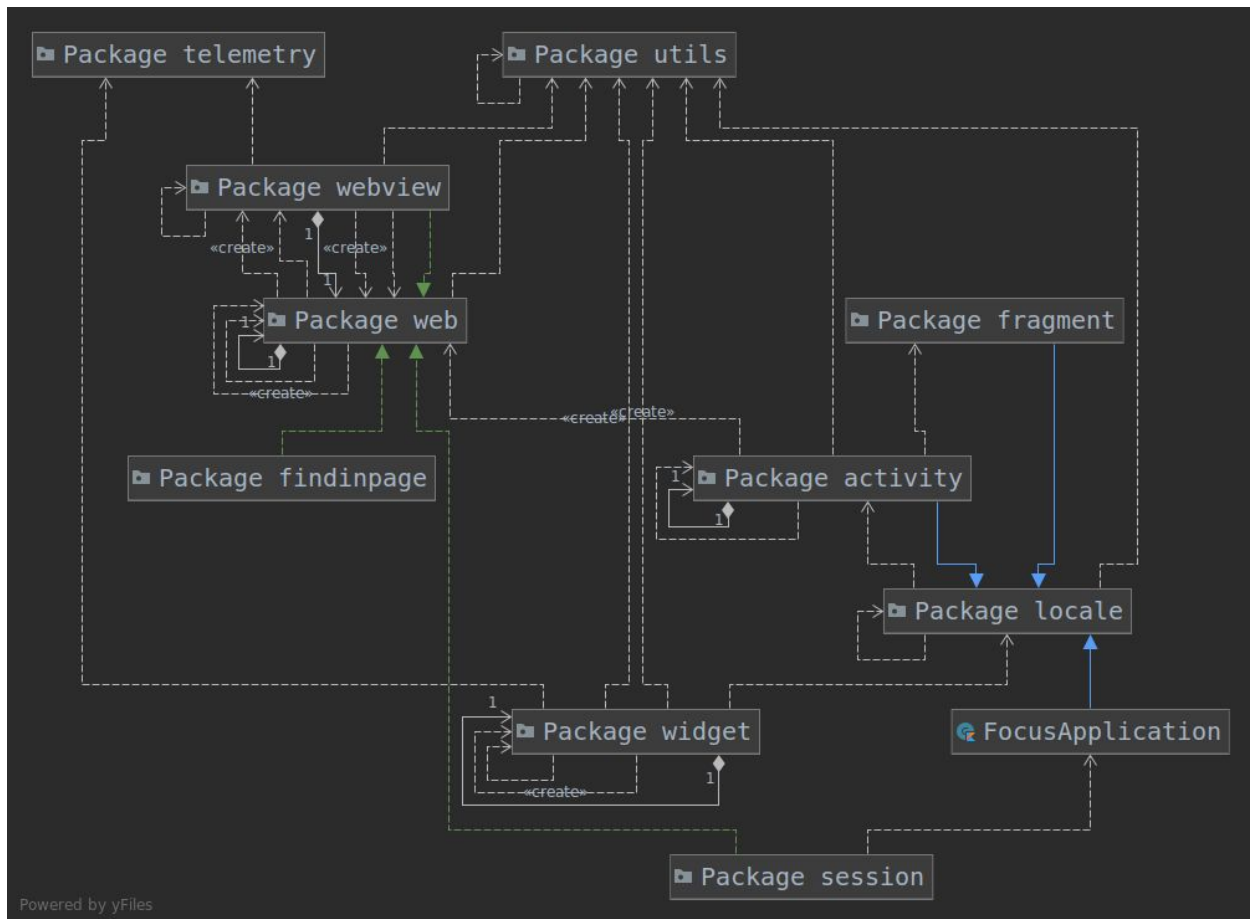
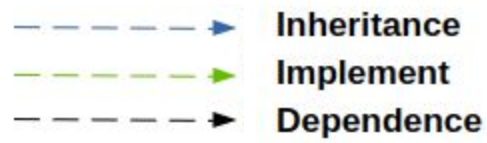
Team Members:
Anthony Le,
Birathan Somasundaram,
Pratana Atikhomkamalasai,
Suxin Hong,
Yuewen Ma

