Bern University
of Applied Sciences

URL-Archiver

Project report

Course of study	Project 1
Author	Nicolin Dora, Abidin Vejseli & Kilian Wampfler
Advisor	Dr. Simon Kramer
Co-advisor	Frank Helbling

Version 0.1 of January 2, 2024

- Technik und Informatik
- Informatik

Abstract

One-paragraph summary of the entire study – typically no more than 250 words in length (and in many cases it is well shorter than that), the Abstract provides an overview of the study.

Contents

Abstract ii

List of Tables v

List of Figures vi

Listings viii

1. Introduction 1
   1.1. Initial Situation . . . 1
   1.2. Poduct Goal . . . 2
   1.3. Priorities . . . 2

2. Specification 4
   2.1. System Delimitation . . . 4
      2.1.1. System Environment (statics) . . . 4
      2.1.2. Process Environment (dynamics) . . . 6
      2.1.3. Operational Processes . . . 6
   2.2. Requirements . . . 8
      2.2.1. Epics and User Stories . . . 8
      2.2.2. Functional requirements (added value) . . . 13
      2.2.3. Boundary and Pre-Conditions . . . 13
   2.3. Usability . . . 13
      2.3.1. Personas . . . 13
      2.3.2. Storyboard . . . 13
      2.3.3. UX-Prototyping . . . 13

3. Implementation 14
   3.1. Architecture (e.g., back-/frontend) . . . 14
   3.2. Processes . . . 14
   3.3. Allocation of roles . . . 14
   3.4. Scrum roles . . . 14
   3.5. Additional roles . . . 15
   3.6. Sprint Goals . . . 15
   3.7. Requirements . . . 17
      3.7.1. Product Backlog . . . 17
      3.7.2. Sprint Backlogs . . . 18

3.8. Scrum Adaptionen . . . 37
3.8.1. Definition of Ready (DOR) . . . 37
3.8.2. Definition of Done (DOD) . . . 37
3.8.3. User Story Template . . . 39
3.8.4. Estimation method . . . 40
3.8.5. Velocity . . . 41
3.8.6. Sprint . . . 41

4. Deployment/Integration 43
4.1. Licensing and Compliance . . . 43
4.1.1. Licensing Overview . . . 43
4.1.2. Compliance with Open Source Licenses . . . 43
4.1.3. Purpose of Compliance . . . 44
4.2. Installation (Sysadmin) Manual & Script . . . 44
4.2.1. Requirements . . . 44
4.2.2. Clone the repository . . . 44
4.2.3. Build and run scripts . . . 44
4.3. User Manual . . . 46
4.3.1. Getting Started . . . 46
4.3.2. Operating Instructions . . . 46

5. Conclusion 48
5.1. Discussion . . . 48
5.1.1. Example from BFH Template - Delete . . . 48
5.2. Bottom Line . . . 48
5.3. Future Work . . . 48

Glossary 49

Bibliography 52

A. Original Project Description 53
A.1. List of Used Libraries and Their Licenses . . . 54

List of Tables

3.1. Scrum Roles 14
3.2. Additional Scrum Roles 15

A.1. List of Used Libraries in the Project 54

List of Figures

2.1. High-level MVC-Pattern from URL-Archiver 5
2.2. Extension with new archiving service XY (using the Factory Pattern) 5
2.3. High-level operational process 7

3.1. Product Backlog 17
3.2. Sprint 1 Backlog 18
3.3. User Story Detail for ”Processing Feedback” 19
3.4. User Story Detail for ”Prompt for file path input” 19
3.5. User Story Detail for ”Automatic File Type Detection” 20
3.6. User Story Detail for ”Processing of Directories” 21
3.7. Sprint 1 Burn Up Chart 22
3.8. Sprint 2 Backlog 23
3.9. User Story Detail for ”Intermediate Documentation” 23
3.10. User Story Detail for ”Creation of Base Structure” 24
3.11. User Story Detail for ”Scan Files for URLs” 24
3.12. User Story Detail for ”Sequential URL Preview” 25
3.13. User Story Detail for ”Implement temporary store for extracted URLs” . . . 25
3.14. User Story Detail for ”Intermediate presentation” 26
3.15. Sprint 2 Burn Down Chart 27
3.16. Sprint 3 Backlog 28
3.17. User Story Detail for ”Intermediate Documentation” 28
3.18. User Story Detail for ”Creation of Base Structure” 29
3.19. User Story Detail for ”Scan Files for URLs” 29
3.20. Sprint 3 Burn Down Chart 29
3.21. Sprint 4 Backlog 30
3.22. User Story Detail for ”Generate CSV File” 30
3.23. User Story Detail for ”Fix bug with selenium on linux” 31
3.24. User Story Detail for ”Fix bug with selenium on MacOS” 31
3.25. User Story Detail for ”Fix bug With selenium and Edge browser” 32
3.26. User Story Detail for ”Document licences” 32
3.27. User Story Detail for ”Periodic complete code review” 32
3.28. User Story Detail for ”Document SCRUM sprint 3” 33
3.29. User Story Detail for ”Creation of a config file” 33
3.30. User Story Detail for ”Progress indicator archiving” 34
3.31. Bug Detail for ”No URL found in file is not handled” 34
3.32. Sprint 4 Burn Down Chart 34

3.33. Sprint 5 Backlog 35
3.34. User Story Detail for ”Integrate Archived URLs into Supported Files” 35
3.35. User Story Detail for ”Create / Fix Tests” 36
3.36. User Story Detail for ”Show archived urls path” 36
3.37. Sprint 5 Burn Down Chart 36
3.38. Screenshot from the user story template. 39
3.39. Ilustration of T-Shirt Sizes 40

Listings

1. Introduction

1.1. Initial Situation

The Internet is constantly evolving, which means that there is no guarantee that a website as it exists today will still exist in a few years' time, let alone contain the same information. While this might not be a concern that the average Internet user has to grapple with, it poses a challenge to the academic demographic, where it becomes crucial to reference sources and potentially integrate links to additional data. If links become inactive, verifying the sources becomes challenging, if not impracticable.

