Progress report marking sheet

Student's name: **Christopher Alastair Irvine**	Student's Reg: **100036248**	
Supervisor: **Dr Katharina Huber**	Date:	Signed:
2nd marker:	Date:	Signed:
Title: **CODEX: A Website and Progressive Web App in Node.js & React for table-top role-playing games**		

	Agreed Marks
Description of project: aims, motivation	/10
Description and understanding of issues and problems addressed in the project	/20
Design and planning	/50
Evaluation of progress	/10
Clarity, structure and correctness of writing	/10
Overall mark	/100

Comments

Report and completed mark sheet should be returned to the LSO.

CODEX: A Website and Progressive Web App in Node.js & React for table-top role-playing games

Christopher Alastair Irvine

registration: 100036248

1. Introduction

D&D is a popular table-top role-playing game which (for the purposes of CODEX) is based on the following principles. D&D is random, operates using simple mathematical formulas and the game cannot survive without a *Dungeon Master* (DM). For a worked example of D&D please see Appendix A. In the example we will see how a group of players (known as the *Party*) explore the world through a series of challenges (such as hitting an enemy or picking a lock, that is collectively known as the *Campaign*) of that the DM has created. A typical D&D set up can been seen in Figure 1

Figure 1: (Left to Right) A: In this image we can see a typical D&D table set up. The items on the table from left to right is the *Monster Manual* (Perkins, 2014), scratch pad of paper, pens, series of die, DM's Screen, a battle mat, set of tokens, a character sheet, more die and the *Player's Handbook* (Mearls and Crawford, 2014b). A group playing D&D does not need all of the listed equipment each. The minimum equipment needed in a group, are the three core books (Player's Handbook, Monster Manual and *Dungeon Master's Guide* (Mearls and Crawford, 2014a)), one set of die and some paper. Everything else is optional.
B: In this image we see the die necessary to play D&D. They are (from left to right); 4, 6, 8, 10, 12, 20 and 100-sided. The die are denoted by the letter d followed by the number of sides (for example the 8-sided die is a d8). The d100 only has 10 sides but the sides increase in multiples of 10 and when rolled together with the d10 is known as the *percentile die*.

1.1. The Dungeon Master

As explained above, the DM creates the world in front of the party and controls the narrative of the Campaign. This narrative could come from a purchased supplementary

book written by Wizards of the Coast(the company that make D&D) or the DM could create the story themselves. Either way the DM needs to know both the world and the story they want to tell inside out. Even if then a DM needs to have excellent improvisational skills for when the Party does something unexpected. The challenge of playing the role of DM is to know what you want to happen next and guide the Party to that event without the Party feeling forced into it. It is that challenge that makes the role of DM hard work and fun at the same time. It is not uncommon for a DM to have a couple of folders full to the brim with notes (Mercer, 2016). Fulfilling the role of the DM is like being a mastermind, pulling strings from the shadows, whilst trying to look after a large litter of over-active puppies.

1.2. D&D is random

Due to the importance placed on the result of die rolls, anything and everything can go wrong. Whilst the overarching story tends to stay the same the small details can change multiple times over the same session. This can make preparing content for the Party a living nightmare for a DM. Many a DM have found themselves having hours of preparation work wasted because of the result of a die roll or a decision made by the Party. But this is no one's fault. The Party do not know what is happening next and the DM can only react to the actions of the Party. It is the chaotic nature of D&D.

1.3. Mathematics of D&D

As previously stated, almost everything that happens during a Campaign is decided by die rolls. However the Party (and the *characters* they encounter) do have a small degree of control over their actions. Each character in D&D has a set of *ability scores* that make them unique, each score having a value of 0 to 20 (unless an item changes that) (Mearls and Crawford, 2014b). These ability scores are translated into *ability modifiers*, which, for better or worse, affect the outcome of the die roll. If a character is particularly skilled in a task they are said to be *Proficient* in it, further adjusting the outcome of a die roll. With the exception of damage rolls, the d20 die is used to make decisions in D&D. Albeit with some minor variations, die rolls in D&D 5th Edition follow this formula:

$$d20 + ability\ score + proficiency = result$$
$$For\ example:$$
$$15 + 4 + 2 = 21$$

