# *Requirement Analysis: DinoCode – A children's Integrated Development Environment*

*Written By Matt Shabaily, Daniel Carter, Hermela Atakelt Wubneh, Tom Austin, Osamende Ekhator, Yousef Mekkawy*

## Project Description

The group is developing an Integrated Development Environment (IDE) tailored for young computer science students aged 12-14. Python has been chosen as the primary language for development. The core concept is to create an educational tool that fosters a comfortable learning environment. Simplified UML class diagrams will aid in understanding object-oriented design principles, aligning with the educational needs of our target users.

To enhance the user experience, the design will incorporate child-friendly backgrounds, making the coding environment visually appealing. Users can also customize themes for a personalised touch. Educational hints and popups will offer guidance and encouragement during programming sessions, including reminders to take breaks for healthy learning habits.
By combining Python development, simplified UML diagrams, child-friendly design, and educational prompts, the IDE aims to provide an engaging and supportive platform for young students to explore computer science and programming.

## Aims and Objectives

*Aims:*

- To create a platform that aids children in learning programming.
- To facilitate children's learning of object-oriented methodologies, through use of a view that presents classes and objects written in the source code as diagrams akin to UML class diagrams - accessible separately from the text editor.

*Objectives:*

- To implement a tool that can take text inputs and produce source code files.
- To implement a system that can allow for thematic customisation, specifically in the colouring of UI elements, written text, and in the background over which text is written.
- To implement hints and popups that inform and educate the user.
- To implement a feature that allows for building source code into object code, and executable applications.
- To implement error handling and syntax highlighting.
- To implement a console in which log outputs can be displayed to the user.
- To implement a visual representation for classes and objects in a way visually accessible to young programming students.
- To implement links that show relationships between objects and classes.

## Key literature and background reading

During research, it was discovered that there is often little, or no testing performed. Thus, we must ensure that we perform adequate testing to increase the chances of a bug-free successful software product. "Only 47% of the projects which claimed to have JUnit tests in the survey actually had intervals showing tests. For the other 53%, their developer did not execute, read, or modify a single test within five months."[1]
A study found during research recognised that young students often use conditional infinite loops when the task could have been more succinctly solved with a simple conditional, going on to conclude that by creating such habits at a young age it is likely that students will continue to use bad practices as they advance. A debate could be had on whether the benefit of providing simple tools which generate interest in coding is beneficial if it festers poor coding habits. More succinctly, from the conclusion of the study - "While we are pleased with the willingness of students to engage in programming by using Scratch and with the technical skills that they develop, we are disturbed by the habits of programming that we uncovered." [2]
In researching the logistics of pair programming, a University of Utah study was found positing that (despite suffering a 15% overhead in time spent on implementation) students that used the approach reported a "psychologically soothing" sense that "no major mistakes had been made"[3] thanks to partner review. Compounding on this suggested improvement in moral, further research found that the knowledge of impending review can lead programmers to "avoid making the mistakes"[4] - out of a greater awareness on their code. It is from these sources that we have concluded that pair programming is to be used

## Development and Implementation Summary

Proposed System Components:
Our application is to be split into two main components: the screen in which text is to be inputted by the user, and the screen in which the generated diagrams are to be displayed. Navigation between these two screens should be implemented as seamlessly as possible, to this end we are to use a navigation bar sitting at the top of the window.

Also included in our design is a terminal from which outputs can be logged to the user, this is to be accessible through a button – triggering its extending out to fill a portion of the screen.
The educational segment of the application will be encompassed primarily through tooltip highlights, with certain elements prompting hints on hovering over them. Any more prominent messages, including those concerning reminders to take breaks, will be conveyed to the user through pop-up boxes.
The remainder of the components involved in our system are concerned with the settings and customisation features, which will be accessible via a drop-down box – with options that open a new smaller window where features can be altered by the user.

**Proposed Development Environment:**
We plan to use version control via Git, with each team member to be assigned a branch where they can implement features. This will allow for a clear record of development to be kept[5], with it being easy and straightforward to identify the author of code[6] – should any team member need contact the original author
To prevent dependence on any individual team member in developing features, pair programming will be used – both in a physical capacity or over communication online (with frequent driver/observer switches). This will help spread knowledge of the code base throughout the team

Our chosen programming language is python, a decision informed by its powerful module base – suiting our needs well. Chosen python modules include *tkinter*[7] for GUI development, *subprocess*[8] for code compilation and *unittest*[9] for automated testing. This approach was also chosen thanks to the familiarity with the language within the team, meaning that the labour of programming can be shared freely across all members.

