**UCN psu0219**

**1st Semester Project**

**Group 13**



**DATABASES & TEST REPORT**

**29th  of May 2019**

**Andreas Richardsen**

**Zahro-Madalina Khaji**

Contents

[Database 4](#_Toc9971476)

[Introduction 4](#_Toc9971477)

[Requirements 4](#_Toc9971478)

[Features 5](#_Toc9971479)

[Database Schema Design / Conceptual model design 5](#_Toc9971480)

[ER Diagram 5](#_Toc9971481)

[EER diagram 10](#_Toc9971482)

[Table Design 12](#_Toc9971483)

[Choosing a database type 12](#_Toc9971484)

[Relational model 12](#_Toc9971485)

[Normalization 14](#_Toc9971486)

[DDL statements 14](#_Toc9971487)

[Data generation 14](#_Toc9971488)

[Query Design 15](#_Toc9971489)

[Important Indexes 15](#_Toc9971490)

[Databases & Security 15](#_Toc9971491)

[Applied Database Logic 15](#_Toc9971492)

[Discussion 15](#_Toc9971493)

[Conclusion 15](#_Toc9971494)

[Appendices 15](#_Toc9971495)

[Test 16](#_Toc9971496)

[Introduction 16](#_Toc9971497)

[Description of the GTL case 16](#_Toc9971498)

[Project Initiation Document 16](#_Toc9971499)

[Risk analysis 18](#_Toc9971500)

[Test Plan 19](#_Toc9971501)

[Scope of testing 19](#_Toc9971502)

[Test Objectives 19](#_Toc9971503)

[Test strategy 20](#_Toc9971504)

[Test approach 20](#_Toc9971505)

[Test automation 21](#_Toc9971506)

[Test coverage 21](#_Toc9971507)

[Test environment 21](#_Toc9971508)

[Testing tools 21](#_Toc9971509)

[Test effort and schedule 22](#_Toc9971510)

[Test Management 22](#_Toc9971511)

[Test Control 22](#_Toc9971512)

[Applied Test Techniques 22](#_Toc9971513)

[Designing a testable architecture 22](#_Toc9971514)

[Discussion 22](#_Toc9971515)

[Conclusion 22](#_Toc9971516)

[Appendices 22](#_Toc9971517)

# Database

## Introduction

In this section of the report, we will discuss the Georgia Tech Library (GTL) case while focusing on database design and implementation. We analyze the requirements of the case, plan and document the process of building a suitable database solution. Throughout, we utilize the knowledge and skills gained from the Database Systems for Developers course. Furthermore, we present and explain the various choices and decisions we made. Then, we reflect on the whole experience, and consider any expansions that could be added in the future. In the end, we will conclude with our final thoughts on the case and our implemented database solution.

### Requirements

The GTL case gives us an overview of the library and its employees. The librarians describe their tasks and the ways in which the business rules of the library are enforced. They also mention some of the functionalities that they would like to obtain from the implementation of a software solution.

After carefully reading the case, we identified the following requirements:

1. Library wants to track the number of book copies that are currently on loan and those that are not on loan.
2. Library wants a list of all the books and their descriptions, which can be filtered by book author, title and subject area.
3. Regular library members can checkout books for 21 days, can loan out a maximum of five books at a time and have a one-week grace period for returning borrowed books, before a notice is sent to them.
4. Librarians require a member’s SSN, campus and home address, and phone numbers to register them into the system.
5. Each member gets a numbered library card with their photo on it, that is valid for four years after its issue, a month before the card expires, a notice is sent.
6. Professors can check out books for three months and have a two-week grace period.
7. Library wants a list of books that cannot be lent such as reference books, rare books and maps.
8. Library want a list of books that they are interested in acquiring but cannot acquire.
9. Library uses ISBN to uniquely identify books.
10. Library cooperates with other libraries and is interested in exposing statistics about them.

### Features

From analyzing the requirements, we selected a list of features for our databases solution. However, because of time and budget constraints, we have a limited amount of resources available to us. Therefore, we split these features into essential and non-essential ones.

