Heritage’s IAAA (HIA3)

Pierre Badra

Ryan Somers

Joshua Magnan

Sebastian Canales Burke

Sabrina Tochkov

Table of Contents

[Introduction 2](#_Toc1561001808)

[Course Competencies 2](#_Toc231230670)

[Development Project I Feedback 3](#_Toc1106165253)

[Agile Sprint Process 4](#_Toc1755252214)

[Budget and Estimates 4](#_Toc2145764971)

[Design 4](#_Toc457913450)

[Refactoring 4](#_Toc1073566541)

[DevOps 4](#_Toc667650893)

[Black Box Testing 4](#_Toc1304679949)

[White Box Testing 4](#_Toc1312705595)

[Conclusion 4](#_Toc269586862)

# Introduction

The purpose of this report is to reflect on the Development Project I course (420-K40) of the Fall 2024 term, the progress we’ve made as the team responsible for HIA3 and the lessons we’ve learned regarding the software development life cycle. In this document, we will go over the course competencies and outcomes and outline our process for the first half of this large-scale project, including the Agile Sprint process, testing process and so on. We estimate that this report will likely take about 8 hours of effort in total, which we’ve evenly split up between team members. Preparing the presentation will most likely take another 2 hours, so 10 hours in total for this final assignment.

# Course Competencies

The Development Project I course addresses two key competencies in the Computer Science program:

* Develop transactional Web applications

To address this competency, we’ve had to build a database with SQL Server, prepare our programming environment using an Azure DevOps Git repository and develop both server-side and client-side application logic using C# and the ASP.NET Core framework. Our application requires a server-side which handles API requests for authentication and authorization, and a client-side GUI which will be used by the client to manage apps, users and roles. Thanks to the use of Git and the Azure DevOps process, we were also encouraged to conduct regular code reviews for pull requests, organize user stories and tasks and keep up with software project development procedures.

* Collaborate on the design of applications

To address this competency, the course spends a lot of time encouraging team-to-client interactions through frequent meetings. We’re taught to analyze and break down the requirements of a somewhat non-technical client, put together a coherent design and keep organized and clear documentation along the way. We’ve created an inception deck, a software design document, and a database diagram (which we’ve had to adjust a few times to adapt to evolving client requirements). The Development Project I course allowed us to participate in the overall design of the application while learning the development standards and best practices in the workplace.

# Development Project I Feedback

According to our team, the course is organized to meet these needs quite well. The flow of assignments made sense, although we feel that the “0th” assignment was not necessary. It was originally meant to act as a sort of refresher assignment that doesn’t count for any grades, but we unanimously agreed that it proved to be more of a waste of time, time which could have been spent on the design phase of the actual project. Other than that, the flow of assignments was pretty self-explanatory and reminiscent of the HVK project.

We feel that we were well prepared to use the software technologies required in this course (C#, .NET, EF, SQL Server, MVC). Although it took some time to wrap our minds around having to design an API (since we were simultaneously learning about them for the first time in the Web Programming VI course), it all ended up working out in the end. We feel confident that we can move forward in this project next semester with little to no roadblocks.

# Agile Sprint Process

As for the agile sprint process, we attempted to replicate it as much as possible given our schedule restricting us in having daily scrum meetings. At the start of each class before development, a “daily scrum” would be held in which we would take turns in answering questions regarding what we had completed prior, what is currently being done, what is yet to be done, and if there are any roadblocks that need to be addressed and are hindering progress. Someone (usually the APM) would put these meetings into notes, so that they can be later referenced. This cycle would last approximately two to three weeks, ending with a meeting with the client in which we would showcase the product increment and receive feedback. Once a sprint iteration ended, we would have a team sprint retrospective meeting to look back and learn from mistakes and apply the experience into future sprints.