However, as far as CODEXis concerned, the Party and DM will perform these calculations for themselves as its part of the fun of the game. Attack rolls, damage rolls and ability checks will be held out width CODEX. The mathematics that CODEXwill handle

is the calculation of Ability Modifiers:

$$\frac{(ability\ score - 10)}{2}\ (rounded\ down)$$

In addition to the tracking of each character's *hit points* during combat. Which is a simple addition or subtraction depending on whether a character took damage or gained healing.

1.4. What is CODEX?

Now we have an understanding of D&D, we can now understand the purpose of CODEX and what problems CODEXseeks to address. CODEX is a Software Engineering Project that aims to engineer a Web-App built in ReactJS and Node.JS technologies. The functionality of which is to:

- Assist the DM with the preparation of D&D content.
- Track the Combat Statistics of each character during Encounter.
- Give the Party access to the rich lore of the world they play in.

CODEX is developed using the Agile Solo methodology (Nyström, 2011). Agile Solo provides CODEX with a structures environment of regular deadlines to ensure that the project stays on track.

ReactJS was chosen because of the ability to rapidly built and prototype User Interfaces (UI) across multiple platforms (Inc., 2017). Node.JS will handle the communication between CODEX and the corresponding database (Node.js, 2017). Both of these technologies follow the basic Web-App Architecture laid out in Figure 2.

2. CODEX Design

In this section, the Design of CODEX that has been developed over the last three months will be critiqued and an overview of the critical elements of the Design will be presented. The development of the CODEX Design followed the Agile Solo weekly sprint formula. This was achieved by the single developer working on CODEX asking themselves a series of questions and recording their answers at the end of each week. Every two weeks the developer would aim to meet with their supervisor to ensure that CODEX was on track and any issues were being addressed. Trello was used to organise and manage the remaining tasks of the Design (Software, 2011).

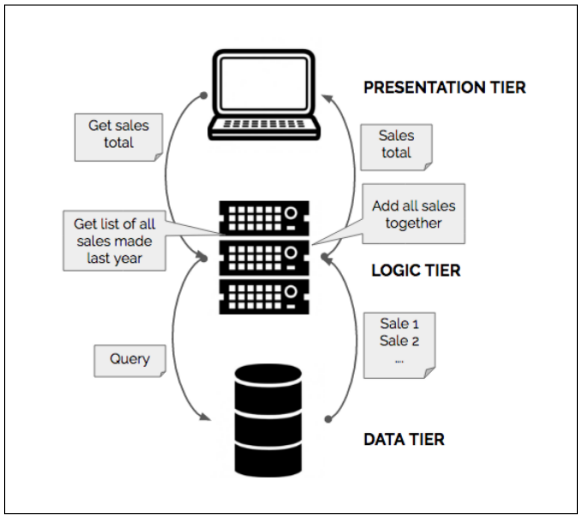

Figure 2: Overview of a three-tier application and its dependencies (De Groef, 2016)

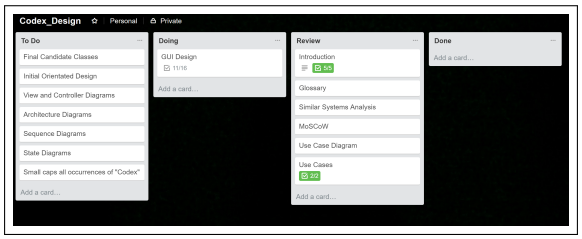

Figure 3: The Trello Board used to track the progress of the CODEX Design Iterations.

2.1. Iterative Design

CODEX has iterated through two design phases over the three months of development. The first phase focused on low fidelity (low-fi) prototyping of the User Interfaces and was designed with the use of PHP and ReactJS as its principle languages. However, to keep CODEX as simple as possible, PHP was replaced by Node.JS in the second iteration. This change was made in order to eliminate a set of testing to accommodate the use of a second language. Now CODEX development can be tested at every stage with a single set of testing classes. In addition to the change of language, the second iteration of CODEX Design focused on medium fidelity (mid-fi) prototyping of the GUI through the use of the Marvel App Website (App, 2013).