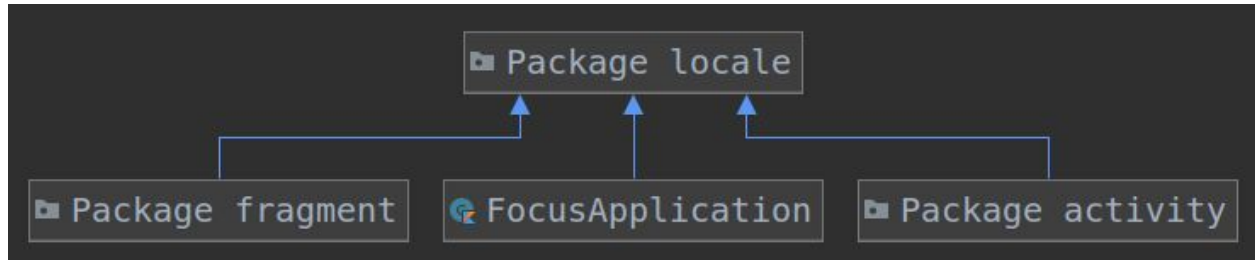
Table of Contents

Table of Contents	2
Architecture	3
Core Package relationship UML	3
Interesting aspects	6
1. Observer design pattern analysis	6
2. Using onNewIntent()	7
3. Implements and inheritance	8
Possible improve for architecture	9
Software Development Process	10
1. Process Decision	10
2. Each Phase	11
Requirement Definition	11
Component Analysis	11
System design with reuse	11
Implementation and unit testing	11
System Validation	12
3. Other Processed Considered	12
Test-driven development process	12
Incremental development process	13
Kanban	14

Architecture

Core Package relationship UML



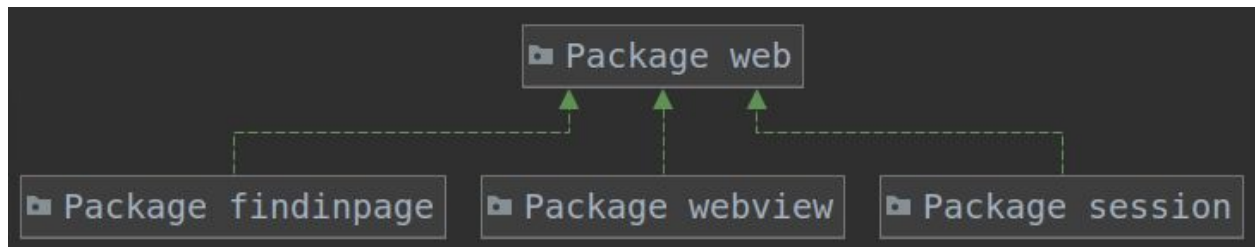


Package locale: Include some subclass of the android classes such as Application, Activity, Fragment... Also can read personalized settings when the application initializes such as language preference.

Package fragment and settings: Include all fragments in the project. The most important functions here. Display above Activities.

Class FocusApplication: Base class for maintaining global application state.

Package activity: Include all activities for each UI interface in the project.



Package web: Include some Interfaces and classes for the basic building block for user interface components. Based on android.view.View. A View occupies a rectangular area on the screen and is responsible for drawing and event handling.

Package webview and findinpage: Include some classes for implementing WebView objects that allow display web content as part of activity layout. WebView is an android built in component and it is based on Chrome.

Package session: For controls activity lifecycle we won't be able to touch it. But will use it for contact with different activities or fragments.

Interesting aspects

1. Observer design pattern analysis

In the project, usually use `registerSessionObserver()` to maintain a list of dependents, and notify them automatically of any state changes, usually by calling one of their methods. However, we found that they **do not implement Observer design patterns by themselves**. They are importing packages from mozilla lib.