Here’s what went well and what needs to be improved in each of the following sprint aspects:

|  |  |  |
| --- | --- | --- |
| **Aspect** | **What went well?** | **What can be improved?** |
| Sprint Review & Sprint Planning Meetings | - Estimates were fairly accurate | - A small improvement would be for the sprint planning meetings; they could have been more structured with a fixed time length. |
| Daily Scrums | - Everyone was ready to answer the questions when asked (Prepared before the meeting) | -Take the time to look at the scrum notes taken from previous daily scrum meetings when needed (They were unused for the most part) |
| Sprints | -Executed the process well | -We could have been more diligent with our code reviews for git merging. |
| Lengths of Sprints | -Fairly consistent, which allowed us to have more and more accurate estimates, and more realistic sprint commits. | -Each sprint could have lasted longer without Assignment 0. |
| Retrospective Meetings | - Everyone was ready to answer the questions when asked (Thought before about what they were going to say) | -A small improvement would be for the sprint planning meetings; they could have been more structured with a fixed time length. |
| Role of APM | -The person responsible for being the APM always took the initiative and was really good at it. | -Deciding who to choose as our APM (we could have all been APM. This made us waste a little bit of time in deciding who to make APM, and unsure at times) |

Based on our experience working on this project, some of the benefits of working with agile are the following:

* Flexibility and Adaptability:

Agile can allow teams to respond and adapt to changes quickly. This was especially useful when we changed our database schema many times throughout development. It gave us the freedom to quickly adjust and briefly shift our focus to something that did not directly contribute to the fixed end-goal but was important to do for maintainability purposes and to limit technical debt.

* Improved Collaboration and Communication:

Through “daily” scrum meetings, sprint reviews, and retrospectives, open communication and collaboration among team members and stakeholders made it so that everyone knew exactly what each person was doing throughout the entirety of development and whether they needed assistance in the event of a roadblock. This in turn improved teamwork and collaboration.

Based on our experience working with this project, some of the drawbacks of working with agile are the following:

* Potential for Scope Creep:

The flexibility of Agile can sometimes lead to uncontrolled scope expansion if changes are not properly managed or prioritized. This happened to us when our client, luckily in the middle of the design phase, added the requirement that the system should be able to accommodate legacy applications from the college by implementing additional SOAP web services. This unexpected addition increased our project scope, which was only possible in the agile process.

* Requires High Customer Involvement:

Agile depends on active stakeholder participation. Lack of engagement or availability can hinder progress and lead to misaligned expectations. Often during each SDLC phase, the more we would think about the project requirements, the more we would come across unanswered questions that we had not thought about during meetings with the client. This required us to make decisions for the client that may have not aligned with their vision of the project’s direction. The agile process made it so that we always needed to consult the client at any time, which is very unlikely given their schedule.

According to our team, there are no recommendations for change that we could use to enhance our team process, as we employed the agile sprint development methodology very well.

For our team, many parts of the project were helpful in our learning. This includes task estimation and assignment, git collaboration, the general usage of a team collaboration tool such as Azure DevOps. These parts of the project were most helpful in our learning to become working professionals.

The work was distributed equally in terms of each entity’s CRUD operation. This was to reduce the number of merge conflicts that we would encounter as each person would work on a different feature of the system.

The group dynamics remained the same throughout the semester. We were all very supportive of each other and we were always willing to help each other. There is nothing that needs to be improved in this aspect of the project. The team was great.

