|  |  |
| --- | --- |
| Student number | 19003311 |
| Main text body word count | 1111 |
|  |  |
|  |  |
| **Reflective abstract (up to 150 words)**  Please help the marker give you useful, personally-tailored, feedback by completing the sections below. Nothing you write here will adversely affect your mark. | |
| What mark do you anticipate your essay will achieve? | I am hoping for above 70 marks for this report. |
| What do you think you did well in the writing of this essay? | I outlined the evaluation criteria well to analyse the modelling tool fairly based on laid out criteria. |
| What aspects of your essay do you think may need improvement? | More use of external resources and sources to back up my findings both from the critical review and reflective report. |
| What areas of feedback would be most helpful for you? | More information on my approach to the UML diagrams and whether they would be helpful in a large-scale enterprise system. |

PART I

I .1 INTRODUCTION

This critical review evaluates the effectiveness of using UML as a modelling framework and the software used to implement it in the development of the UWEFlix case study, specifically the DrawIO modelling tools. This review will examine the suitability of the modelling software in supporting the modelling process.

1.2 EVALUATION CRITERIA

The UML tool was evaluated using nine key criteria:

Usability evaluates the ease-of-use, user-interface and user-experience.

Functionality assesses the capability to support UML modelling and its notation.

Integration evaluates the integration with other tools.

Support assesses documentation and community support.

Cost examines the cost-effectiveness of the tool in comparison with other tools.

Customisability assesses the users ability to change preferences and add their own changes.

Collaboration examines if the tool can be used in collaboration with other members of a team.

1.3 EVIDENCE (WITH EXAMPLE ILLUSTRATIONS) AND ANALYSIS

DrawIO is a modern web-based UML modelling tool. The layout of the UI is sleek and simple, the tools and services are easily recognisable and in expected places. This makes the tool useful for both ‘modelling for analysis’ and ‘modelling for communication’, paradigms laid out by Chaudron, Heijstek, and Nugroho (2012), wherein models can be used as a loose sketch to understand the purpose of a software, or they can be used more rigidly to communicate between teams the architecture of the software.

The diagrams are easy to read including for people with differing sight abilities such as colour-blindness. Figure 3 clearly denotes the scope of the software, the actors, and the use cases accessibly.

DrawIO has a full suite of UML capabilities and most diagrams have templates ready-to-use, making the development of UML diagrams very quick and painless. One of the main critiques of rigid UML modelling is the time it takes to develop models. Alhumaidan (2012) expresses this in his discussion on the weaknesses of use-case diagrams, as they can take a lot of time to map and model, however DrawIO has use-case diagram templates to reduce wasted time. Furthermore, DrawIO has basic shapes that can be added to diagrams, making it much easier to create your own items such as interface socket-and-ball notation. For example, in Fig. 2, the interface circles are not the provided ‘Interface’ notation objects, but basic circle shapes. As DrawIO is open-source, it has plenty of support available from the community and official guides. This also makes it free to use, so it is naturally more cost effective than competitors such as Astah, which provides $6 per month plans, and Visio, which provides a GBP3.80 plan. In terms of customisability, DrawIO has some basic features the user can change in their preferences, however if the user would like to make further customisations, they can edit the source code directly by cloning the git repository. There are no plugins or extensions available for DrawIO, therefore any major changes must be done in the source code and recompiled. DrawIO does have collaboration abilities, as diagrams can be stored directly to cloud services such as Google Drive, and from there can be used in collaboration with a team. Finally, DrawIO models can be saved as non-proprietary XML, HTML, and SVG files which allow them to be edited in other modelling tools that support these formats, meaning a higher level of integration than other tools which save files as proprietary types.

1.4 CONCLUSIONS

DrawIO is an adaptive modern UML tool creating a multitude of models easily, with a high level of customisability due to the source code access; the capability for cloud-based collaboration; integration with other tools and services; a free-to-use code base, and a functional and effective UI/UX.