Our list of essential features contains:

* The book loaning
* A view of all the book copies that are being currently loaned
* A view of all the books that are overdue
* A view of all the books that cannot be loaned according to the library’s business rules
* A view of all the books that match a search criterion such as the book title

Our list of features that we will omit:

* A view of all the books that they are interested in acquiring
* The sending of notifications
* Exposing statistics about other libraries

The scope of this project is now determined and a plan for implementing these features is needed next. We start with choosing the number one priority, which for us is the book loaning feature. This is a core functionality that every library needs to have. GTL also specifies numerous business rules regarding this feature. This highlights its importance but also the complexity of its design and implementation. Therefore, the book loaning is what we will center our solution around. For the rest of the essential features, we prioritize them in order that they are listed above.

Moving forward, we detail the process of designing and implementing our database solution.

## Database Schema Design / Conceptual model design

The conceptual design of the database requires us to have an Entity-Relationship (ER) model and an Enhanced-Entity-Relationship (EER) model. These models are represented using diagrams. We use the ER and EER diagrams to map them into a database schema.

### ER Diagram

The ER high-level conceptual model is used to model the important entities and relationships of our database solution. In this ER model we have the following four entities: Person, Card, Book and Map, and two relationships called Use and To\_Borrow.

#### Entities

The **Person** entity models the users of the system which could be library staff or regular members such as students or professors. The key attribute of this entity is SSN attribute which is unique for every Person. The simple attributes are FName which stands for the Person’s first name , LName which stands for the Person’s last name and Type which stands for the type of user the Person could be. For the type attribute we have decided on the following values: Librarian, Student and Professor. The Person entity has a multiple valued attribute called Phone\_No. In the requirements it was specified that the library would like to store multiple phone numbers associated to a Person. The composite attribute called Address has the sub-attributes Home and Campus. It was also noted that the library would like to store a Person’s home address and campus address.

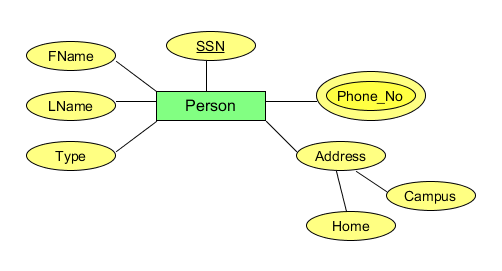


Figure 1: The Person entity in the ER diagram

The **Card** entity is used to model the card issued when a Person registers in the library system. The library assigns each card a unique card number. Card\_No is the key attribute for the entity. We added the simple Card\_Issue\_Date attribute as we want to make sure that the system does not loan books (feature) to instances of the Person entity that have an expired card. Per the library’s stated business rules, a card expires four years after its issue date.

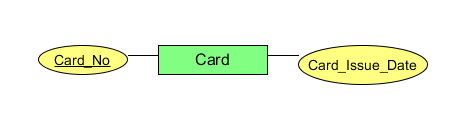


Figure 2: The Card entity in the ER diagram

The **Book** entity is a central entity for our model. The key attribute is the ISBN of a book, which is what the GTL library uses to uniquely identity its books. The simple attributes of the Book entity are Title, Author, Description, Publishing, Year\_Publishing and Type. Except for the Type attribute, all of them are self-explanatory. The Type stands for normal, reference or rare book. The reasoning for choosing these three types will be explained as we continue a bit further down.

Book has two multiple-valued attributes called Subjects and Copies. We assume that each book would fall into one or more subject areas therefore we want to store multiple values for them. The Copies attribute is an interesting one. The library usually has multiple copies of a book at its disposal. For the loaning feature, we realized that we needed to keep track of the individual copies of books that can be loaned. To identify a copy of a book, the ISBN could not be used, so we decided to assign the copies a unique barcode. This allows us to easily keep track of the individual copies of the books.

When it came to model the concept of the book copies, we had three options: to have the book copy as a separate normal entity, to store the copies as multi-valued attribute of the Book or as a weak entity dependent on the Book entity. The reason why we chose the first option is because unlike a typical weak entity the book copy has a barcode that can be used to uniquely identity it. In addition, the book copy is tightly related to a book so it could not stand on its own as an entity and we did not want to facilitate a situation where copies of a book that does not exist are stored in the system. Therefore, we chose to add it as a multiple-valued attribute of the Book entity.