**Data Sources**

Possibly the most prevalent data that is to be used comes from the official syntax documentation of any languages that we will include support for in our error checking and syntax highlighting. This data will, where possible, be accessed from official websites and parsed into a form that we can test code against on demand from the main application.

Other data that we plan to access include publicly available logos and graphics. These resources, currently, are expected to comprise mostly of logos for programming languages – for use in the user-interface. These graphics will be gathered through official channels, with the majority being available for download from language documentation websites.

Outside of licensed logos, any navigational graphics or UI elements will be generated by our team – for the sake of ensuring the project has as few links to external media as possible. This will prevent the need for a lengthy accreditation section which may impact the user experience.

The main bulk of data that is to be generated by our group itself is expected to come from the educational hints that will appear periodically, these hints will be produced independently – under an amount of reference to available online resources when necessary. Some standouts in terms of reference materials include: W3schools, Code Academy, and Free Code Camp – though these platforms will not be taken from verbatim.[10]

All data will be used in accordance with the licences specified, with proper accreditation being done where necessary.

**Testing & Evaluation**

There will be many testing stages to ensure our product meets the specified requirements. The IDE will be split into three stages and each stage will be tested and evaluated separately, using a checklist of requirements. If all these requirements are met, without bugs and unexpected behaviour, then the product can be deemed successful. Each section builds upon the last providing more advanced features and therefore a more rigorous set of tests.

Due to the nature of the problem and the fact that it will be split into three sections, an iterative approach to testing is probably best. Each section can be considered an iteration of the last, adding slightly more. This means that the full combined product will be tested regressively alongside the iterative tests of individual modules. The use of version control (GitHub) lends nicely to this style of testing as well. If an iteration is faulty and the nature of the fault cannot be found, an earlier working version can be restored, and a different approach can be taken.

Boundary testing will be used, among other methods, for verifying the product against requirements. This would entail seeing how the IDE deals with maximums and minimums. Many IDEs allow for multiple files to be open at once and you can flick through them easily by clicking the tab itself. However, there must be a limit to the number of tabs that can be opened at once. This is because there is only so much that you can display without fonts becoming shrunken and unreadable. Many IDEs have a little drop-down arrow which allows you to see the rest of the tabs when this limit is exceeded, which would be a nice feature for the IDE. This would involve boundary testing as, what happens on the cusp of the limit and when the limit is exceeded, will be tested, to ensure the proper expected behaviour.

The IDE should be easy to use and therefore it must be tested rigorously to ensure that it is user-friendly. This can be tested by asking new coders to volunteer and attempt to code using the IDE. This feedback can then be used to tailor the experience to new coders, who are the target demographic. If the peer review is kept up throughout production, then the final product won't stray from its intended purpose.

**Ethical Considerations**

***Informed Consent of Participants:***

Children will be used in research and testing the application. When involved in research, participants will be explained the purpose of the research and what is involved before participating for informed consent. To address the ethical implications of involving minors in the project, permission from parents/guardians will be required with signatures. They will be free to leave and withdraw any data drawn at any given time.

***Protection of Participant Data:***

Data will be stored securely on Teams with access restricted to group members. To keep the participants data anonymous, they will be given alias numbers when referred to.

***Ethical Guidelines for Testing:***

Participants will be recruited from current connections, such as family members, ensuring comfort for participants and a smoother testing process.

***Compliance with University Ethics Policy Responsible Use of Data Sources:***

We will adhere to the University of Liverpool Academic integrity policy by not plagiarising and maintaining references and citing any sources used. Data used will be obtained by legitimate sources verified users and reputable software development platforms.

***Data Privacy and Security in the IDE:***

To ensure the IDE promotes privacy and security for users it will comply with industry standard laws like GDPR and COPPA. Users will be required to accept a privacy policy before using the software, ensuring they agree to the terms. To protect users' personal information within the platform, we will ensure no code or general data will be shared with third parties and security protocols will be used.