2.2. The Use of ReactJS and Node.JS

As discussed in section 1.4, CODEX aims to follow the Web-App Architecture. The *presentation* and *logic* tiers will be built using ReactJS with the communication to and

from the *data* tier will be written in Node.JS (with the database itself written in SQL). In addition to providing a clear separation of languages and functionality the three-tier Web-App Architecture also follows the Model-View-Controller (MVC) architectural pattern (Kruchten, 1995). However, the MVC architecture is complicated by the use of ReactJS, which blurs the line between the *Model* (logic) and *View* (presentation) tiers. This is a problem that will be addressed in Section 4.

2.3. CODEX System Requirements

The requirements for CODEX were generated by identifying the features that would solve the problems with D&D that were discussed in Section 1. Once the features were generated, a *MoSCoW Analysis* was performed on the list. The MoSCoW Analysis asks the developer which features *must exist* in order for the system to be functional, then what *should* be in the system in order to make it viable in the market. Once those two lists have been defined what *could* the system posses that would be a nice addition to the other features. Finally MoSCoW asks what *will not* be in the system at any point.

Table 1: MoSCoW analysis for CODEX, showing the differing importance of wanted features to the system as a whole.

Must have	**Should have**	**Could have**	**Wont have**
Track Combat	Multiple Campaigns per DM Account	Game Scheduling	Full descriptions of D&D
Accounts	Settings	Items	Pre-made Settings & Campaigns
Encounter Planning	Characters		Inter-account Direct Messaging
Random Encounters	WorldWiki		
Populated Database	Run, Save & Delete Games		
Session Planning			

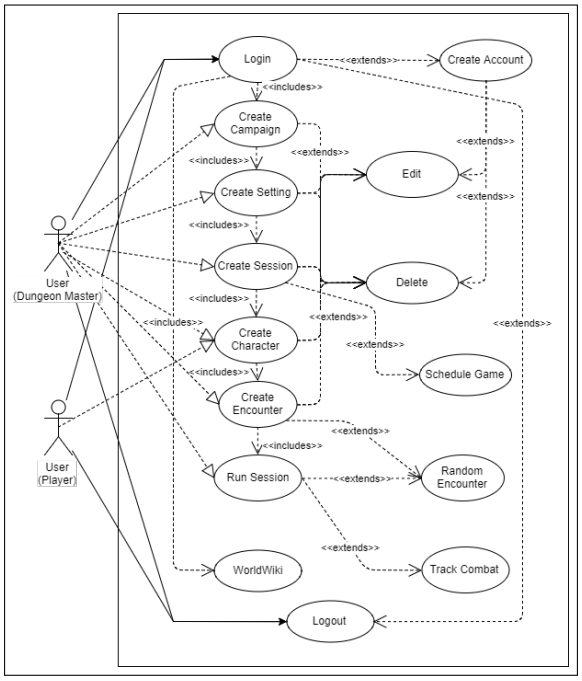

Figure 4: CODEX Use Case Diagram showing the accessibility to functions and the flow of information between them

2.4. Use Cases

Another critical part of the Design of CODEX is planning how the functionality of the system will flow. Use Cases and the Use Case Diagram inform a developer how a data is expected to flow between systems during a function. See Figure 4 for the CODEX Use Case Diagram. This creates a road map for the development of a system to follow, increasing the speed of development due to the fact that developers already know what is expected of each function. At this current time, CODEX has 25 Use Cases. Minor functionality such as changing screens and submitting a form does not have their own Use Case, but are included in related Use Cases.