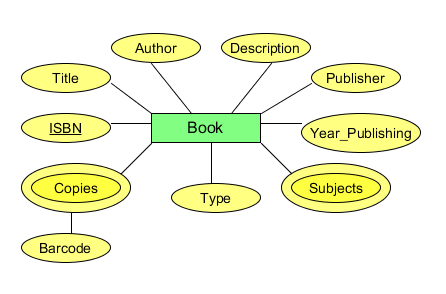


Figure 3: The Book entity in the ER diagram

The **Map** entity is also included, as it is mentioned in the requirements. The key attribute is the ISBN, which is allowed because ISBNs can be assigned to maps as well. [ ISBN reference ] The entity has the following simple attributes Title, Author, Description, Publishing, Year\_Publishing and No\_Copies. No\_Copies is added because we assumed that the library would like to store of the number of copies of each map it has. We are not concerned with tracking the individual copies of the maps in the system because the library does not loan out maps. We assumed that although maps are not loanable, the library would like to store information about them in the system.

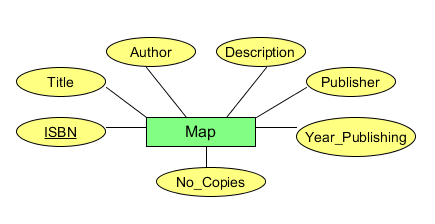


Figure 4: The Map entity in the ER diagram

#### Relationships

In our ER diagram, two important relationships between entities, have been identified. The first one is a binary relationship called **Uses**. This relationship between the Person and Card entity has the following structural constraints: a 1:1 (one-to-one) cardinality and a total participation constraint. This means that exactly one member of Person set, and one member of the Card set are required to take part in the Uses relationship. The total participation constraint implies that both members must be fully involved in the relationship, and that the relationship does not hold if any of them are missing.

The next relationship is called **To\_Borrow**. This is a binary relationship between the Card and Book entity. The To\_Borrow relationship has a N:M (many-to-many) cardinality and a partial participation constraint, which basically means that many cards could borrow multiple books, but they don’t have to. Unlike the Uses relationship, To\_Borrow has two simple attributes associated with it. The first one is called Is\_Returned and it signifies the status of a book borrow, whether the loaned item has been returned or not. The Date\_Borrowed attribute stores the date on which a book borrow has taken place.

We have not added a relationship like the To\_Borrow one, between the Card and Map entities, because in the case, the library states that it does not loan out maps. The flow of the diagram goes as follows: Person uses Card to borrow Book. However, this diagram can be improved by converting it into an Enhanced Entity Relationship (EER) diagram.