***Accessibility and Inclusivity:***

Design considerations will be made to accommodate learning needs, including adjustable font sizes and colour contrast.

***Copyright and Licensing:***

To keep in line with copyright and licensing laws we will reference all design aspects regardless of the license will anyway as its best practice.

**BCS Project Criteria**

***An ability to apply practical and analytical skills gained during the degree programme:***

- With the graphics element, there may be some matrix manipulation (resizing and rotating.)
- Boundary testing on text (maximum and minimum font sizes.)
- Iterative testing (adding modules,) regression testing (all modules together) and version control (GitHub.)
- Good HCI – the software will be designed to be easy to use. It should be intuitive, and users should be able to make educated guesses on most, if not all, of the functionality.

***Innovation and/or creativity:***
An IDE that allows you to see the relationships between your items is unique and creative. Furthermore, the kid-friendly approach to the IDE is innovative as it solves an age-old problem – IDEs are complicated and scary. If the IDE can be made easy to use, by removing advanced features and fitting it with fun and customisable colour themes, then it can be enjoyed by a younger and newer audience.

***Synthesis of information, ideas and practices to provide a quality solution together with an evaluation of that solution:***
The unique concept of a graphical IDE that shows the relationships between classes in a UML style was synthesised as a quality solution for new programmers who are struggling to structure their code and lose track of what is affected by what. This idea was further built upon to develop other useful features for new programmers to make a user-friendly IDE – a real toolbox of helpful features for the new coder. This solution will be evaluated using a requirements checklist, built up from the nature of the problem itself, 'how can we make an IDE simplistic?'

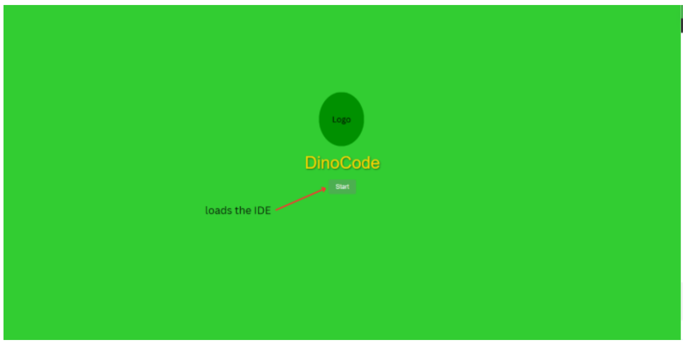
***That your project meets a real need in a wider context:***
IDEs are complex and scary, but they don't have to be. The product genuinely meets a real need, as new programmers need something friendly. Something that they can dive into without needing to have lots of pre-existing IDE based knowledge and experience. Equipped with a library of helpful tools, this IDE fills the gap in the market for lightweight, easy-to-use, visual IDEs.

***An ability to self-manage a significant piece of work:***
The workload will be distributed evenly throughout the team and status reports will be used to ensure everyone is on track. Weekly meetings also help keep things going at a steady pace and give the members a chance to alert the team of any potential struggles and concerns. GitHub will be a primary force of progress in the project, as code can be peer-reviewed before being merged into the main branch.

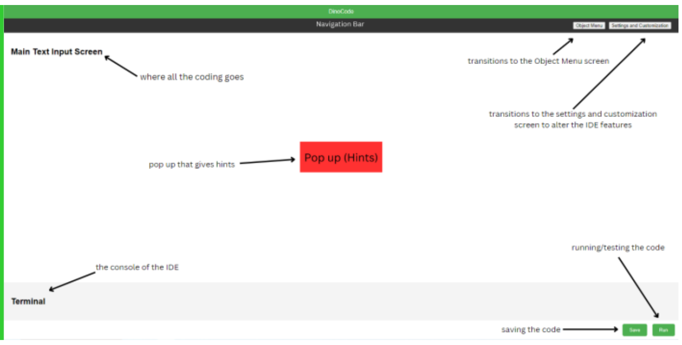
***Critical self-evaluation of the process:***
As well as reviewing each other's code (GitHub merge requests,) we will need to review our code. Each module will be moulded around the requirements and tested regularly against a set of prerequisites. This extensive testing will be our way of self-evaluating the code that we write. With each stride that we make, as a team, in any aspect of the problem (requirements, design, development etc…), we will make a note of what we (as individuals) have completed and what problems need to be passed on to other team members.