// See the Observable interface they used in project

(<https://mozac.org/api/mozilla.components.browser.session/-session-manager/-observer/>)

register(observer: T, owner: LifecycleOwner, autoPause: Boolean = false)

Example in project code (line 85, in MainActivity.kt):

// Registers an observer to get notified about changes.

`components.sessionManager.register(observer, thisActivity)`

// Create Observer innerclass as argument for register

// The selection has changed and the given session is now the selected session.

override onSessionSelected

// show the result page

`showBrowserScreenForCurrentSession()`

// The given session has been removed.

override onAllSessionsRemoved

// show the default page waiting for the input

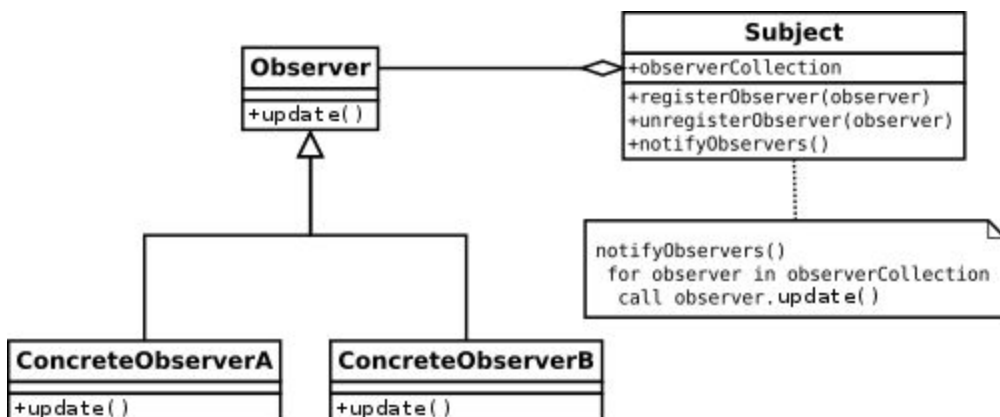
`showUrlInputScreen()`

// All sessions have been removed. Note that this callback will be invoked whenever `removeAll()` or `removeSessions` have been called on the `SessionManager`. This callback will NOT be invoked when just the last session has been removed by calling `remove()` on the `SessionManager`.

override onSessionRemoved

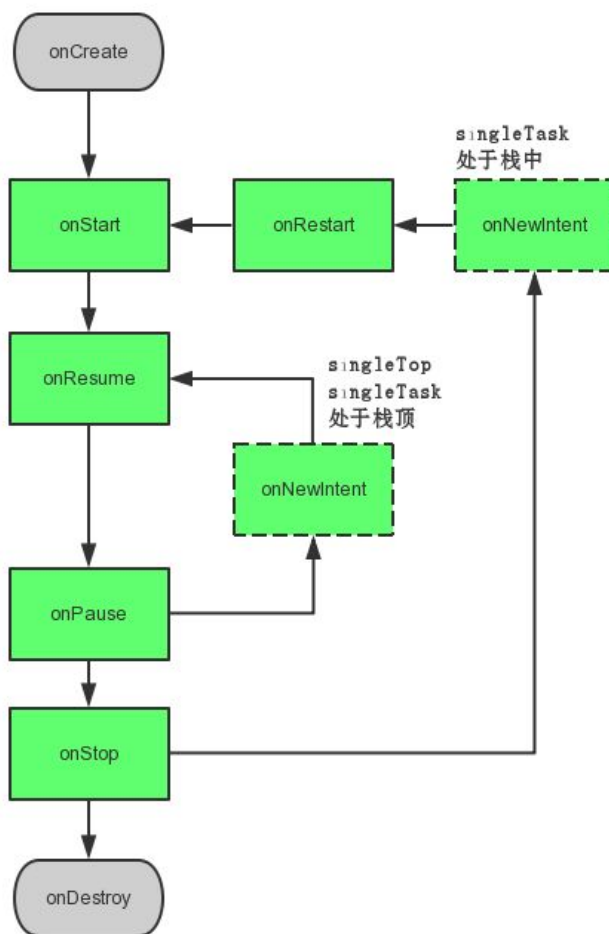
// If needed show the first run tour on top of the browser or url input fragment.

