G2U - a second life for every treasure

Course ID.: CPE-334

Submitted By-

Chawit Pimapansri	(ID: 65070503411)
Sorawit Tonpitak	(ID: 65070503438)
Jutamas Kaewchuenchai	(ID: 65070503444)
Nichaporn Manachaiprasert	(ID: 65070503446)
Thanakit Chokbunsuwan	(ID: 65070503448)
Arita Tragulmalee	(ID: 65070503470)
Yuil Tripathee	(ID: 65070503480)
Tom Medhi Pannier	(ID: 67540460025)

Submitted To-

Department of Computer Engineering
in partial fulfillment of the requirements
for the completion of
CPE-334 Software Engineering course.

Supervised by-

Dr. Natasha Dejdumrong
Associate Professor
Department of Computer Engineering

King Mongkut's University of Technology Thoburi (KMUTT)
Bangkok-10140, Thailand
1/2024

Revision History

Revision	Date	Author(s)	Description
v0.1	2023-04-21	Yuil Tripathee	First release, add outline
v0.2	2024-11-05	Tom Medhi Pannier	Add EU market evaluation

Abstract

We would like to think about it later.

Keywords: *We would like to think about it later.*

Terms, Acronyms, and Abbreviations

Keyword	Description
Δx	displacement from x_0 to x_1.
Δt	time taken from t_0 to t_1.

Keyword	Description	Keyword	Description
Δx	displacement from x_0 to x_1.	Δt	time taken from t_0 to t_1

Contents

List of Tables

List of Figures

Part I

Project Description

Chapter 1 Introduction

1.1 Background and Motivation

As of 2024, we stand in the face of turbulent geopolitics, frequent extreme weather events, and escalating living cost all around the world. And we chose to make a small step today towards a greater impact tomorrow. Unreasonable consumer demands is believed to be one of the leading factors that is keeping us behind on our goal of developing sustainable societies and planet. UN SDG goal 12 promotes sustainable consumption and production patterns, ensuring efficient use of natural resources.
This inspired us to found G2U. We are developing on online e-commerce platform where every treasure has a find its second life, to a new owner. Finding the gap in this nascent niche market, we intend to incorporate lean startup model to gain market knowledge & feedback to the maximum. Then, our agile engineering team will capitalize on the market input swiftly to capture the sizable market share.

1.2 Market study

1.2.a SEA Market

1.2.b EU Market

1.3 Scope of work

1.4 A dummy section

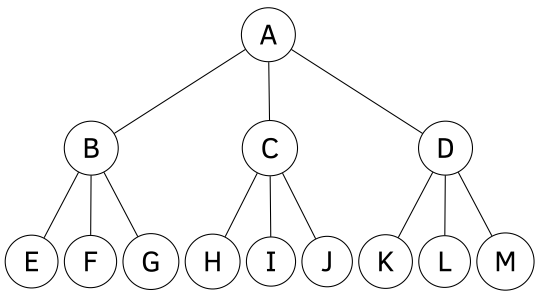

Figure 1.1: The proposed tree structure for the array implementation.

Chapter 2
Project Management

TODO: Comparative analysis of each methodology and what they deliver

2.1 Incremental funding methodology

Used for high level decisions.
TODO: Lean startup and cash model [1]
Instagram acquisition story on 2012 on $ 1 billion having 13 full time employees only. CPC from $ 0.5 to $ 3.5, CPM for $ 2 to $ 20 and SPE from $ 0.03 to $ 0.08. [2]
TODO: Project management (lecture 1) -> define scope, timeline, budget.
TODO: Communication and Rationale Management.

2.2 Agile Method with Kanban Tool

Used for low level decision and workflow orchestration.

Part II

Requirements

Chapter 3
Requirements Elicitation

3.1 Elicitation Techniques

TODO: Before analyzing the system, various technique are employed to gather its requirements.
TODO: Explain:

- Interviews
- Questionnaires
- Workshops
- Observation
- Prototyping

3.2 Stakeholders

3.3 System Analysis - Data Flow

TODO: Data Flow diagram

3.4 Use Case Analysis

TODO: Here is a breakdown of the main use cases for the system, along with involved actors.