**UI/UX Mock-up**

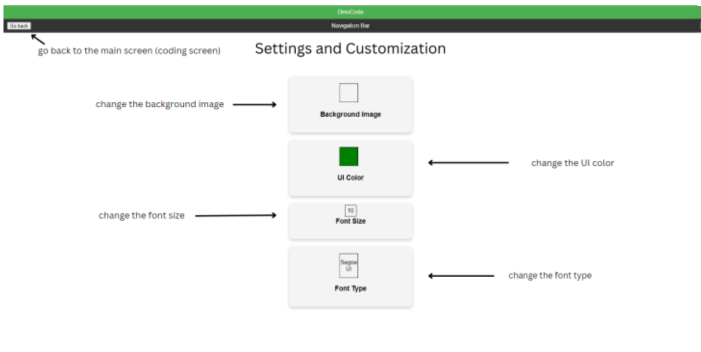
**Main Menu**



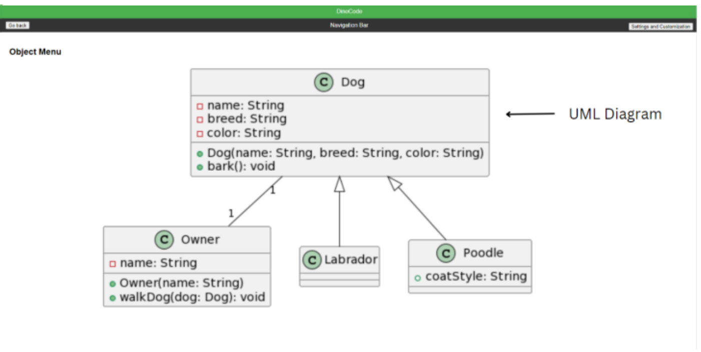
**Main Text Input Screen**



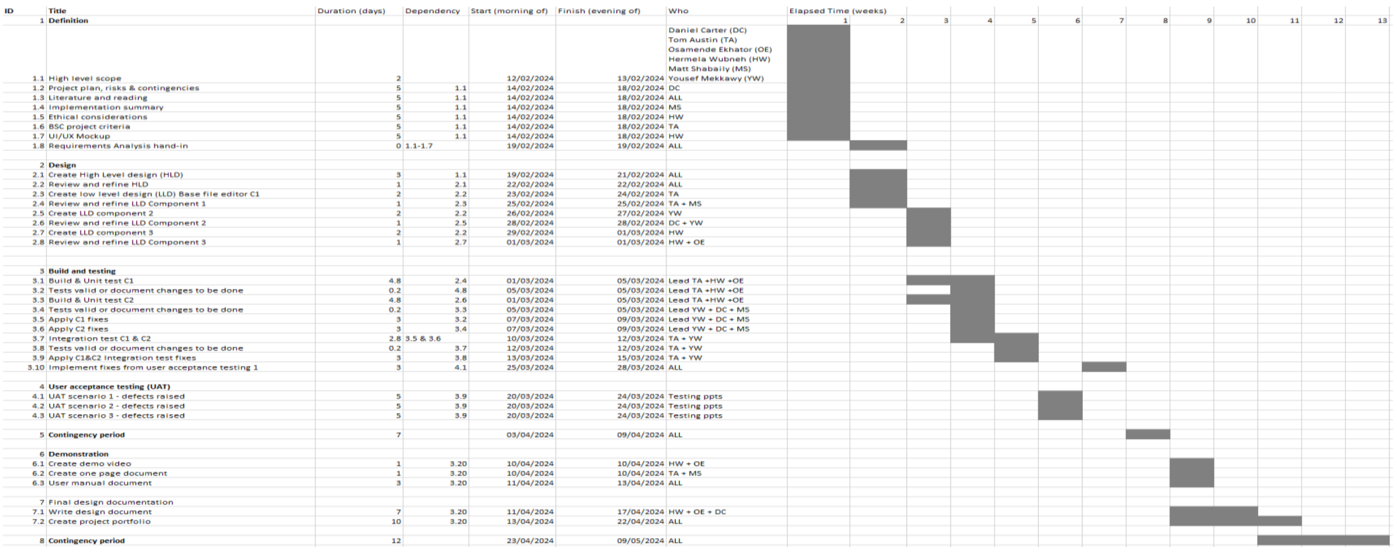
**Settings and Customization**



**Object Menu**



**Project plan, Risks & Contingency plans:**
Below is the Gantt chart outlining the key tasks which will be performed over the course of the project.