# Budget and Estimates

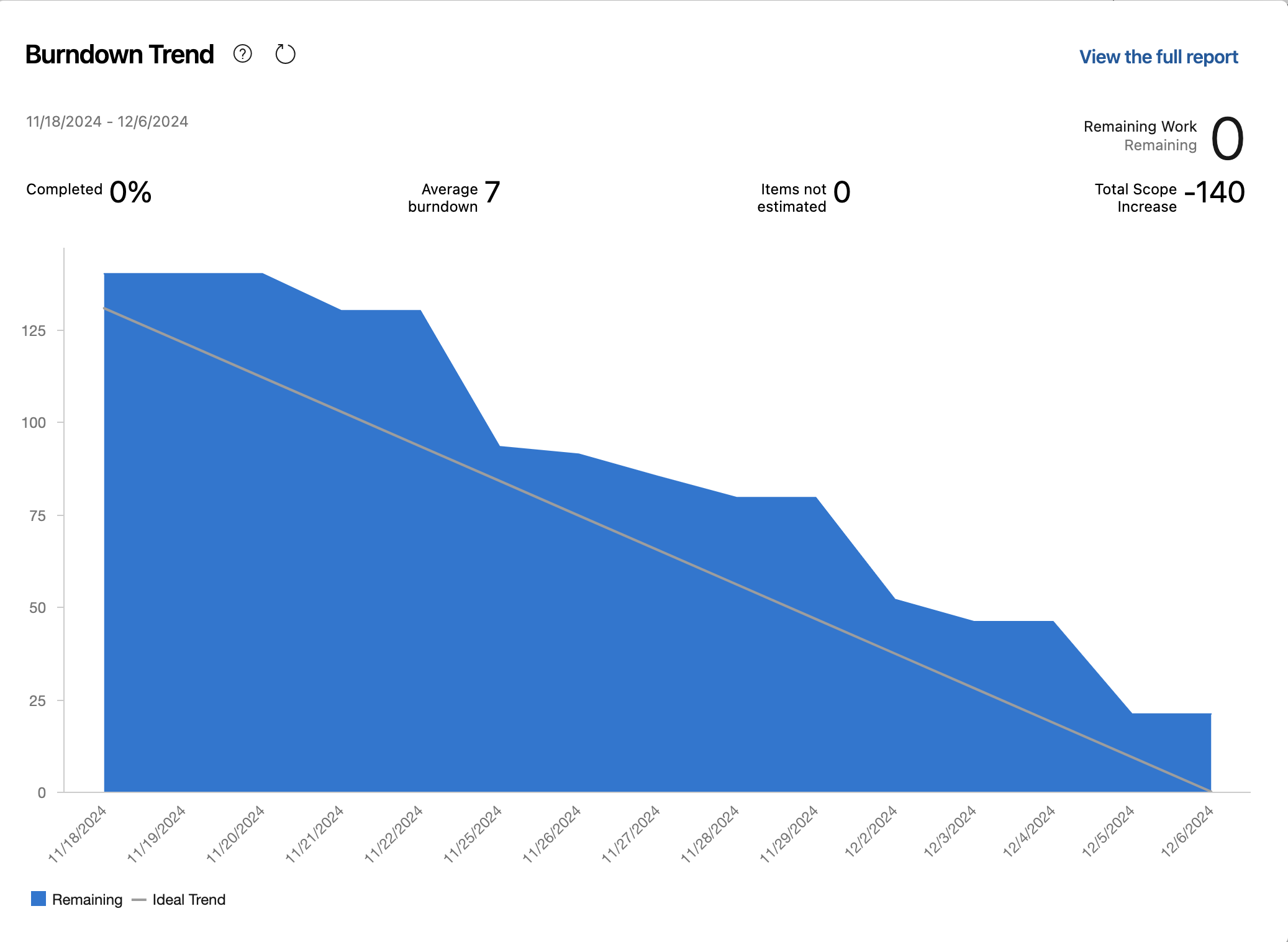
Budget and plan/Actual

For the sprint, the team had 165 hours budgeted. We planned on using 140 hours (85%), which was approximately split evenly between each person. Each person was assigned approximately 28 hours.

Analyze

The originally budgeted hours were insufficient, as we had planned to complete most of the functionality in this sprint, giving us more time for refactoring and maintenance in the upcoming sprints. This resulted in us having to move the remainder of our assigned tasks into the next sprint. In hindsight, we could have committed to completing less features in the system and be more realistic about what can be completed in the given time frame.