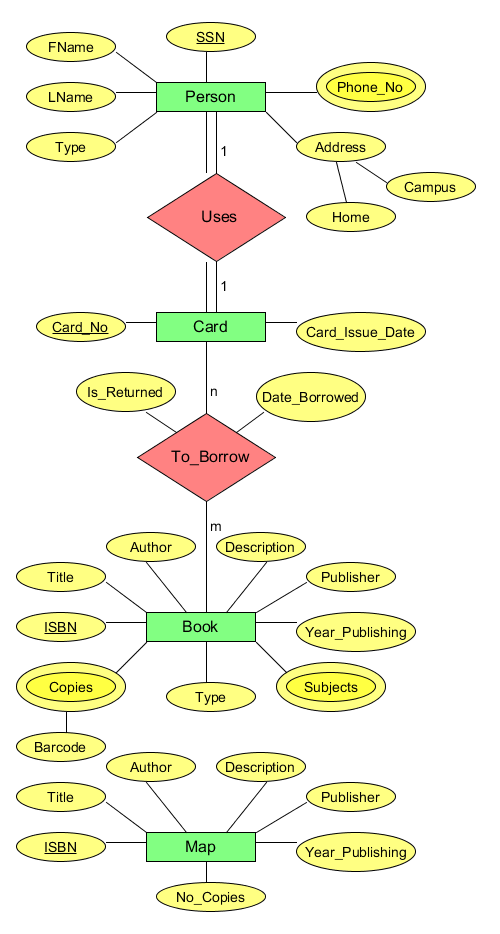


Figure 5: Entity-Relationship (ER) diagram

### EER diagram

The EER diagram, has several additions and differences compared to the ER diagram. We start by describing the changes made to the Person entity. The terms library member and librarian are mentioned in the case. The way we interpreted the case requirements, the librarians are the ones that use the library loaning system to borrow books to members of the library, which can be either students or professors but not both at the same time. The Person entity then becomes a superclass and two new entities are added, Librarian and Member, which are its subclasses. This hierarchy has a disjoint constraint, noted with the letter ‘d’, because as stated before, Librarian and Member are specializations of the Person entity. This superclass entity also has all the attributes that are common to its subclasses, which are the same ones as the ER version of the Person entity, except for the Type attribute which was removed. The Librarian subclass has an attribute called Position, which is used to store the role that the librarian plays in the library such as chief librarian, reference librarian, assistant librarian etc. The Member subclass has a simple attribute called Role which stands for student, professor or other. We decided that it would be necessary to distinguish between library members that are students or professors , because the library has different business rules regarding each one. The type of disjoint is total which means that a member of the Person set must be either a Librarian or a (library) Member in this context. Going forward, we shortly discuss the Uses relationship. We have kept the relationship between the Person and Card entity unchanged. Our reasons for doing so, is because although Person specializes into a Librarian or Member, we assumed that librarians can also use library cards to borrow books. Moving on, we added the Item entity as a superclass for the subclasses Book and Map. The Item holds the following common attributes: ISBN, Title, Author, Description, Publisher and Year\_Publishing, of which ISBN is the key attribute. This hierarchy has a total disjoint constraint, which means that an Item must either be a Book or a Map, but not both. The Map subclass has the simple attribute No\_Copies. The Book subclass has the multiple Subjects, Copies with the simple attribute Barcode, and the simple attribute Type which could be normal, reference and rare. The reason why we have not specialized the Book entity into a NormalBook, RareBook and ReferenceBook is because we could not find any attributes for those possible subclasses. Therefore, we made the design choice to simply store the Type of a book as an attribute on the Book entity. We kept the To\_Borrow relationship the same as in the ER diagram, because the case mentions that only a certain type of book can be borrowed.

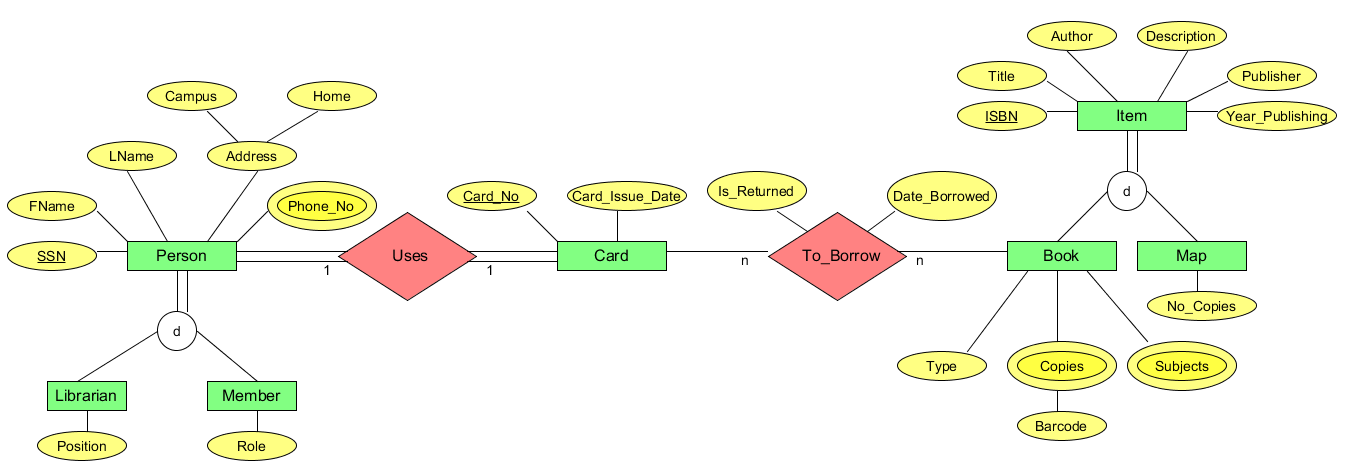


Figure 6: Enhanced-Entity Relationship (EER) diagram

## Table Design

### Choosing a database type

When it came to choose a database type for our database solution, we had three options available:

* Relational database
* NoSQL database
* Hybrid database

We chose a relational database for our database solution for several reasons. First, we considered our experience and skills set as developers, and so we decided that working with a familiar technology stack and SQL, is something we prefer a lot. The learning curve required to mastering a NoSQL database such as the document-based database called MongoDB, would take too much time out of the total time allocated to this project. Secondly, from our analysis of the requirements, we think that the data entities have many relations between them, so in this case the relational database would be the best fit. In the end, we chose to implement a relational database, with MSSQL as the RDMS (Relational Database Management System).

In addition, we strongly consider the option of switching to a hybrid database in the future for this project. We predict that the data in some tables could grow at a high rate, enough to quality for a Big Data solution. Therefore, a relational and a NoSQL database solution, would meet the needs of the project if such a case would manifest in the future.

### Relational model

The next step is to move on to the design of GTL relational model. To accomplish this, we followed the steps of the ER-to-Relational Mapping algorithm and then the steps for mapping EER constructs to relations. Maybe explain the steps?

In total, there are four options for mapping specialization, of which we chose option 8A that states “Multiple relations-Superclass and relations”.[ref to db book] This rule can be applied to any specialization and in our case, resulted in the creation of three relations: Person, Librarian and Member for the first specialization Person and for the second one also three relations: Item, Book and Map. Other options that could have been chosen by us are 8B and 8C.

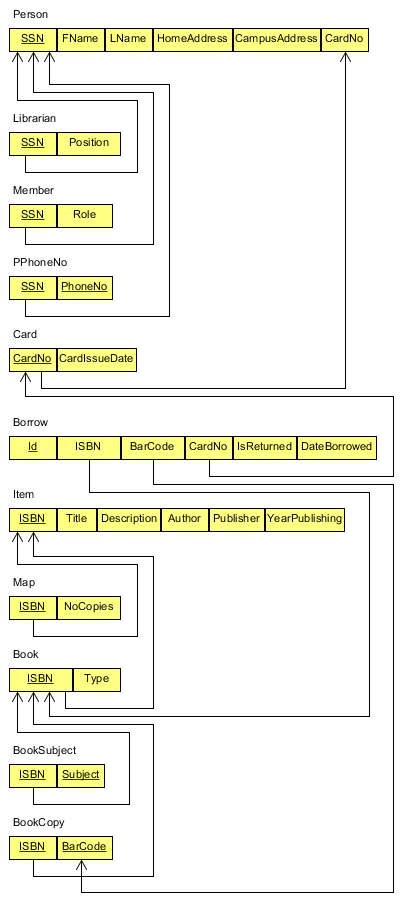


Figure 7: Relational database schema

### Normalization

We studied our relational schema to try to figure out what functional dependencies there are in the tables and in what normal form they each are.

### DDL statements

These are the DDL statements that we used to create our tables and relations with SQL in MSSQL.