`showFirstRun`



2. Using onNewIntent()

Usually when first time to execute app Activity. App will execute as **onCreate()**→**onStart()**→**onResume()**. However, if we press the home button and execute from the background. The system will execute as **onNewIntent()**→**onRestart()**→**onStart()**→**onResume()**. The onCreate won't be called. Note that if the android system doesn't have enough memory or cleaned the background then when you execute Activity will go back to **onCreate()**→**onStart()**→**onResume()**.



MainActivity	
Companion	Companion
ACTION_ERASE	String
ACTION_OPEN	String
EXTRA_NOTIFICATION	String
EXTRA_SHORTCUT	String
EXPERIMENTS_JOB_ID	int
intentProcessor\$delegate	Lazy
previousSessionCount	int
isCustomTabMode()	boolean
getCurrentSessionForActivity()	Session
getIntentProcessor()	IntentProcessor
onCreate(Bundle)	void
registerSessionObserver()	void
applyLocale()	void
onResume()	void
onPause()	void
onStop()	void
onNewIntent(Intent)	void
processEraseAction(SafeIntent)	void
showUrlInputScreen()	void
showFirstRun(Session)	void
showBrowserScreenForCurrentSession()	void
onCreateView(String, Context, AttributeSet)	View
onBackPressed()	void
checkBiometricStillValid()	void

3. Implements and inheritance

Most components in the software do not implement/extend directly from Android's mainactivity app. Rather they go through a number of layers of inheritance to improve code reusability, maintainability, readability. Each parent class serves a specific purpose/functionality.

For instance, we can look at the MainActivity Class. The purpose of the main activity class for Firefox Focus is to manage what behavior the app is when it's created, start, stop, pause, resume, etc. The constructor can be found in Android's MainActivity class. However, instead of implementing/extending directly from this class, MainActivity inherit from LocaleAwareAppCompatActivity

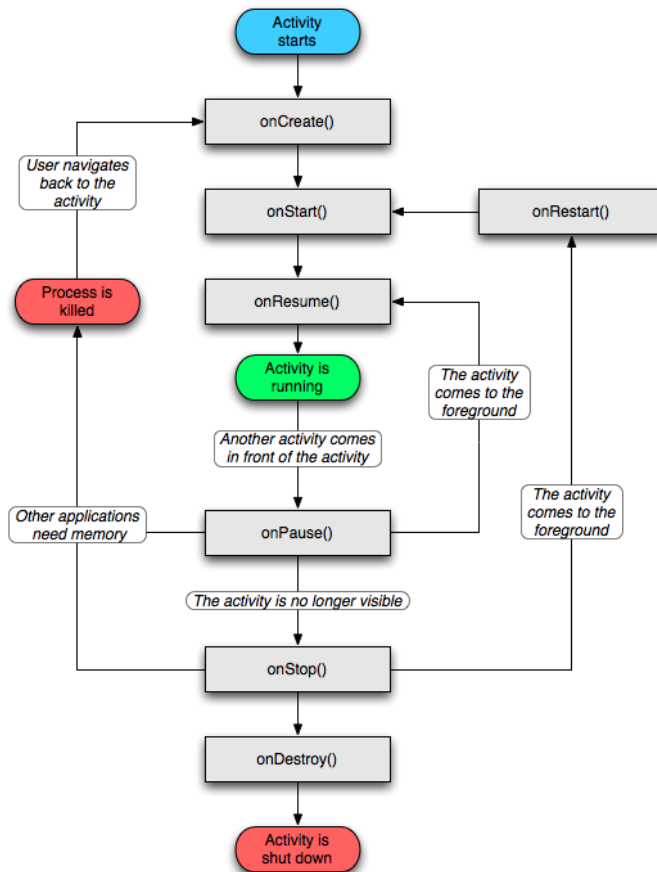
```
open class MainActivity : LocaleAwareAppCompatActivity() {
```









LocaleAwareAppCompatActivity defines some behavior that an App might take to comply with some **localization** on the device, such as **language, accessibility, date and time**, etc. These behaviour are applied whenever the app is created or localization changes

LocaleAwareAppCompatActivity extends Android MainActivity class, and calls its parent constructor after setting up localization.