2.5. Low-Fi Prototyping

Discuss:

- What is Low-Fi
- Purpose of Low-Fi
- Think-Aloud Evaluations & Results
- Changes to GUI made because of Think-Aloud

Use Case 6 (Edit Campaign)

USE CASE NAME	Edit Campaign	
Goal in Context	DM edits the details of a Campaign they have already created.	
Scope & Level	Dungeon Master System	
Preconditions	DM already has created a Campaign on their account	
Success End Condition	DM edits details about their campaign and the changes are saved to the database	
Failed End Condition	Changes fail to save to the database. DM must attempt their changes again at a later date.	
Primary Actor	Dungeon Master (DM)	
Trigger	DM selects the "Edit Campaign" button	
SUCCESS SCENARIO	**Step**	**Action**
	1	DM is redirected to the Edit Campaign screen
	2	DM changes the data about the campaign that they have previously entered.
	3	DM selects the "Submit Changes" button, the new data is sent off to the Database and changes are saved. Confirmation code is returned to Codex.
	4	DM is redirected to the Homepage, with pop-up message informing them that changes have been saved.
ALTERNATIVE SCENARIO	**Step**	**Branching Action**
	1a	DM is not redirected, will have to try again later
	3a	Changes to the Campaign are not sent to the database/fail to save/success code failed to be returned
	3b	Codex presents DM with error pop-up saying that changes failed to save, check connection and try again later. Redirected to homepage
	4a	DM is not redirected and/or success message not presented.
	4b	DM will need to manually return to homepage
	4c	DM will have to manually check Campaign details are correct.
RELATED INFORMATION		
Priority	Low	
Performance Target	0.7 seconds	
Frequency	Low	
Subordinate Use Cases	None	
Channel to Primary Actor	Edit Campaign Screen	
Secondary Actors	Database, Codex	
Channel to Secondary Actors	Internal System Communication	
OPEN ISSUES	None	
SCHEDULE	N/A	
AUTHOR	Christopher Irvine (23/11/17)	

Figure 5: The Use Case for Editing an existing Campaign within CODEX

2.6. Med-Fi Prototyping

Show the current med-fi prototypes built using MarvelApp.

2.7. Object Orientated Design

Discuss:

- What is OOD?
- What is the purpose of an OOD Diagram?
- Show the OOD Diagram for CODEX

3. CODEX Implementation Plan

Discuss:

- Recap the Design Section that is directly related to Implementation (i.e. Choice of Languages/Architecture/OOD)
- Link to new Gantt Chart (see Appendix 7)
- Show command line modelling for combat? Screenshot of outputs?
- Reiterate Agile Solo

4. Outstanding Issues

Outstanding Issues:

- Coursework taking up more time than expected, a couple of lost development days
- Haven't addressed the questionnaire mentioned in the proposal, pushed back to Design 3 and Development 2

4.1. Solutions

- already simplified the MVP to help with the coursework issue
- need to develop the questionnaire and run it through ethics approval, use Christmas break to start this process

5. Conclusion

Discuss:

- overall happy with the current development of CODEX

- iterate the importance of holiday development time
- summarise the issues and solutions

A. Dungeons and Dragons Example

B. Gantt Chart - Original

C. Gantt Chart - New

References

App, M. (2013). Marvel app.

De Groef, W. (2016). *Client- and Server-Side Security Technologies for JavaScript Web Applications*. Phd thesis, Faculty of Engineering Science.

Inc., F. (2017). Reactjs.

Kruchten, P. B. (1995). The 4+ 1 view model of architecture. *IEEE software*, 12(6):42–50.

Mearls, M. and Crawford, J. (2014a). *Dungeon Master's Guide*. Wizards of the Coast.

Mearls, M. and Crawford, J. (2014b). *Player's Handbook*. Wizards of the Coast.

Mercer, M. (2016). Setting up your gamemaster's screen! (gm tips w/ matt mercer).

Node.js, F. (2017). Node.js.

Nyström, A. (2011). Agile solo-defining and evaluating an agile software development process for a single software developer. Masters thesis, Department of Computer Science and Engineering.

Perkins, C. (2014). *Monster Manual*. Wizards of the Coast.

Software, F. C. (2011). Trello.