| ID | Title | Duration (days) | Dependency | Start (morning of) | Finish (evening of) | Who | Elapsed Time (weeks) | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | **Definition** | | | | | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
| | | | | | | Daniel Carter (DC)<br>Tom Austin (TA)<br>Osamende Ekhator (OE)<br>Hermela Wubneh (HW)<br>Matt Shabaily (MS) | ▨ | | | | | | | | | | | | |
| 1.1 | High level scope | 2 | | 12/02/2024 | 13/02/2024 | Yousef Mekkawy (YW) | ▨ | | | | | | | | | | | | |
| 1.2 | Project plan, risks & contingencies | 5 | 1.1 | 14/02/2024 | 18/02/2024 | DC | ▨ | | | | | | | | | | | | |
| 1.3 | Literature and reading | 5 | 1.1 | 14/02/2024 | 18/02/2024 | ALL | ▨ | | | | | | | | | | | | |
| 1.4 | Implementation summary | 5 | 1.1 | 14/02/2024 | 18/02/2024 | MS | ▨ | | | | | | | | | | | | |
| 1.5 | Ethical considerations | 5 | 1.1 | 14/02/2024 | 18/02/2024 | HW | ▨ | | | | | | | | | | | | |
| 1.6 | BSC project criteria | 5 | 1.1 | 14/02/2024 | 18/02/2024 | TA | ▨ | | | | | | | | | | | | |
| 1.7 | UI/UX Mockup | 5 | 1.1 | 14/02/2024 | 18/02/2024 | HW | ▨ | | | | | | | | | | | | |
| 1.8 | Requirements Analysis hand-in | 0 | 1.1-1.7 | 19/02/2024 | 19/02/2024 | ALL | | ▨ | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | |
| 2 | **Design** | | | | | | | | | | | | | | | | | | |
| 2.1 | Create High Level design (HLD) | 3 | 1.1 | 19/02/2024 | 21/02/2024 | ALL | | ▨ | | | | | | | | | | | |
| 2.2 | Review and refine HLD | 1 | 2.1 | 22/02/2024 | 22/02/2024 | ALL | | ▨ | | | | | | | | | | | |
| 2.3 | Create low level design (LLD) Base file editor C1 | 2 | 2.2 | 23/02/2024 | 24/02/2024 | TA | | ▨ | | | | | | | | | | | |
| 2.4 | Review and refine LLD Component 1 | 1 | 2.3 | 25/02/2024 | 25/02/2024 | TA + MS | | ▨ | | | | | | | | | | | |
| 2.5 | Create LLD component 2 | 2 | 2.2 | 26/02/2024 | 27/02/2024 | YW | | | ▨ | | | | | | | | | | |
| 2.6 | Review and refine LLD Component 2 | 1 | 2.5 | 28/02/2024 | 28/02/2024 | DC + YW | | | ▨ | | | | | | | | | | |
| 2.7 | Create LLD component 3 | 2 | 2.2 | 29/02/2024 | 01/03/2024 | HW | | | ▨ | | | | | | | | | | |
| 2.8 | Review and refine LLD Component 3 | 1 | 2.7 | 01/03/2024 | 01/03/2024 | HW + OE | | | ▨ | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | |
| 3 | **Build and testing** | | | | | | | | | | | | | | | | | | |
| 3.1 | Build & Unit test C1 | 4.8 | 2.4 | 01/03/2024 | 05/03/2024 | Lead TA +HW +OE | | | ▨ | ▨ | | | | | | | | | |
| 3.2 | Tests valid or document changes to be done | 0.2 | 4.8 | 05/03/2024 | 05/03/2024 | Lead TA +HW +OE | | | ▨ | ▨ | | | | | | | | | |