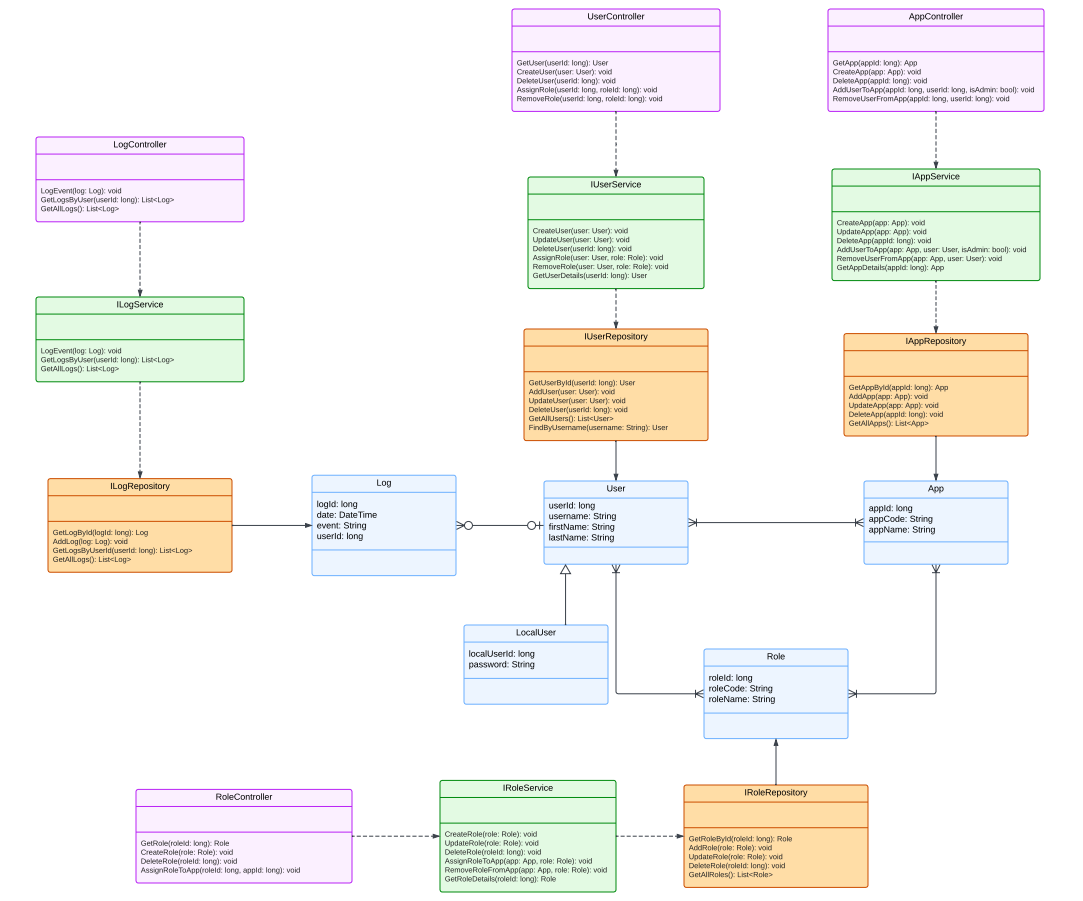
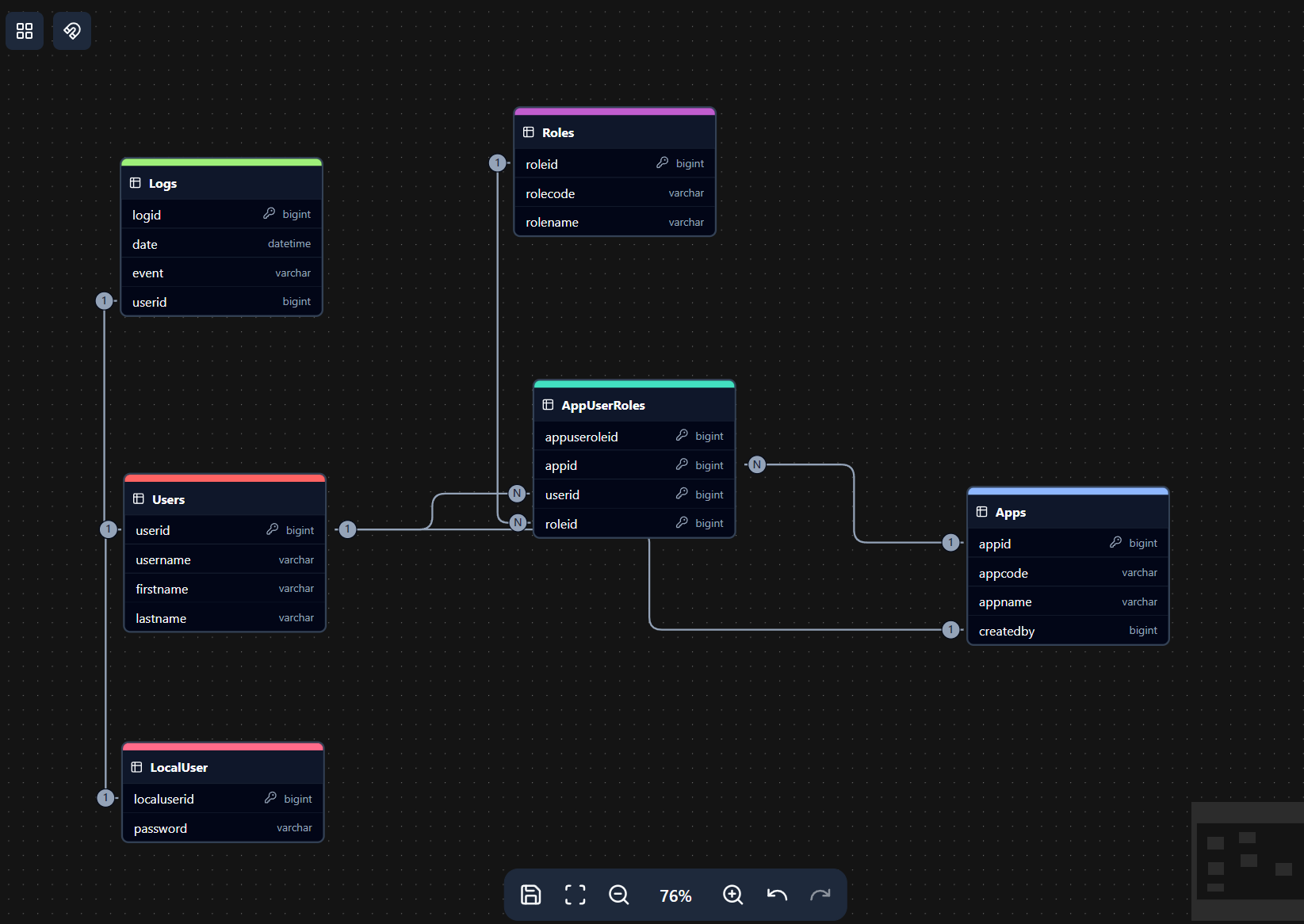
A sprint burndown chart is a visual tool used in agile project management, particularly within frameworks like Scrum, to track the progress of a sprint. It displays the amount of work remaining over time, and helps teams monitor their pace and determine if they are on track to complete their sprint goals.