Possible improve for architecture

In the project when we open a new activity (first time start): the program will execute onCreate, then onResume and stop at this onResume, the program will do some operations such as go to other activities. Press the return key: execute onPause, then onStop, then onDestroy. After executing onDestroy the Activity is destroyed. However, in the most common normal case the activity Lifecycle should be like this:



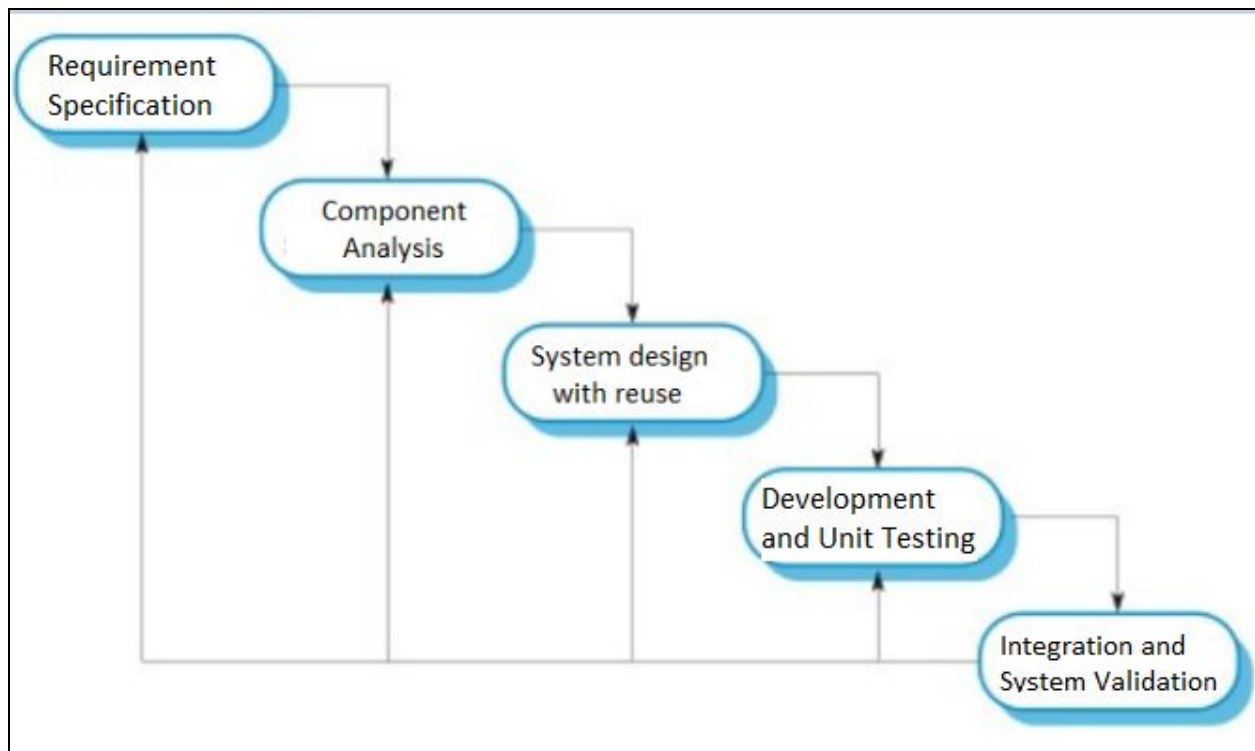
MainActivity		
	Companion	Companion
	ACTION_ERASE	String
	ACTION_OPEN	String
	EXTRA_NOTIFICATION	String
	EXTRA_SHORTCUT	String
	EXPERIMENTS_JOB_ID	int
	intentProcessor\$delegate	Lazy
	previousSessionCount	int
<hr/>		
	isCustomTabMode()	boolean
	getCurrentSessionForActivity()	Session
	getIntentProcessor()	IntentProcessor
	onCreate(Bundle)	void
	registerSessionObserver()	void
	applyLocale()	void
	onResume()	void
	onPause()	void
	onStop()	void
	onNewIntent(Intent)	void
	processEraseAction(SafeIntent)	void
	showUrlInputScreen()	void
	showFirstRun(Session)	void
	showBrowserScreenForCurrentSession()	void
	onCreateView(String, Context, AttributeSet)	View
	onBackPressed()	void
	checkBiometricStillValid()	void