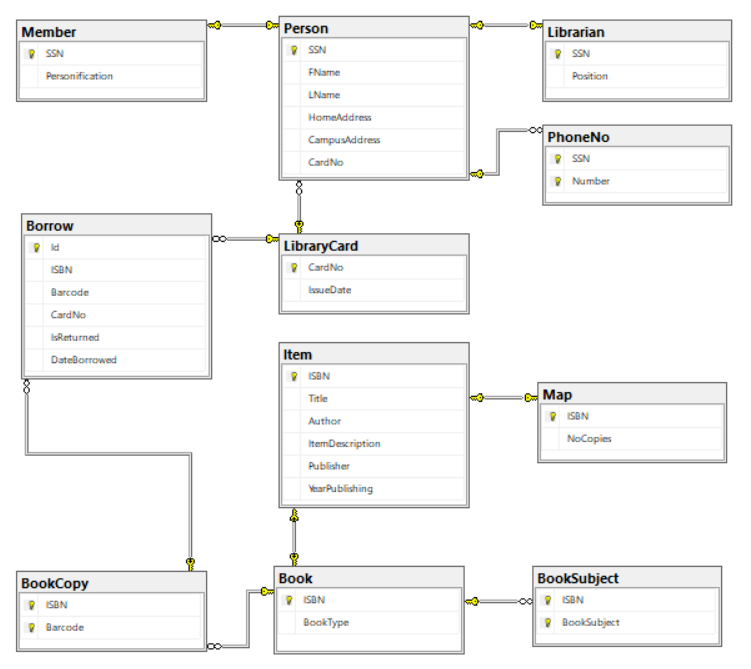


Figure n: GTL Relational Database Diagram

### Data generation

Data generation inserted not normal books into the Borrow because we had not set a constraint, the same copy of the same book has been loaned by two different persons on the same day. We did not think to add constraints to the Data generation require us to.

## Query Design

## Important Indexes

## Databases & Security

Only librarians can use the system. In a completed system, librarians would have to login and depending on their position they would have different privileges e.g. Chief can do everything, reference librarians can only look up books etc.

## Applied Database Logic

## Discussion

## Conclusion

## Appendices

# Test

## Introduction

In this report, we will introduce and discuss the Georgia Tech Library (GTL) case and we will present our proposed solution for the case with a strong focus on the software testing aspect of it. In continuation, we will describe the development process that was followed by us, how testing fit in with it and the various testing techniques that were used. We plan to go through the evolution of the project and how we managed to adapt the testing process to fit our changing environment. We explain our motivations and reasons for why we chose to test certain parts of the system and how we decided which parts were critical/important enough to need focused or additional testing.

### Description of the GTL case

The main point of the GTL case that we were tasked with, is that the library needs a book loaning system and that we will build it for them. There are several business rules that were mentioned in the case, that must be implemented and accounted for. As well as that, the case gives us an overview of the library and how it functions. At the end of the case, we are assigned with building a working prototype for managing the library and its areas of concern. The case does not specify whether the library already has a software system like the one they are requesting us to build except for the mention of an online catalog of books which is quite vague. Therefore, we assumed that we would need to build a completely new system for them.

### Project Initiation Document

For the sake of this project, we constructed a Project Initiation Document (PID). The purpose of a PID is to clearly define what the library, which is the customer in this case, should expect to receive from us, the suppliers of the system, at the end of the project. The document is included below.

#### Business objective

* To modernize the library’s loaning system by switching to a digital solution

#### Project objective

* To implement a working prototype for a library loaning system that will keep track of the members, books and borrowing activity

#### Scope

* Boundaries
  + Member registration
  + Book management
  + Tracking of items status such as available or borrowed / on loan
  + Reliable item loaning functionality
  + Sending notifications according to member type
  + Enforcing the communicated business rules such as
    - Restricting the borrowing of certain items
    - Restricting the number of books, a member can loan
* Activities
  + Supplier will only provide the software of the discussed solution; additional software that may be required as a platform for the solution to run on is not provided;
  + Supplier will not provide any hardware components; that is considered the customers responsibility
  + Supplier will not be responsible for any security issues of the system
  + Supplier will not be taking on the task of maintaining the system, after the completion of the project
  + Supplier will not be tasked with training the library’s employees on how to use the system
* Deliverables
  + Software system consisting of a working prototype with the functionalities that were agreed upon by both parties
  + Documentation about the systems design, implementation, usability and other relevant features

#### Constraints

* The development lifecycle that will be used is the spiral model
* The development approach is an Agile one, based on SCRUM
* The proposed software system will run on the Windows 10 operating system
* The technology stack used is the .NET framework with the C# programming language

#### Authority

* The person responsible for representing the customers interests and tasked with approving the final product is one of the developers

#### Resources

* The projects cost is fixed
* The staff working on developing the system consists of two developers
* Time is two months to complete
* The hardware required to develop the system is provided by the supplier

Besides the specifics of the product that is to be delivered, the PID also describes the project regarding important constraints such as project scope, delivery time, the desired quality and the cost of the project. These factors were taken into consideration by us, when we chose our test strategy and test approach.

## Risk analysis

Before beginning any project, it is critical to conduct a risk analysis to be able to handle any possible issues that might force the project to a halt. To be thorough, we decided to look at both the project and the product risks in this analysis.