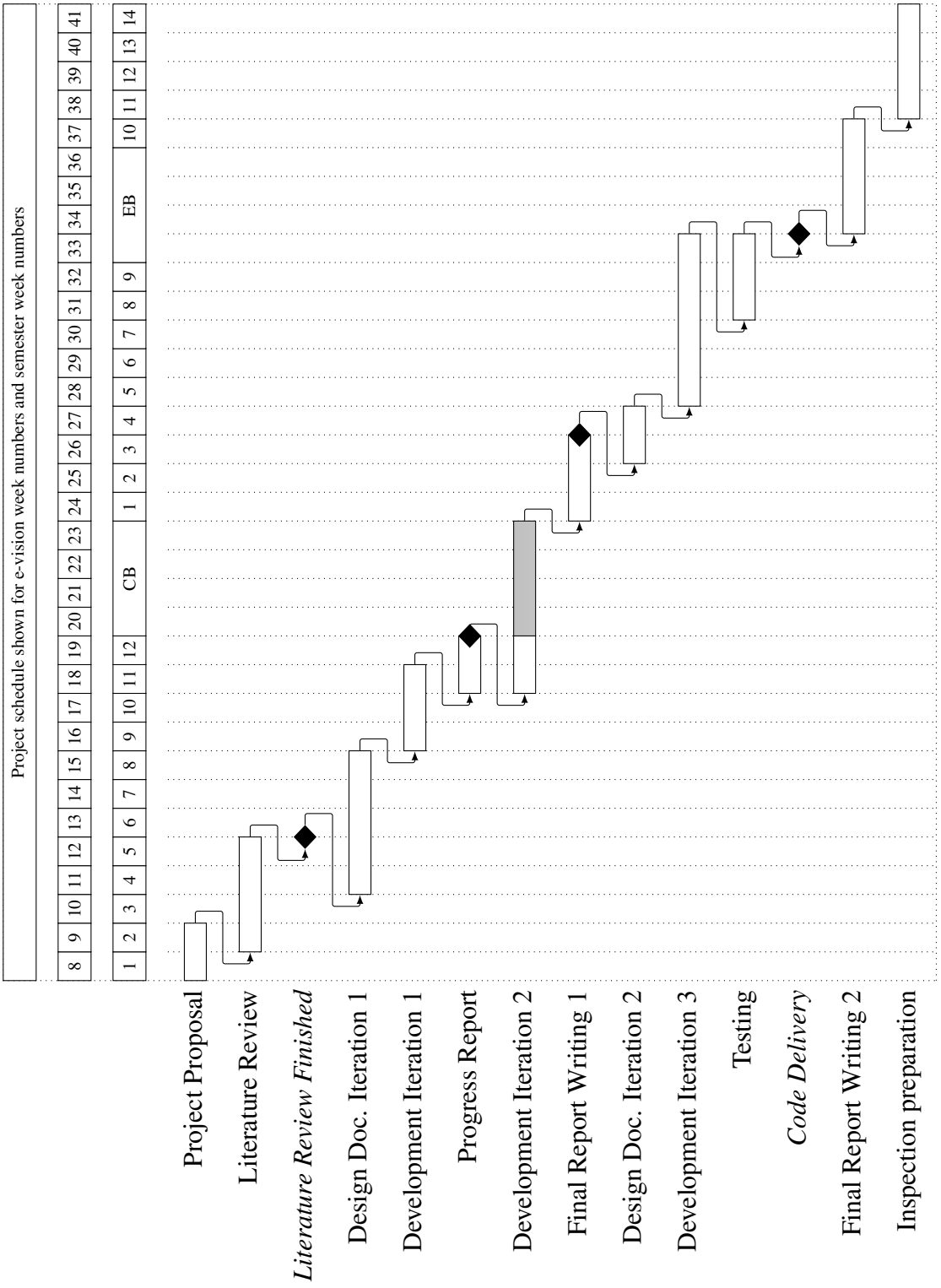

Figure 6: CODEX Gantt Chart, outlining the major tasks and deliverables