Archiving the existing status of a website is achievable, but it currently necessitates a manual and hence time-intensive operation, which not many people take the time to do. The objective of this project is to devise an automated solution to this predicament that is independent of platforms.

The stakeholders for this solution include:

- Legal professionals and researchers who need to preserve web content as evidence or for case study references.
- Journalists and media agencies that require archiving web pages for future reporting or fact-checking.
- Librarians and archivists tasked with the digital preservation of online materials for historical records.
- Content creators and marketers who wish to maintain records of web content for portfolio or audit purposes.
- Educators and students who need to collect and cite online resources for academic projects and research.
- Organizations and businesses that need to archive their web presence for compliance and record-keeping.

1.2. Poduct Goal

The product goal is a platform independent Java application called “URL-Archiver“. The application must be Free/Libre and Open Source Software (FLOSS) licensed and fulfil the following functionalities:

1. The software should be CLI[1]-based and offer a clear command line.
2. The software should allow the user to input a path, which can be a folder or any Unicode Text File.
3. The software examines the contents of a file or folder to extract any web URLs using a standard regular expression or similar method.
4. If desired, URLs can be automatically opened in a web browser.
5. The extracted URLs are archived on Archive Today and/or Wayback Machine as per the user’s preference.
6. The software outputs the resulting archive URLs to the user.
7. The software generates a CSV file containing the original URL and the archived Version of the URL.
8. Optionally, the archived Versions are written back into the provided BibTeX file.

The product goal is achieved if the software covers all the functionality listed above. Furthermore, the code should be minimalistic, modular, and self-explaining. In addition to the code, it is essential that the following documents are provided:

- User manual
- Installation instructions (including installation script)
- Software documentation

1.3. Priorities

The following priorities are listed in order of importance:

1. **Functionality**: The primary priority is the accurate extraction and archiving of URLs. The software should reliably identify URLs in varied file types and ensure their successful archiving on Archive Today[2] or Wayback Machine[3].

[1] Command Line Interface
[2] https://archive.today
[3] https://web.archive.org/save/

2. **Usability**: Given the diverse potential user base, the program should be platform-independent and possess a user-friendly interface. While the underlying mechanisms may be complex, the user experience should be seamless and intuitive.
3. **Code Quality**: Emphasis should be placed on writing clean, minimal, and modular code. This not only aids in potential future enhancements but also in debugging and troubleshooting.
4. **Documentation**: As with any software project, proper documentation is paramount. The project report should be concise, adhering to the principle of being "maximally informative, minimally long," ensuring clarity of information without overwhelming the reader.
5. **Integration with Existing File Types**: The ability to seamlessly insert archived URLs into BibTeX files is a priority, given the potential academic applications of the software.

2. Specification

2.1. System Delimitation

2.1.1. System Environment (statics)

System Overview

The primary purpose of the URL-Archiver is to extract URLs from Unicode text files and PDFs, and archive them on supported platforms: Archive Today and the Wayback Machine. The system provides the archived URL versions to the user via a CSV file. Additionally, when a BibTeX file is provided by the user, the original BibTeX file is updated with a note field containing these archived URLs for each entry.

Hardware Specifications

The URL-Archiver does not impose any special hardware requirements. However, an internet connection is essential for the archiving process to function.

Software Components

The URL-Archiver is platform-independent , operating on major systems such as Windows (tested on Windows 10, version 22H2 and Windows 11, version 23H2), macOS (tested on macOS Sonoma), and Linux (tested on Ubuntu 20.04.3 LTS). The system has varying browser dependencies based on the operating system: Chrome is required for macOS, Edge for Windows, and Firefox for Ubuntu/Linux (Latest stable versions of the browsers are recommended). Users can change the default Browser in the configuration of the application. Other dependencies are installed with the URL-Archiver and do not require separate installation.

System Architecture

The URL-Archiver uses the Model-View-Controller (MVC) pattern, as illustrated in figure 2.1, to enable future enhancements, such as adding a GUI interface. The Factory Pattern is applied where appropriate to simplify the extension of functionalities. For instance, adding extra archiving services can be easily accomplished by introducing a new archiving service, as shown in figure 2.2.