| 3.3 | Build & Unit test C2 | 4.8 | 2.6 | 01/03/2024 | 05/03/2024 | Lead TA +HW +OE | | | ▨ | ▨ | | | | | | | | | |
| 3.4 | Tests valid or document changes to be done | 0.2 | 3.3 | 05/03/2024 | 05/03/2024 | Lead YW + DC + MS | | | | ▨ | | | | | | | | | |
| 3.5 | Apply C1 fixes | 3 | 3.2 | 07/03/2024 | 09/03/2024 | Lead YW + DC + MS | | | | ▨ | | | | | | | | | |
| 3.6 | Apply C2 fixes | 3 | 3.4 | 07/03/2024 | 09/03/2024 | Lead YW + DC + MS | | | | ▨ | | | | | | | | | |
| 3.7 | Integration test C1 & C2 | 2.8 | 3.5 & 3.6 | 10/03/2024 | 12/03/2024 | TA + YW | | | | ▨ | | | | | | | | | |
| 3.8 | Tests valid or document changes to be done | 0.2 | 3.7 | 12/03/2024 | 12/03/2024 | TA + YW | | | | | ▨ | | | | | | | | |
| 3.9 | Apply C1&C2 Integration test fixes | 3 | 3.8 | 13/03/2024 | 15/03/2024 | TA + YW | | | | | ▨ | | | | | | | | |
| 3.10 | Implement fixes from user acceptance testing 1 | 3 | 4.1 | 25/03/2024 | 28/03/2024 | ALL | | | | | | | ▨ | | | | | | |
| | | | | | | | | | | | | | | | | | | | |
| 4 | **User acceptance testing (UAT)** | | | | | | | | | | | | | | | | | | |
| 4.1 | UAT scenario 1 - defects raised | 5 | 3.9 | 20/03/2024 | 24/03/2024 | Testing ppts | | | | | | ▨ | | | | | | | |
| 4.2 | UAT scenario 2 - defects raised | 5 | 3.9 | 20/03/2024 | 24/03/2024 | Testing ppts | | | | | | ▨ | | | | | | | |
| 4.3 | UAT scenario 3 - defects raised | 5 | 3.9 | 20/03/2024 | 24/03/2024 | Testing ppts | | | | | | ▨ | | | | | | | |
| | | | | | | | | | | | | | | | | | | | |
| 5 | **Contingency period** | 7 | | 03/04/2024 | 09/04/2024 | ALL | | | | | | | | ▨ | | | | | |
| | | | | | | | | | | | | | | | | | | | |
| 6 | **Demonstration** | | | | | | | | | | | | | | | | | | |
| 6.1 | Create demo video | 1 | 3.20 | 10/04/2024 | 10/04/2024 | HW + OE | | | | | | | | | ▨ | | | | |
| 6.2 | Create one page document | 1 | 3.20 | 10/04/2024 | 10/04/2024 | TA + MS | | | | | | | | | ▨ | | | | |
| 6.3 | User manual document | 3 | 3.20 | 11/04/2024 | 13/04/2024 | ALL | | | | | | | | | ▨ | | | | |
| | | | | | | | | | | | | | | | | | | | |
| 7 | Final design documentation | | | | | | | | | | | | | | | | | | |
| 7.1 | Write design document | 7 | 3.20 | 11/04/2024 | 17/04/2024 | HW + OE + DC | | | | | | | | | ▨ | ▨ | | | |
| 7.2 | Create project portfolio | 10 | 3.20 | 13/04/2024 | 22/04/2024 | ALL | | | | | | | | | ▨ | ▨ | ▨ | | |
| | | | | | | | | | | | | | | | | | | | |
| 8 | **Contingency period** | 12 | | 23/04/2024 | 09/05/2024 | ALL | | | | | | | | | | | ▨ | ▨ | ▨ |