This is our sprint burndown chart:  


The burndown chart above shows steady progress, with us completing most of the work by the end. Although the chart indicates steady progress, it remains slightly above the ideal burndown trend, suggesting overcommitment during planning.

# Design

Plan



Actual

The difference between the planned design and what has been implemented thus far is the entire log functionality. This was not a feature that we intended on committing to in this sprint. Another minor difference between the planned design and our currently implemented design is the database schema. Specifically, some tables have new columns added to them.

Analyze

The original system design was mostly correct except for some small adjustments as mentioned above. In hindsight, a potential change we could have made to our process in conceptualizing the system architecture would be making additional research before making a permanent decision. For instance, we could have further analyzed the implications of having one associative entity linking three different entities, rather than separating it into smaller associative entities. At first, it seemed like a good idea, but as development went on, it became an increasingly challenge to manage considering all the use cases.

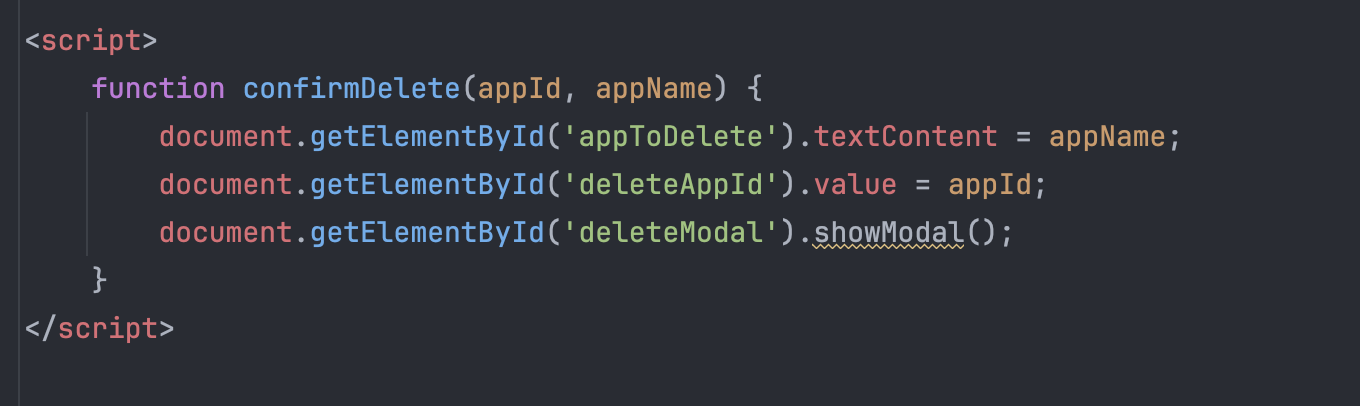
# Refactoring

Code refactoring involves restructuring existing code without altering its external behavior, aiming to enhance its internal structure. This practice improves code readability, maintainability, and performance. This makes it easier to understand, modify and extend.