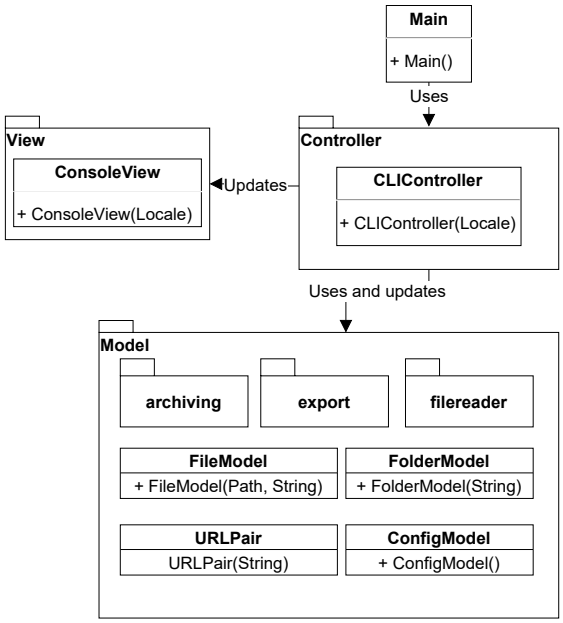


Figure 2.1.: High-level MVC-Pattern from URL-Archiver

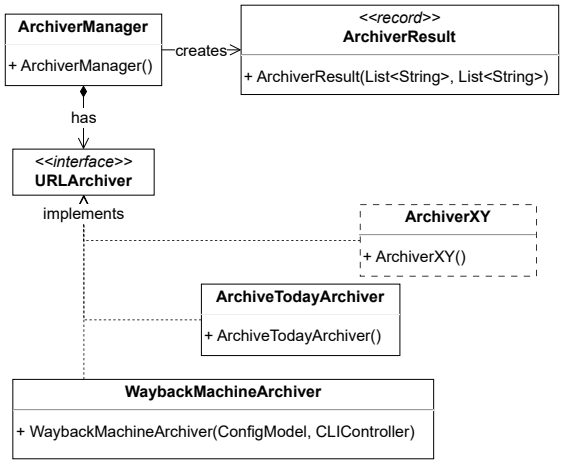


Figure 2.2.: Extension with new archiving service XY (using the Factory Pattern)

Data Management

Upon completion of its execution, the URL-Archiver generates a CSV file where each line contains an extracted URL and its archived versions, separated by semicolons. For example, a line like `https://xy.com;https://web.archive.org/xy;https://archive.ph/xy` shows the original URL and its archives. This simple format makes it easy to track and manage archived URLs.

Optionally, if the user provided a BibTeX file, URLs are integrated into the note field of each entry. If there's no existing note field, a new one is created with the format `note = Archived Versions: url1, url2`. If a note field already exists, the archived URLs are appended to it in the format `note = <current note>, Archived Versions: url1, url2`. This approach ensures that the archived URLs are neatly added to the BibTeX entries, maintaining the integrity of the original data.

User Interface

Currently, the system uses a command-line interface. The MVC-Pattern lays the groundwork for potential future implementation of a GUI interface.

Integration with Other Systems

The system integrates with the Wayback Machine via API, with certain limitations detailed in their API documentation[1]. For archiving on Archive.today, which lacks an API, Selenium is used to automate the process as much as possible. However, users must manually complete captchas.

Maintenance and Support

Currently, there are no specified maintenance requirements or a support framework for the URL-Archiver.

2.1.2. Process Environment (dynamics)

2.1.3. Operational Processes

The URL-Archiver is initiated by the user, who provides a path to Unicode text or PDF files or a directory that contains such files. The application extracts URLs from these files and presents them sequentially to the user. The user then has options to open or archive that URL. He has also the following other options:

- **s**: Show a list of previously archived URLs.

[1] https://archive.org/details/spn-2-public-api-page-docs/mode/2up

- **u**: Update and view pending archive jobs.
- **n**: Navigate to the next URL.
- **q**: Quit the application.
- **c**: Change application settings.
- **h**: Access the Help Menu for assistance.

Upon completion, the user is prompted to save URL pairs to a CSV file and, if a Bibtex file is provided, to write the archived URLs back into it.

Event Handling

In the URL Archiver, user actions are efficiently facilitated through the main menu. When archiving a URL, users can select either the Wayback Machine or Archive.today. In addition, the 'c' option in the menu allows users to configure settings, including setting up API keys for the Wayback Machine and selecting a default browser. The application cleverly handles unsupported input and incorrect paths by prompting the user for the correct information or action. This ensures smooth operation and user guidance throughout the process.

Life Cycle

The URL-Archiver's life cycle begins with launch and path input, proceeding to URL extraction and user interactions via the menu options, and ends with prompts for data saving upon completion. See the high-level process in figure 2.3.

Error Management

Errors within the URL-Archiver are caught and handled, typically prompting the user with a customized error message asking to retry the action. No system stack traces are shown

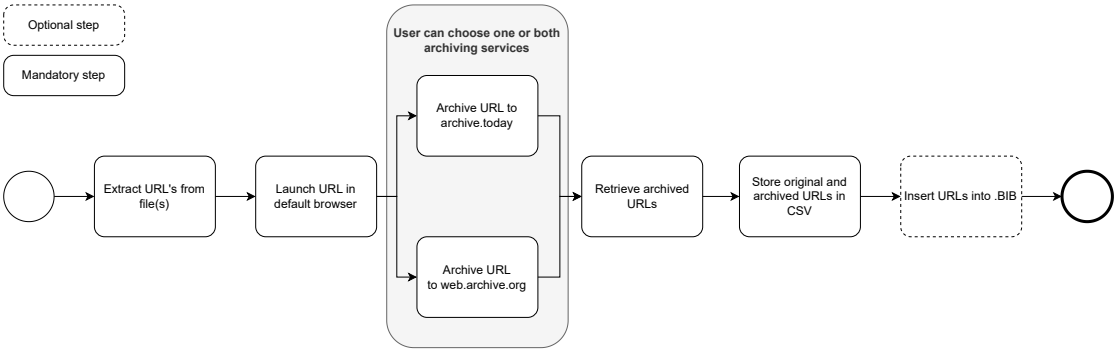


Figure 2.3.: High-level operational process

to the user.

Backup and Recovery

Currently, there are no backup features; progress is not saved if the application ends unexpectedly, which is slated for future improvement.

Update and Upgrade Policies

Software updates require manual download and recompilation from the Git repository. The system does not provide automatic updates or an in-built feature for update checks.

2.2. Requirements

2.2.1. Epics and User Stories

In this section, we outline the main features (Epics) of the project and break them down into detailed user tasks (User Stories). This helps provide a clear understanding of the desired functions and behaviors of our software.

Epic 1: File Input and Processing

Goal: Allow the user to input various file types via the command line and prepare these files for further processing.