Project risks

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| No. | Risk | Description | Impact | Priority | Mitigation |
| 1 | Lack of testing experience | Team doesn’t have enough concrete experience with the process of thoroughly testing a software system | High | 1 | Team must rely on following best practices and standards instead of experience |
| 2 | Lack of testing tools knowledge | Team is unacquainted with the testing tools necessary to properly test the system | High | 3 | Team must allocate time resources to reduce knowledge gaps |
| 3 | Product owners not available | The actual product owners cannot be involved in the process which is an important factor in agile development | High | 2 | Team must assign one of its members to act as a product owner for the duration of the project |

Product risks

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| No. | Risk | Description | Impact | Priority | Mitigation |
| 1 | Vague or incomplete requirements | The case, from which the requirements are to be extracted, is not detailed enough | High | 1 | Team must make assumptions when necessary and try to guess what the library would want |
| 2 | Lack of | Team is unacquainted with the testing tools necessary to properly test the system | High | 3 | Team must allocate time resources to reduce knowledge gaps |
| 3 | Lack of business case | No business case has been provided which makes it hard to figure out what the main objectives that the system should solve are | Medium | 2 | Team must resign to select some possible objectives from the description of the case |

## Test Plan

As stated in the GTL PID, we follow an agile development process along with the SCRUM agile method. The purpose of a test plan is to plan testing activities in advance, to increase the likelihood that they will take place when they should or when we decide to execute certain tests. Like many other types of documentation, when it comes to agile, test plans are a controversial topic. We think that there are valid arguments both for using and for not a test plan. We have chosen to write a test plan for our agile project, mainly for the reason mentioned earlier, regarding their use. Secondly, we think that we can make a test plan work in our case, if we add a few adjustments to it. First, we need to change our mindset and view this document as more of a general guideline for our project tests instead a rigid plan. Secondly, we decided that the plan should only include elements that are helpful in our case. Having said that, the contents of our custom test plan are as follows:

1. Scope of testing
2. Test objectives
3. Test strategy
4. Test approach
5. Test automation
6. Test coverage
7. Test environment
8. Test tools
9. Test effort and schedule

### Scope of testing

The project we will be working on is a software development project which will implement a library loaning system meant to replace GTL’s already existing analog loaning system. The system will be used internally, and our testing will be focused on the Business Layer. The main part is focused on the loaning of books and films to its members. The project objectives we are concerned with are keeping track of the loans, the members and the due dates.

### Test objectives

These are our test objectives:

1. Verify that a loan is registered in the system
2. Verify that system sends notification on due date depending on the member type
3. Verify that system does not allow a member to borrow more than a maximum of five books simultaneously
4. Verify that a loan’s status is updated when member returns the book
5. Verify that system does not loan out books that can’t be borrowed
6. Verify that system does not allow members that own cards older than four years to borrow books

### Test strategy

The test strategy is an important key in the success of any project. For our project, we decided that a risk-based analytical strategy would be best suited. Some of the factors that we considered when we picked our strategy are: risk, skills, objectives and product. Managing both project and product risks has been a major concern for us since the beginning. We conducted a detailed risk analysis, where we weigh in on all the risks that we thought we might face. In addition, we also devised ways to mitigate those risks. A risk-based strategy, for us, means that we use testing as a tool to manage the risks. For example, when it comes to the risk caused by a lack of time resources. In this case, we should strive to choose types of tests that allows us to deliver the most value in the small time slot that we have.

Skills, especially testing skills, have also been an obstacle for us during this process. It is necessary to consider the skillset within our team, when deciding on a strategy, to make sure that we can properly implement it in the end. If we take into consideration our limited testing skills, then a strategy such as a standard-compliant one, could have also been a good option for our team.

Regarding the objectives, we mostly took into consideration our testing objectives. These objectives act as a guide for identifying the most important user stories that we have and ensuring that those end up being properly tested.

Our product, which is a small library book loaning system has been a huge factor for deciding our strategy. The product that we deliver to the customer is dependent on the scope of the project that we set out to implement. The project scope has been clearly laid out in the PID document. Besides, the actual features that we need to build, it helps to know the type of the product as well. Because, we know that a prototype for a library system does not have as high of a criticality as a hospital management system for example. This tells us that we don’t need a methodical strategy, which implies that we follow an industry-standard for testing.