**Some benefits of Code Refactoring:**

* Improved Code Quality:
  + Refactoring enhances the design of code to make it cleaner and more efficient.
* Easier Maintenance:
  + Well-structured code simplifies debugging and reduces the likelihood of errors. Making it easier for updates and modifications.
* Enhanced Performance:
  + Optimizing code through refactoring can lead to better performance and resource utilization.

**Two Improvements for the HIAAA project:**

* Move Scripts to a Dedicated JS File
  + Keeping JavaScript code in separate files ensures cleaner and more modular code.
  + If we stored our scripts in a separate file instead of inline in the views, we could then re-use the code across multiple views.
  + If we needed to update or debug the script, it would be easier to locate and modify it in a single JavaScript file rather than searching through multiple .cshtml files. HIAAA/Views/App/Index.cshtml
* Shared DTOs
  + Shared DTOs ensure that the data structure is consistent between the backend and frontend.
  + Instead of redefining DTOs in multiple places, we can define them once and reuse them across projects.
  + This can also be better to maintain. You only must modify the DTO that is in one place and not in multiple locations
  + For example, the DTOs in **HIAAA/DTO/AddAppDTO.cs** and **HIAAAServices/DTO/AddAppDto.cs** are the same but are used in two different projects.

# DevOps

**Traditional Approaches:**

* Building:
  + Code is written and compiled, often with minimal integration with other parts of the system. Builds are done manually or at scheduled intervals.
* Testing:
  + Testing happens in a dedicated phase after development is completed. Bugs are identified late in the process, often requiring significant rework.
* Deployment:
  + Deployment is a separate manual process handled by an operations team. This created bottlenecks and potential for human error.

**DevOps Approach:**

* Building:
  + Continuous Integration (CI) automates code builds every time new changes are pushed to a repository. Developers get immediate feedback on their code.
* Testing:
  + Continuous Testing ensures automated testing (unit, integration and performance) happens alongside the build process, catching issues earlier.
* Deployment:
  + Continuous Deployment (CD) automates deployment pipelines. This enables frequent and consistent delivery of updates to production or staging environments.

**Advantages and Disadvantages of the DevOps Approach:**

Advantages:

* Faster Delivery: Automated CI/CD pipelines accelerate build, test, and deployment cycles.
* Higher Quality: Automated testing and monitoring catch issues early on thus reducing defects in production.
* Enhanced visibility: Azure DevOps and other tools provide tons of valuable data and transparency in the development pipeline.

Disadvantages:

* Learning Curve: Requires team members to learn new tools and practices.
* Upfront Costs: initial setup of CI/CD pipelines can be time-consuming and resource intensive.

**Recommendation for the HIAAA project:**

* Continuous Integration (CI):
  + Use Azure DevOps Pipelines to automate the build process
  + Run automated tests (unit and integration) on each pull request to catch errors early.
* Continuous Deployment (CD):
  + Set up automated deployments to staging environments.
  + Automate production deployments with approval gates to ensure manual checks when needed.
* Automated Testing:
  + Integrate xUnit for test cases.
  + Add these tests as steps in our CI pipeline to ensure code quality.

**Note**: Many of these features are in Azure DevOps but it will be a question of whether we can access and use them in this upcoming semester (Development Project II).

# Black Box Testing

**Overview:**

We conducted black box testing during our sprint through User Story acceptance validation. Each item that was created was tested using ATTD. While this ensured individual feature functionality, it didn’t fully validate the system’s integrated behavior from an end-user perspective.

**Functional Testing:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Test ID** | **Purpose** | **Expected Result** | **Actual Result** |
| FN01 | View all roles in roles page | Display list of all roles with correct permissions | Failed (Defect #1) |
| FN02 | Create new App-Admin account | App-Admin successfully created and appears in system | Passed |
| FN03 | Assign App-Admin to their application | App-Admin gains access to designated app | Failed (Defect #2) |

**Defect Table:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ID** | **Description** | **Severity** | **Priority** | **Steps to Reproduce** |
| 1 | Roles page fails to load role list correctly | 3 | 2 | 1. Navigate to roles page 2. Observe missing role data |
| 2 | System creates duplicate App-Admin entry when assigning to app | 3 | 2 | 1. Create App-Admin 2. Assign to app 3. Observe duplicate entry |
|  |  |  |  |  |