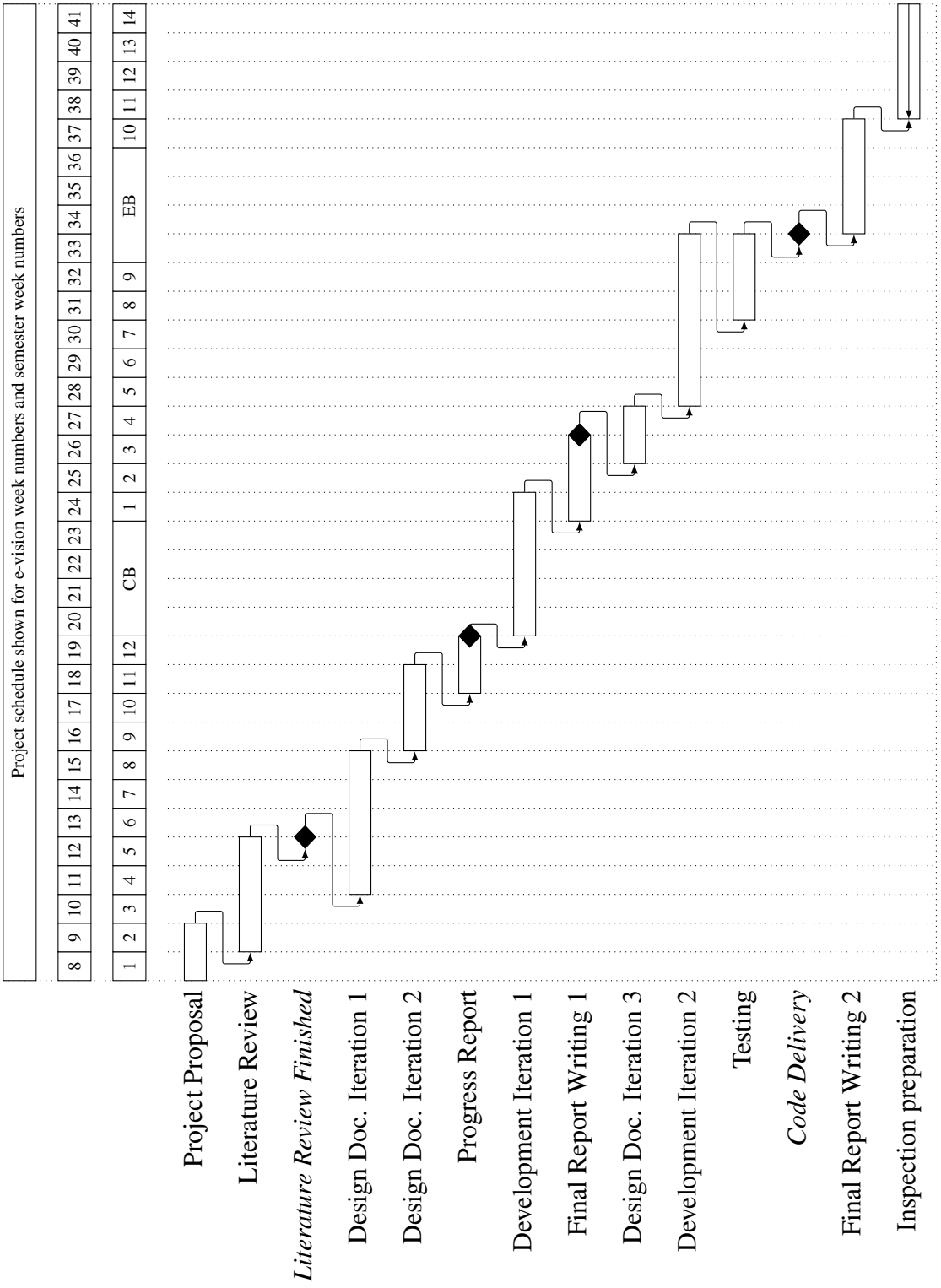

Figure 7: CODEX Gantt Chart, outlining the major tasks and deliverables