1. **Prompt for File Path Input**
   - **Description**: As a user, when I start the tool, I want to be prompted to input the path to my file, so the tool knows which file to process.
   - **Acceptance Criteria**:
     - Upon starting the tool, it prompts the user to enter a file path.
     - On inputting an invalid path or if there are permissions issues, the tool provides a relevant error message.
2. **Automatic File Type Detection**
   - **Description**: As a user, I want the tool to automatically detect the file type (based on file extension) and treat it accordingly so that I don't need to specify the file type separately.
   - **Acceptance Criteria**:
     - The tool automatically identifies if the file is a .BIB, .TEX, .HTML, or .PDF.

- For unrecognized file types, the tool provides an appropriate error message.

3. **Processing of Directories**
   - **Description**: As a user, I want to input a whole directory, so the tool processes all supported files contained within.
   - **Acceptance Criteria**:
     - The tool can accept directory paths after the prompt.
     - It processes all supported file types within the directory.
     - The tool gives a message if files within the directory are skipped due to their type.
4. **Processing Feedback**
   - **Description**: As a user, I want to receive feedback when the tool starts processing the file and when it finishes, to know the status.
   - **Acceptance Criteria**:
     - A message is displayed when the processing of a file starts.
     - Upon completion, a confirmation message is shown, which also includes any potential errors or warnings.

Epic 2: URL Detection and Extraction

Goal: Accurately detect and extract URLs from input files for further processing.

1. **Scan Files for URLs**
   - **Description**: As a user, I want the system to scan my input files and identify any embedded URLs so that they can be extracted for archiving.
   - **Acceptance Criteria**:
     - System can detect URLs in a variety of file formats including .BIB, .TEX, .HTML, and .PDF.
     - Detected URLs are listed without any duplication.
2. **Use Regular Expressions for Extraction**
   - **Description**: As a user, I want the system to use regular expressions or other reliable techniques to extract URLs so that all valid URLs are captured without error.
   - **Acceptance Criteria**:

- System uses a robust regular expression pattern that matches most URL formats.
- Extracted URLs are validated to ensure they are in the correct format.

3. **Store URL Line Number or Context**
   - **Description**: As a user, when a URL is detected and extracted, I want the system to also store its line number or contextual information from the original file, enabling precise placement of its archived counterpart later on.
   - **Acceptance Criteria**:
     - Upon URL detection, the system captures and stores the line number or relevant context of the URL from the source file.
     - This information is utilized later if archived URLs need to be placed back into the original files.
4. **Compile a List of URLs**
   - **Description**: After extraction, I want all URLs to be compiled into a single list, eliminating any duplicates, so that I have a clean list for archiving.
   - **Acceptance Criteria**:
     - The list contains all the unique URLs found in the input files.
     - Invalid or broken URLs are flagged or removed from the list.

Epic 3: Web Browser Integration

Goal: Seamlessly open detected URLs, one at a time, in a web browser for user verification, and immediately initiate the archiving process upon user decision.

1. **Sequential URL Preview**
   - **Description**: As a user, I want to preview each detected URL in my default browser sequentially to verify its content.
   - **Acceptance Criteria**:
     - System opens one URL at a time in the default browser.
     - Immediately after the URL is displayed, the system presents the user with the option to archive.
2. **Immediate Archiving Upon Decision**
   - **Description**: After reviewing a URL in the browser, I want to decide if it should be archived. If I decide to archive, the system should immediately initiate the archiving process.

- **Acceptance Criteria**:
  - System provides a prompt to accept or decline the archiving of the displayed URL.
  - If the user chooses to archive, the system directly begins the archiving process, and the user may need to manually solve captchas.

3. **Track Archiving Progress**
   - **Description**: As a user, I want a clear indicator of how many URLs have been displayed, archived, and how many are left to process.
   - **Acceptance Criteria**:
     - The system displays a counter indicating the number of URLs already shown to the user.
     - Another counter indicates how many URLs have been chosen for archiving.
     - Yet another counter shows how many URLs remain to be processed/displayed.
4. **Store User Decisions for Reporting**
   - **Description**: As a user, after making a decision about archiving each URL, I want the system to store my choices so that they can be referred to or reported on later.
   - **Acceptance Criteria**:
     - The system maintains a record of each URL and the user's decision (archived or not archived).
     - The stored decisions are available for any subsequent reporting needs.

Epic 4: Interaction with archive.ph

Goal: Automate the process of archiving URLs via archive.ph while ensuring user interaction is seamless and all necessary data is captured for later use.

1. **Automated URL Submission**
   - **Description**: As a user, I want the system to automatically fill in the URL into the archive.ph input field and submit it for archiving.
   - **Acceptance Criteria**:
     - Upon initiation, system opens the archive.ph website in a browser.
     - System auto-fills the given URL into the appropriate input field.
     - System automatically triggers the submission process for archiving.

2. **User Interaction for Captchas**
   - **Description**: If required, I want to manually solve captchas to ensure the URL gets archived.
   - **Acceptance Criteria**:
     - If archive.ph presents a captcha, the system allows the user to solve it manually.
     - The archiving process proceeds once the captcha is successfully solved.
3. **Automatic Retrieval of Archived URL**
   - **Description**: Once a URL is archived, I want the system to automatically retrieve and display the archived URL to me.
   - **Acceptance Criteria**:
     - System captures the new archived URL from archive.ph after the process completes.
     - The archived URL is displayed to the user immediately.
     - The archived URL is stored for later processing and reporting.

Epic 5: Output and Reporting

Goal: Provide the user with an organized CSV file detailing URLs and their archived counterparts. Also, allow for integration of archived URLs back into supported input files.