**Risks And Contingencies:**

| Risks | Contingencies | Likelihood | Impact |
|---|---|---|---|
| Hardware failure | Ensure frequent backups of code on alternative pieces of hardware, and on online repositories | Low. It is unlikely for a catastrophic failure of hardware, however still a possibility. | Medium. Although threatening serious delay,s the availability of backed-up university machines pre-empts these issues |
| Software failure | Ensuring appropriate time and due diligence is given to unit testing of our project. | High. It is likely there will be minor errors which become apparent during testing which we may be unable to fix within the time constraints. These errors should not be critical for execution of the program. | High. Other non-critical failures can be assessed as to their impact as the program is developed, and either fixes implemented, mitigated **or** ignored. |
| Running out of time/Scope creep | Ensuring a reasonably set, unchanging scope, determined at the start of the project – with optional features to be completed should time allow. | High. While it is likely we run out of time for completing all ideal features as tasks will expand to fill time available, it is unlikely that we have a non-working program. | Low. Given reasonable planning, key features should be implemented, with tests being performed as part of development, rather than after the program has finished. |
| Conflicting understanding of original requirements/later updates | Holding regular full team meetings discussing intricacies of the project, and ensuring all team are up to date on the latest changes and advancements | High. Each member will quickly form their own schema of what the project is/involves, it is crucial we all have similar ideas and understanding of what we wish to implement and what will be left out. | High. If team members have conflicting views on what the project is, it may result in having to re-write parts of code, wasting time and reducing efforts in other areas. |
| Poor/lack of communication | Ensuring that any confusion is resolved by reference to meeting minutes, or addressed in Q/A during the meetings themselves | Low. Many methods of communications, including discord, can host clarification on project critical questions before beginning tasks. | High. There is a risk of labour repetition and delay if team members misunderstand designs and scopes |
| Team member availability | Ensuring that team members with currently low availability are not given time-critical tasks. | Medium. It is likely that there will be issues regarding team member availability due to other university requirements, part time jobs and other personal responsibilities. | Low. Other members can take on the time-critical tasks and allow the other team members to finish the work in their available hours. |
| Coding skill disparity | Establishing that teams' members are assigned tasks according to their ability. Some assuming | High. It is likely that some team members will have a much more in-depth knowledge of both software development, python and its libraries. | Medium. Although development rates will vary with team ability, less qualified members will gain chance to develop skills. |

| | | | |
|---|---|---|---|
| | supporting roles, focusing on documentation and minor coding | | |
| Lack of required module functionality | Ensuring that the scope is limited to functions feasible to our environment, with proper research into chosen modules | Low. Our chosen modules and language are well supported, though oversights are still a possibility that must be pre-empted | Medium. Since the IDE is aimed at children, it does not need the complexities of a full-fledged IDE |
| Inadequate user testing | Performing as much user testing as possible within ethical and time limits, though this is curtailed by the limitations around research involving children | High. Due to a lack of availability of participants in beta testing, we must test a lot of the project using adults who must assume the "perspective of a child". | High. Unfortunately, with a lack of user testing from different target audience participants, the program is likely to suffer a detachment with our user-base |

**References:**

1 Beller, M. et al. (2015) 'When, how, and why developers (do not) test in their IDES', Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, pp. 186–186. doi:10.1145/2786805.2786843.

2 Meerbaum-Salant, O., Armoni, M. and Ben-Ari, M. (2011) 'Habits of programming in Scratch', *Proceedings of the 16th annual joint conference on Innovation*
*and technology in computer science education*, pp. 172–172. doi:10.1145/1999747.1999796.

3 Cockburn Alistair, Williams-Laurie 'The Costs and Benefits of Pair Programming' - University of Utah Computer Science 2001

4 C. Jones, Software Quality: Analysis and Guidelines for Success, Boston. MA:International Thomson Computer Press, 1997.

5 Git (2005) – Git History | https://git-scm.com/book/en/v2/Git-Basics-Viewing-the-Commit-History

6 Git (2009) – Git Blame | https://git-scm.com/docs/git-blame

7 Python Docs – Tkinter | https://docs.python.org/3/library/tkinter.html

8 Python Docs – Subprocess | https://docs.python.org/3/library/subprocess.html

9 Python Docs – Unittest | https://docs.python.org/3/library/unittest.html

10 W3Schools (2024) https://www.w3schools.com/about/about_copyright.asp

Borse, J., Robles, E. and Schwartz, N., 2006. Designing for Kids in the Digital Age: Summary of research and recommendations for designers. In The 1st interaction Design & Children Conference.

Bennett, C., 2006. Keeping up with the kids. Young Consumers, 7(3), pp.28-32.