**Usability Tests:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Test ID** | **Purpose** | **Expected Result** | **Actual Result** |
| US01 | Admin dashboard accessibility and navigation | All admin functions accessible within 3 clicks | Failed (Defect #3) |
| US02 | Admin user management interface consistency | User management forms maintain consistent layout and validation | Passed |
| US03 | Admin audit log readability | Audit logs display in clear, filterable format | Failed (Defect #4) |

**Defects Table:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ID** | **Description** | **Severity** | **Priority** | **Steps to Reproduce** |
| 3 | Some admin functions not properly completed so cannot fully test. | 3 | 2 | Attempt to go to some not completed pages. |
| 4 | Audit log not implemented | 3 | 2 | Click on the Monitor page. |
|  |  |  |  |  |

**Exploratory Tests:**  
Conducted exploratory testing focusing on:

* Edge cases in role-based access control
* Concurrent user operations

**Conclusion:**

Testing revealed 4 defects:

* 2 functional defects (1 high severity, 1 medium)
* 2 usability defects (1 medium, 1 low)

Overall system quality is moderate, with critical role-based access control working but requiring real-time updates and UI consistency improvements.

# White Box Testing

**Purpose of Unit Testing:**

Unit testing involves testing individual components or units of code in isolation to ensure they work as expected. The goal is to catch bugs early in the development process and validate the correctness of specific functions or methods.

**Acceptance Test-Driven Development (ATDD):**

* ATDD involves collaboration between developers, testers and stakeholders of the project to define acceptance criteria for features before development begins.

**Test-Driven Development (TDD):**

TDD is a development process where tests are written before the code. Developers aim to write just enough code to pass the tests.

**Pros vs. Cons of Unit Testing and TDD:**

Pros:

* Early Bug Detection: Bugs are caught early in development. This reduces time and effort.
* Improved Code Quality: Writing tests encourage modular and maintainable code.
* Documentation: Tests serve as documentation for expected behaviour.

Cons:

* Time-Consuming: Writing and maintaining tests can slow down development initially.
* Overhead: Poorly written or redundant tests can add unnecessary complexity.
* Learning Curve: TDD requires developers to learn a disciplined approach, which may be challenging for teams new to testing.

Compared to Not Testing:

* No Testing: Faster development initially, but there is a higher risk of bugs or fragile codebases. This can also make debugging expensive.
* Testing/TDD: Slower initial progress but results in a stable and maintainable codebase over time.

**Initial Timing for Tests:**

* When to run Tests:
  + After each build via a CI pipeline.
  + Before merging pull requests.
  + Before releasing to staging or production.
* Testing in Sprints:
  + Ideal Timing: Write tests alongside development (TDD/ATDD) and run tests continuously.
  + Regression Tests: Run all existing tests in every sprint to ensure new changes don’t break previous functionality.

**Code Coverage:**

Definition:

Code Coverage measures the percentage of code executed during automated tests. It indicates how thoroughly the codebase is tested.

Ideal vs. Realistic Code Coverage:

* Ideal: 100% coverage ensures all code paths are tested.

Realistic and Sufficient: Achieving 100% is often impractical. Aiming for 80% coverage is good and try to focus on the critical paths and high-risk areas of the application.

**Assessment of Current Project Testing:**

Current State:

* # of Tests: 0
* # Pass: N/A
* # Fail: N/A
* % Code Coverage: 0

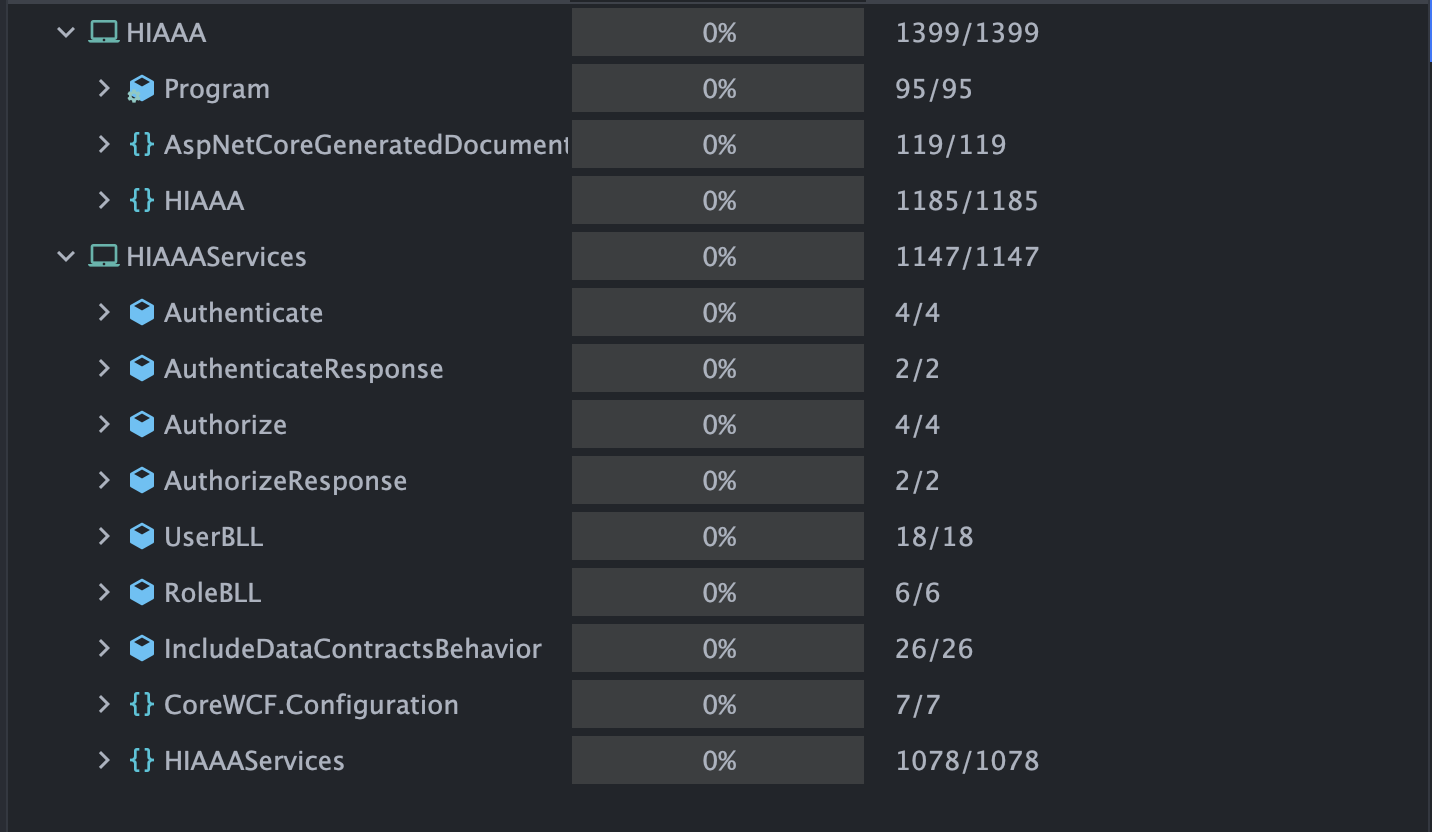
Assessment:

* Status: Needs improvement. There’s no way to verify code functionality. This could lead our system to be more error prone.

Plan for Next Semester:

* Allocate a full month for testing to ensure a robust application.
* Use tools like xUnit (unit testing framework) to help with testing.
* Implement TDD for new features and write regression tests for existing code.

HIAAA’s code coverage results: 0%.



# Conclusion

To summarize, the benefits of this course are that we as 3rd year students get to experience the full picture of the software development life cycle with no red tape or restrictions in our way. It’s like a sandbox environment, to let us experience every facet of a software project, since in the workplace we’d probably only get a glimpse of it. We are given the opportunity to participate, make mistakes, learn from them and improve as developers. It also allows us to better gauge what we like in the vast world of IT and hone our interests when searching for a job.

We appreciate that we are given the freedom to design an application as we see fit, but we’d say that a little more guidance and advice from the project manager would be incredibly beneficial. In the workplace, a team would be able to consult experts for different parts of the project (deployment, database design, security, etc.) If we are doing something the wrong way, it’s crucial to tackle that issue and fix it at the core as soon as possible instead of waiting for the assignment feedback, when that part of the project feels far behind, out of sight and out of mind. Despite its faults, this is a great course and appropriately challenging and we would definitely recommend it to next year’s class.

After completing this report and presentation, we found that our original estimate was accurate, give or take an hour. It has given us the time to gather our thoughts and reflect on our progress so far, which we are quite proud of.