1. **Generate CSV File**
   - **Description**: As a user, I want the system to produce a CSV file containing all original URLs and their corresponding archived URLs.
   - **Acceptance Criteria**:
     - A CSV file is generated upon completion of the archiving process.
     - Each row in the CSV contains the original URL and its archived counterpart.
2. **Integrate Archived URLs into Supported Files**
   - **Description**: If desired, I want the system to insert the archived URL back into the original file, following its corresponding original URL.
   - **Acceptance Criteria**:
     - The system recognizes supported file types for this integration process.
     - Upon user approval, the archived URL is inserted in the appropriate location (e.g., following its original URL) within the file.

2.2.2. Functional requirements (added value)

2.2.3. Boundary and Pre-Conditions

2.3. Usability

2.3.1. Personas

2.3.2. Storyboard

2.3.3. UX-Prototyping

3. Implementation

3.1. Architecture (e.g., back-/frontend)

3.2. Processes

3.3. Allocation of roles

In this chapter, the Scrum roles (Product Owner, Scrum Master, Developer) and additional roles such as Customer, Stakeholder, etc. are defined.

3.4. Scrum roles

We have decided to structure our Scrum team in the following manner:

Role	Person
Product Owner	Nicolin Dora
Scrum Master	Abidin Vejseli
Developer	Nicolin Dora, Abidin Vejseli, Kilian Wampfler

Table 3.1.: Scrum Roles

Nicolin took on the role of Product Owner as he had concrete ideas and visions for the product at the start of the project. Additionally, he took on this role because he wanted to deal with the subjects surrounding the product backlog.

Abidin took on the role of the Scrum Master as he has the most experience with the agile way of working. He has already had the opportunity to perform this role professionally on several smaller projects in the past.

Kilian took on the role of a Developer, as he is an active programmer in his job and has already gained some experience with Scrum. Therefore, self-organization is not a foreign concept to him.

Besides Kilian, all the other members of the group were also assigned the role of Developer, as otherwise the project would not have been feasible in the given time. This is due to the

fact that we all work alongside the university.

3.5. Additional roles

In addition to the Scrum roles, we have assigned the following roles to our specialist lecturer and PM-coach.

Role	Person
Stakeholder	Dr. Simon Kramer
Customer	Dr. Simon Kramer
PM-Advisor	Frank Helbling

Table 3.2.: Additional Scrum Roles

3.6. Sprint Goals

We have defined the goals of our past and current sprints in the best possible way according to the SMART[1] criteria. The goals of our sprints are listed below:

Sprint 1 Implement input handler for files (any unicode file e.g. .bib, .txt, .html and .pdf) and basic user guidance (Menu, Error messages).

Sprint 2 Implement a function to scan a provided text in order to identify and extract any URLs contained within it and upgrade our current console-based interface to enable users to easily open any extracted URL using their default web browser.

Sprint 3 Develop and implement a fully automated URL submission system that integrates with the Wayback Machine and Archive Today to ensure at least a 98

Sprint 4 Enhance the system's stability and usability by resolving identified Selenium bugs across Linux, macOS, and Edge browsers, documenting the sprint process and licenses, conducting a thorough code review, and establishing a new configuration management file, aiming for zero critical bugs at sprint closure and readying the system for seamless URL archiving integration in subsequent sprints.

[1] Atlassian blogpost: "How to write smart goals"

Sprint 5 Complete application refactoring for asynchronous archiving and .BIB file URL integration, ensuring no critical bugs and preparing for seamless future enhancements.

Sprint 6 TODO - Add Goal

3.7. Requirements

In this chapter, we present our product and sprint backlogs, structured according to Scrum.

3.7.1. Product Backlog

Our product backlog consists of user stories and epics created by our Product Owner. The user stories are prioritised and represent a set of initial requirements that must be met to achieve our product goal. The product backlog is maintained in Jira.

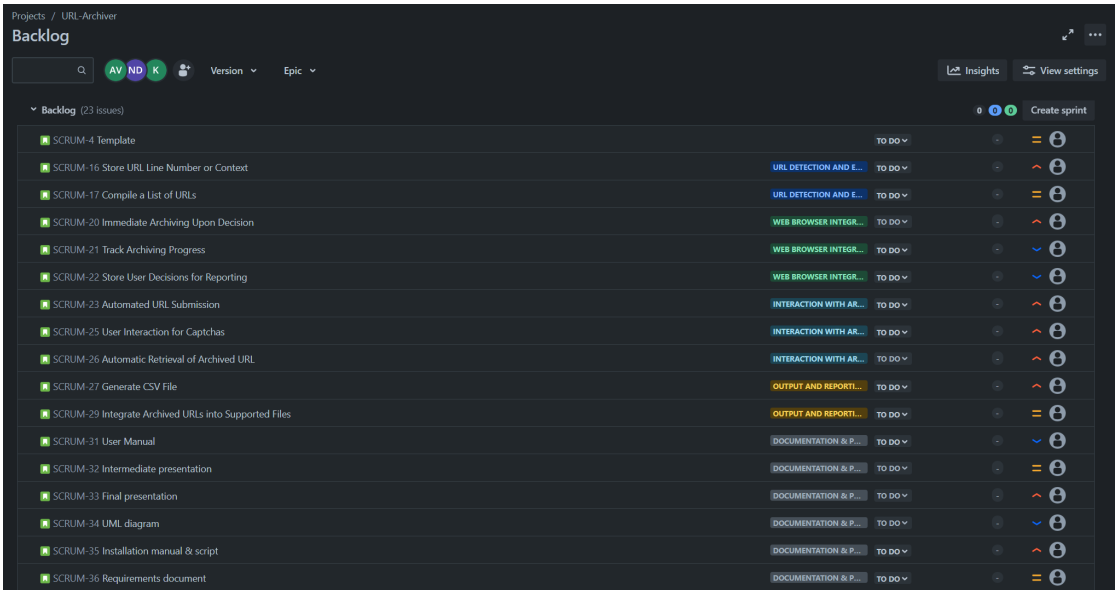


Figure 3.1.: Product Backlog

The prioritisation of user stories in the backlog is based on the business value field, which has a value between one and ten. The business value is a vague estimate of how much value the individual user story has to the business, or in our case, to our stakeholders. We endeavour to estimate the business value based on the expected importance of the function to the stakeholder. The following priorities are possible:

- Highest
- High
- Medium
- Low
- Lowest

3.7.2. Sprint Backlogs

Below we describe our recent and current sprints. During the sprint planning we fill the respective sprint backlog with user stories that serve the sprint goal. A user story must satisfy our Definition of Ready before it can be included in the sprint. In addition, the stories must be estimated and the total number of story points must not exceed our defined velocity.

Sprint 1

Below is a screenshot of our board from the first sprint with the corresponding sprint goal.

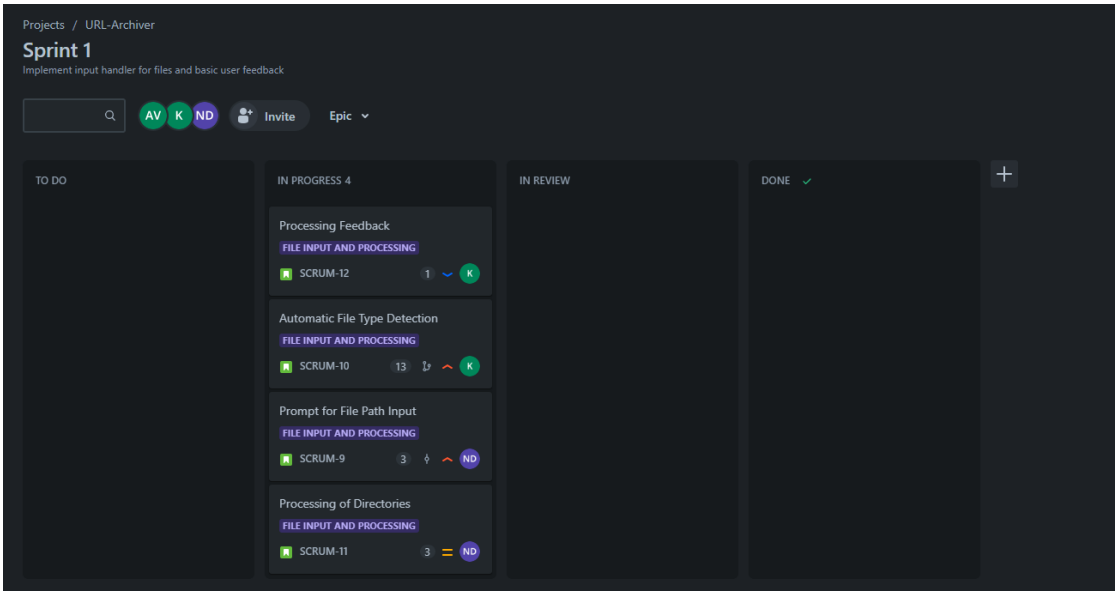


Figure 3.2.: Sprint 1 Backlog

The user stories from the first sprint are shown below. The stories have been estimated and prioritised.

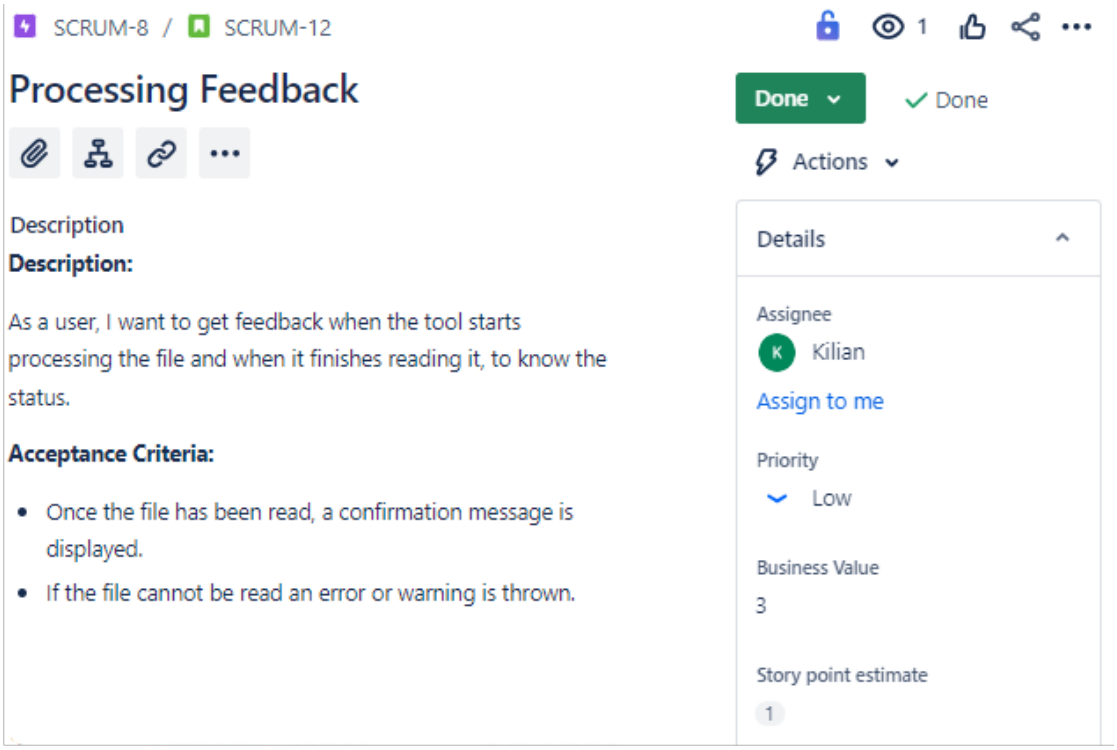


Figure 3.3.: User Story Detail for "Processing Feedback"