Compared project code with the graph we know they put the code in `onStart()` and `onRestart()` into both `onCreate()` and `onResume()` so that they don't have `onStart()` and `onRestart()` in the project. However, We can't live without `onStart()` because that is the state when the activity becomes "visible" to the user, but the user can't "interact" with it yet may be cause it's overlapped with some other small dialog. This ability to interact with the user is the one that differentiates `onStart()` and `onResume()`.

Software Development Process

1. Process Decision

For the next deliverable we plan on using the **Waterfall development process** with a few applications of the **reuse-oriented development process (RODP)**.



We think plan-driven models like Waterfall and RODP are good choices mainly because they are both simple and clear-cut processes that are fairly easy to implement/understand given our busy school schedules. A well-defined set of requirements, which is required by the waterfall model, can be easily produced from detailed issues from the open source community for Firefox Focus. This is because the issues are already given to us by other developers and we have a head start for producing the requirements than if we were to derive them from scratch using clients. They are also less likely to change. This well-defined set will also make testing in later phases much easier, since the requirements would already detail certain testing scenarios. However, the Waterfall model can also be quite slow and does not allow for much flexibility when it comes to developing onto an already well-established code base.

In order to counteract this lack of flexibility, ideas from RODP were also incorporated since this project would most likely involve the reuse of multiple interconnected components such as **Android Framework to create activities and fragments**. Since we only need to make

changes to code that directly concerns an issue/bug and not any components or large scale design surrounding it, we will save a lot of time, reduce risks and improve our chances of success. Also, since the main language used in this project is a language called **Kotlin**, which none of us have previously worked with, it would be better to just follow their example and build on top of the existing code base.

2. Each Phase

1. Requirement Specification

The first stage is **Requirement Specification** like in waterfall where we will produce a well-defined set of requirements for our chosen issues from the open source Firefox Focus repository. This will be done by analyzing community discussions related to the open issues and getting a strong understanding of them using Mozilla's [Codetribute](#) website and documentation. Researching similar privacy-focused browsers such as **TOR** and **DuckDuckGo** may also provide some insight on the issues. Once this is done, we can break down the problem into detailed requirements and we will have a general idea of what we are working towards and how we are going to be splitting up the work.

2. Component Analysis

The second stage is **Component Analysis**. Here, we will find the components that are directly related to our issues and get an idea of how they are connected with others and what kind of dependencies they have. Reading the documentation of these other components will help us avoid writing redundant code and also help us figure out which components are affected by the components we have chosen to modify. By doing this, we will get a larger understanding of the issues and code we will be working with along with all its related components.

3. System design with reuse

The third stage is **System design with reuse**. In this stage we will design a solution that includes the use of reusable components, for the issues we've chosen to solve and that fills all the necessary requirements. This solution will then be broken down into subtasks and split evenly among the team. When splitting these subtasks we need to make sure that the subtasks match individual skill sets and each issue has a subteam that is comfortable with working with it.

4. Development and unit testing

The fourth stage is **Development and unit testing**. After we are assigned our subtasks, each team will be responsible for implementing and unit testing their own code. This will be an iterative process, unlike waterfall, where we will be testing in parallel as we are implementing our solutions. Each team must also make sure to report progress to Suxin, our project leader. There will be a separate branch for each team and another separate branch for each team member and merging to master will not occur without a team review.