PART II

2 .1 INTRODUCTION

This report will outline the development process from requirements modelling to the final stage of the implementation of the UWEFlix enterprise software. During this time, I worked with a team to understand the requirements and build use cases for the software.

2.2 REFLECTIONS

There were a few highlights for what went well and what didn’t. The case study made it easy to understand the software and its purpose, and by extension to design class diagrams, use-case diagrams and other such models. This adds weight to the argument that making diagrams prior to beginning development reduces development time, as opposed to developing ad-hoc and stitching together ideas. The requirements were laid out in the case study well, so user stories were easy to extract and therefore implement. The Django framework also makes it easy to implement a web application due to much of the boilerplate being completed for you, such as authorisation models and traffic handling. This eliminated the need to handle sensitive user information such as passwords directly, as the framework handles this for you. To help with the development process, I made sure to stick to a stylesheet for the HTML to make the site look uniform and professional. I extended this stylesheet to the Python code to enforce a clean-code structure, as having a uniform and easy-to-read code base makes it much easier to look back on and make changes when necessary (Anaya, 2018). Finally, I kept to a loose but thorough testing strategy. In hindsight, instead of using an ad-hoc testing strategy, I would have used Django’s provided unit-testing module to thoroughly unit-test the software throughout development.

Due to not thoroughly researching the design of Django, some models and structures had to be changed during implementation. For example, using a User superclass with Cinema Manager, Account Manager, Club Representatives as child classes was not implemented, as this made it needlessly difficult for permission authentication. A reflection made would be to thoroughly investigate the framework before diving straight into the code to eliminate any unforeseen setbacks. Furthermore, as I was developing between multiple systems, I had to make extensive use of Git version controlling. Finally, getting Docker to work on all these systems was difficult as Docker requires a lot of permissions and external software to be installed before it can be used correctly, however the containerisation process was simple. I went with a light-weight Linux distribution with only Python installed, so that the container is small and cheap to run on any system with Docker. This is better than running the web service directly or on a VM, as Docker handles the Python package requirements and port handling, as opposed to manually installing the required packages.

2.3 CONCLUSIONS

Using an enterprise development model to develop software even of this small size proves to be beneficial as the structure of UML diagramming and maintaining structured and clean code cuts out a lot of time in the future, and ensures the software is readable, maintainable, and extendable in the future. Containerising the software makes sure it can be run on any system with Docker, eliminating version and machine-based issues.

REFERENCES

**Critical Review:**

[1] Chaudron, M.R.V., Heijstek, W. and Nugroho, A. (2012) How Effective Is Uml Modeling? an Empirical Perspective on Costs and Benefits. Software & Systems Modeling [online]. 11 (1), pp. 571-580. [Accessed 10 January 2023].

[2] Alhumaidan, F. (2012) A Critical Analysis and Treatment of Important Uml Diagrams Enhancing Modeling Power. Intelligent Information Management [online]. 4 (1), pp. 231-237. [Accessed 10 January 2023].

**Reflective Report:**

[1] Anaya, M. (2018) Clean Code in Python: Refactor Your Legacy Code Base. Birmingham: Packt Publishing Ltd.

[2]

Alhumaidan, F., , and , (2006) Experiences From Introducing Uml-based Development in a Large Safety-critical Project. Empirical Software Engineering [online]. 11 (1), pp. 555-581. [Accessed 10 January 2023].

Appendix A

https://gitlab.uwe.ac.uk/rj2-ingham/uweflix

Appendix B

<< Insert all modelling diagrams in this section. Make sure each figure has a figure title with which you cite each within the text body >>

Diagram

Description automatically generated

Fig 1. Business <<type>> diagram

Diagram

Description automatically generated

Fig 2. Interface Diagram

Diagram

Description automatically generated

Fig 3. Use Case Diagram

Diagram

Description automatically generated

Fig 4. Club Creation Sequence Diagram

Diagram, schematic

Description automatically generated

Fig 5. Account Management Component Spec Diagram