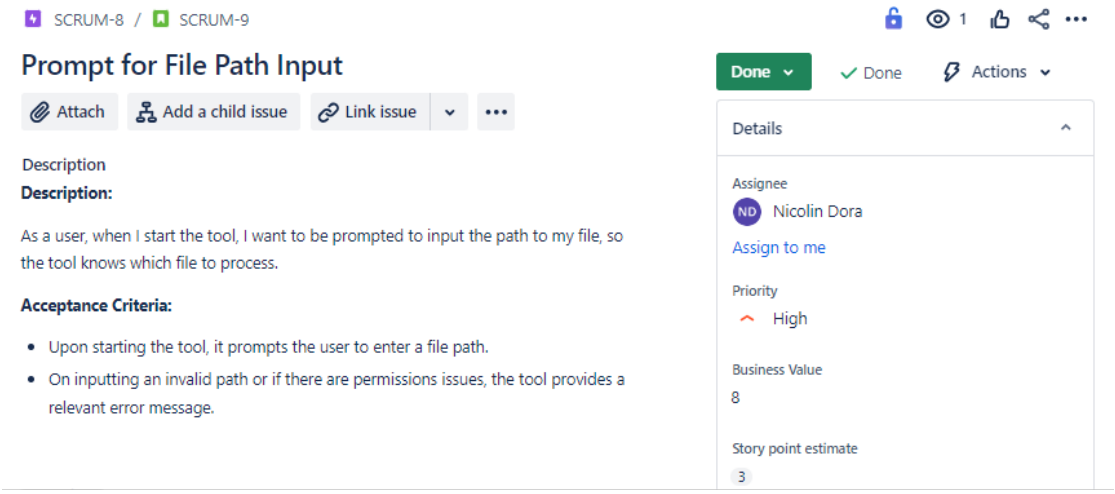


Figure 3.4.: User Story Detail for "Prompt for file path input"

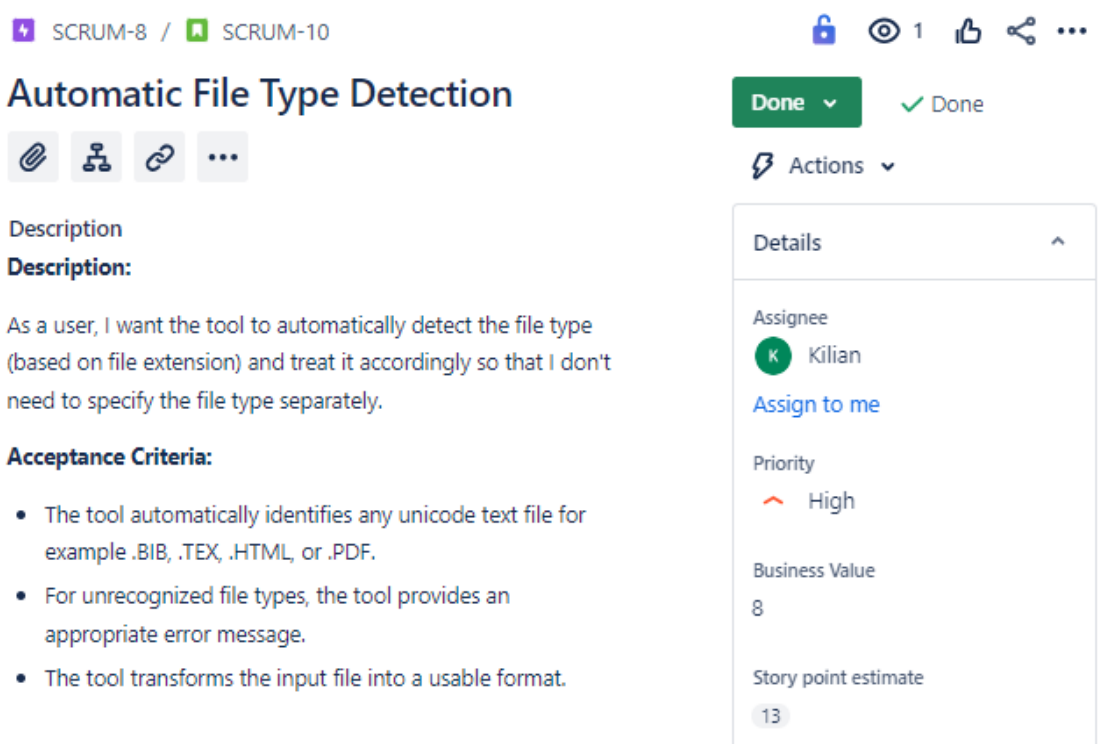


Figure 3.5.: User Story Detail for "Automatic File Type Detection"