- Actors
- Use Cases

For clear observers, it is evident that use case diagram present here is not in line with the data flow diagram. This is the result of agile practice. We kept data flow analysis out of SDLC loop as it is redundant to sequence diagram (below). We can effectively model both system behavior, reaction (in terms of data changes and transfers) much effectively across our implementation team.

3.5 Functional Design

TODO: Here are some functional requirements (example)

- User Registration
- Tutor Scheduling and Availability
- Online class

TODO: Translate this to user story when doing Kanban
TODO: Each functional requirements should have details and implementation in description list

3.6 Other Non-functional requirements

TODO: Quantize these requirements

- Scalability
- System Availability
- Security
- Usability
- Performance

3.6.a Mandated constraints

Examples include: Economics

3.6.b Regulatory compliance

Chapter 4
Usability Requirements

[3]

Part III

Design and Development

Chapter **5**

Systems Analysis and Design

5.1 Software Analysis

5.1.a Class Diagram

TODO: class diagram

5.1.b Components Diagram

5.1.c Sequence Diagram

5.1.c.1 Limitations

Unconventional implementation: Coding syntax such as including SQL as part of modeling app sequences. This is not part of best practice as specified in the UML specifications. However, it serves us more effectively as the means of communication for different technical sub-teams (front end, backend and the deployment team). Our rationale behind this was to address the knowledge gap our current development team has. For the further updates, we intend to move up to the standard specification as knowledge gap is shortened and the backend team is ready to receive requirements inputs using the top level description.

Less flexibility to stack changes For the current timeline, we chose the backend stack to adapt changes as quickly as possible. However, if the requirements changes results in change of technical stack (for example, we switch from HTTP REST & SQL interface to GraphQL &gRPC); our specification for interaction sequences will be obsolete. The SQL-inspired specification cannot address changes to other modern day NoSQL schema (such as document store, column store). Therefore, if we change the database structure the specifics in the sequence diagram regarding SQL interface should be changed.

5.2 Systems Design

5.2.a Demonstration model

5.2.b Full scale production model

Chapter 6
Implementation

6.1 Low Code

6.2 Prototyping

6.3 Coding

6.4 Systems Integration

Part IV

Test and Evaluation

Chapter 7
Evaluation of Outcomes

7.1 Testing Methodologies

Testing -> Second part of the course!

7.2 Results

7.3 Discussion

Chapter 8

Conclusion

8.1 Discussion

8.2 Future Work

8.3 Recommendation

8.3.a Opportunities

Secret to quality software We learned (in a somehow hard way) that better code can make program better, but not a better software.

8.3.b Challenges

If we had to do it over again, we learned that following challenges should be solved before the implementation phase begins:

Line of communication We have when programming team members are unfamiliar to each other due to some reasons (such as unmatched learning curve). Front end and back end teams have different backgrounds that shape their idea of integrating two ends together. There are some workable solutions to try out such as add a personnel specializing in full stack development. Role of such person would be to develop interfaces in the back end part and sharing data directly to the front-end team as they like. In this case both parties can maintain consistency when there is changes to schema or business logic. However, this luxury might not be available all the times. Therefore, a common documentation such as shared OpenAPI specification [4] is found to be better alternatives. In order follow this approach, either team (front end or back end) has to put their desired specification in a shared document. Every time there is a breaking changes from either team, it has to go through this document first prior to code changes.

Retaining quality user's feedback This is perpetually existing challenge in teams of all tiers (from)

References

[1] Steve Blank. Everything about lean startup in 12 minutes, July 2024. URL https://www.youtube.com/watch?v=G-wwOK4X0lc.

[2] Investor Relations at Meta (Previously Facebook). Press release - facebook to acquire instagram, April 2012. URL https://investor.fb.com/investor-news/press-release-details/2012/Facebook-to-Acquire-Instagram/default.aspx.

[3] Duolingo. Duolingo brand guidelines, December 2024. URL https://design.duolingo.com/. Accessed on 2024-12-04.

[4] swagger.io. Openapi specification version 3.0, December 2024. URL https://swagger.io/specification/.