We continue by giving a definition of the test process that we intend to follow. First, we create test cases based on the acceptance criteria noted on our user stories. In the case of the most critical ones, we might require additional details. Then, we consider the levels at which testing will occur and our levels of automation. Another, step in this process is define our exit criteria or when we are done testing.

In the end, our strategy requires us to frequently use our analytical skills to prevent and manage various risks and to ensure that adequate testing is being carried out.

### Test approach

The test approach is described as the actual implementation of a test strategy. Test strategies can be reused across multiple projects, while testing approaches must be tailored to each individual project. At this point we need to introduce the Agile Testing Quadrants. [ref to agile testing book] This is a tool used to help developers cover all the types of tests could be considered useful in the context of their project. The tests listed in each quadrant have a different purpose. For example, the types of tests in Q1 are unit tests and integration tests. This is where we will focus the vast majority of our testing resources and efforts. Continue to Q2.

Elements of a good test strategy in agile – automation is important – at what levels should we automate? What about exploratory testing? Non-functional requirements?

Help define how the testing process looks like in agile.

Automation strategy is important because of regression testing

The team is encouraged to do their own planning and test estimation instead of the test manager who used to write a large test plan

Unit tests

Integration tests

System tests

Non-functional tests NO because prototype

Acceptance test cannot be accomplished given time and cost restraints.

### Test automation

### Test coverage

Metrics can be very helpful when testing, to make sure that the team stays on track. For us, the metric that we decided to track, throughout our development process, is test coverage. The standard, that we set out to achieve or maintain, is to have at least 80% of the code covered by tests, at all time. Using TDD helped us tremendously when it came to sticking to our test coverage goals.

We used the in-build Visual Studio tool for measuring the test coverage. Although, we did not log our numbers in a formal document, we made sure to check on our test coverage numbers a few times a day, while working on the code. As a team, we found it was very motivating to ensure that our test coverage met the standard. On the same hand, it also drove us to write more tests, because we were thinking of ways to raise our numbers even more.

Our latest numbers displaying our test coverage, are shown below.

[figure showing test coverage]

### Test environment

When it comes to a testing environment, the aim is to have it as similar to the production environment as possible. In an ideal scenario, we would use a Virtual Machine or Docker to ensure a consistent and proper testing environment for our project. Also, we would utilize a remote database to have the testing process be as similar to production mode as possible. However, because of time and cost restrictions, we could not opt for those options. Instead, we used the technology that was already available to us, which includes one laptop running on a Windows 10 operating system and a local MSSSQL database.

### Testing tools

We use the following testing tools: NUnit for .NET framework for writing and automating unit tests.

### Test effort and schedule

The planned test effort and schedule is shown below:

|  |  |  |
| --- | --- | --- |
| **Task** | **Members** | **Estimate Effort** |
| Create the test specification | Developers/Testers | 1 week |
| Perform test execution | Developers/Testers | 2 weeks |
| Test report | Developers/Testers | 3 weeks |
| Total | Developers/Testers | 6 weeks |

## Test Management

Managing tests within an agile project is both an interesting and complex topic. The most challenging part of this whole undertaking for us was certainly trying to figure out how to reconcile test management with agile. It is a well-known myth that agile development does not require any documentation. Documenting one’s work in a lightweight manner, is encouraged in agile. We agree that having minimalistic documentation that is useful for the team and makes sense in the context of the project, is an overall positive. The issue we faced with test management is that the idea of carefully planning out our tests in advance still seems to go against the core of the agile mindset.

So instead of focusing on providing rigid and prescriptive rules for testing and planning out everything, we will focus on writing a guideline that are supportive example we will define what done means in the context of our project.

Agile testing is very much influenced by the skills of the team at the current moment.

Test plan contains

## Test Control

## Applied Test Techniques

Unit test – we follow the naming convention Unit under test, scenario, expected outcome[shown in picture x].

## Designing a testable architecture

Dependency injection

Inversion of control

Simple factory

Abstract factory

## Discussion

## Conclusion

## Appendices