5. Integration and System Validation

The fifth and last stage is **Integration and System Validation**. After all work is finished and merged into master, we will run all integration tests to ensure our modified components are working correctly, and related components are unharmed by those modifications. Some UI tests may be difficult to implement, so manual testing of the application may be required (this also applies for unit testing in the previous stage).

3. Other Processes Considered

a. Test-driven development process

We have considered Test-driven development process because

- Safety and Security
 - Due to the nature of web applications, safety and security is one of our main concerns for this project. By developing test suites and having them available at the beginning of the development process, we would be able to notice any design issue or bug early.
- Easy to refactor, integrate and maintain codebase
 - Test driven development process will also force us to review the code base and make small changes regularly. This will make the code become more modular which in turns makes it easier to be refactored or integrated into the source. The maintainability of the codebase will also be improved as any modification made will be tested and corrected as we commit.

But ultimately decided against it because

- Time Consuming
 - It would take a lot of time to get started and work on specific UI issues, where we might have to use 3rd party testing tools like Selenium, which could take a lot of time to learn and implement. We also need to allocate time and resources in developing and maintaining these cases.
- Knowledge Required

- In order to develop these test cases, we need a deeper understanding of the overall project architectures and designs. Since we did not develop this project from the ground up or have any previous experience with **Focus**, we believe that leaving the tests for the end would be a better option as we could have missed important requirements

- **Unfinished Codebase**

- To test certain parts in isolation we may need to mock other unfinished parts, which there are many of based on the Focus issues listed. Some parts might also be difficult to test. This would slow down the development as additional resources will have to be allocated to fix this issue.

b. Incremental development process

We have considered **Incremental development process** because

- **Easy to test and polish**

- For every iteration we only work toward a small set of requirements, it would not take us long to develop a minimum working version. We would be able to start testing and polishing early in the development process

- **Flexible**

- Changes may arise during the process, This model allows us to accommodate them directly in the next iteration. Implementation of these changes would be easier since we will be gradually adding features compared to waterfall where everything is likely to be designed and finalized.

But ultimately decided against because

- **No Direct Customer Feedback**

- Due to the nature of the course, we have no direct way of contacting and receiving customer feedback for each deliverable. There is no need to deliver a minimum working version to be tested and evaluated as we only work on a small number of bugs. We also do not have a lot of time for many iterations. If we could decrease the delivery time for each iteration, it would be too short and not resource effective to deliver anything meaningful.

- **Maintenance and System Degradation**

- Additional resources will be needed when implementing these changes. We will have to spend time developing test cases when we make any modification to the previous iteration. The code will also be hard to modify after all since we will not be able to spend much time developing and refactoring. As we introduce new changes, the system architecture and design will degrade and lead to a harder to maintain codebase.

c. Kanban

We have considered **Kanban** because

- **Visibility and Limited work in progress**

- Every bug will be broken into tasks and subtasks. They are displayed clearly under categories such as To Do , Doing and Done. Over assigning is unlikely to happen as we can tell directly from the board when the column is overflow. It will also be easier for us to keep track and improve team performance.

- **Improved Communication and Collaboration**

- It can also be used to improve communication and collaboration within the team . Kanban requires us to hold us a regular daily short meeting at the board. Each team member will get a chance to inform the group about their work , give feedback and express their concerns.

But ultimately decided against because

- **Scope Change**

- Regular feedback we will receive will likely lead us to requirement change. It will also distract us from the original goal and increase our workload.

- **Time and Teammember commitment**

- Due to the regular meeting , we also will have to dedicate our time and attend meetings. We are students and have other commitments. The bugs we will be working on also do not need a regular update as requirements are clearly set and outlined by Mozilla.

Conclusion

In conclusion, we chose to use a process that incorporates ideas from both the Waterfall and reuse-oriented development processes. This merged development process is easy to understand with clearly defined stages and is also suitable for our team given our school schedules and other commitments.