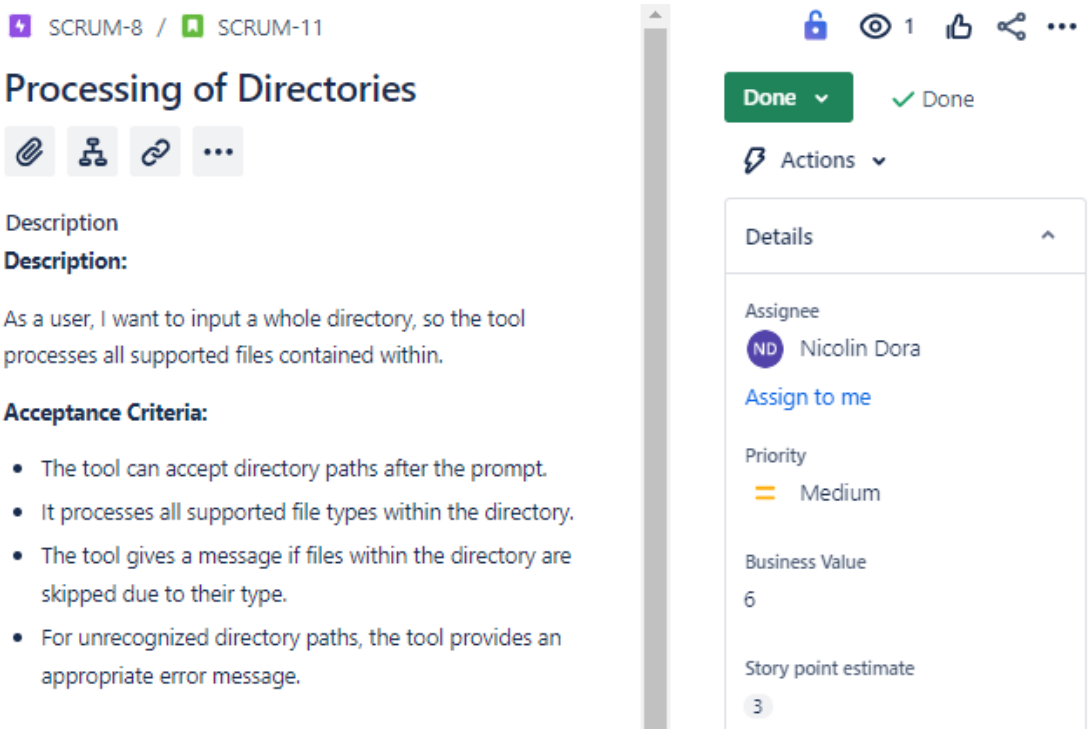


Figure 3.6.: User Story Detail for "Processing of Directories"

In the first sprint, the burn down chart looks like this:

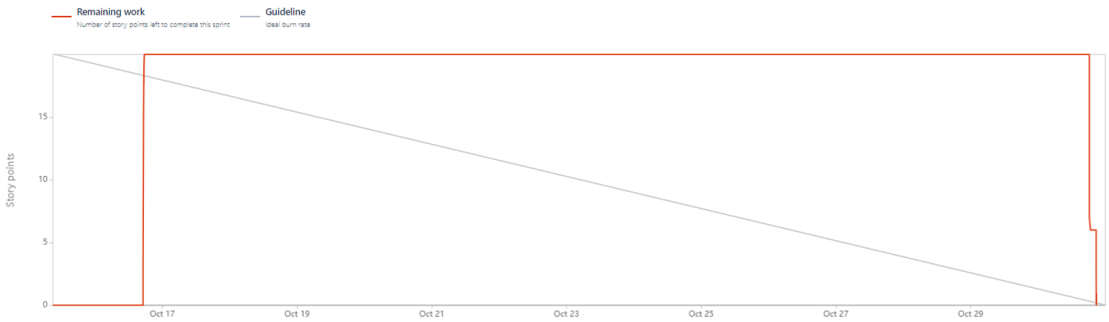


Figure 3.7.: Sprint 1 Burn Up Chart

The reason for this is that we did not create tasks for our user stories as they were small enough. Furthermore, a code review for corresponding user stories could only be conducted towards the end of the sprint, resulting in the finalisation of user stories at that point.

Sprint 2

Below is a screenshot of our board from the second sprint with the corresponding sprint goal.

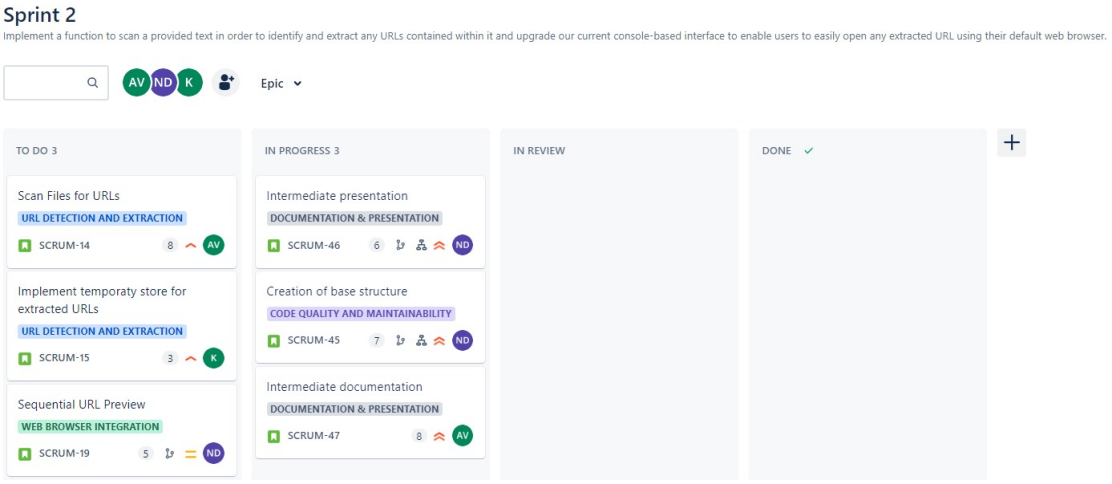


Figure 3.8.: Sprint 2 Backlog

The user stories from the second sprint are shown below. The stories have been estimated and prioritised.

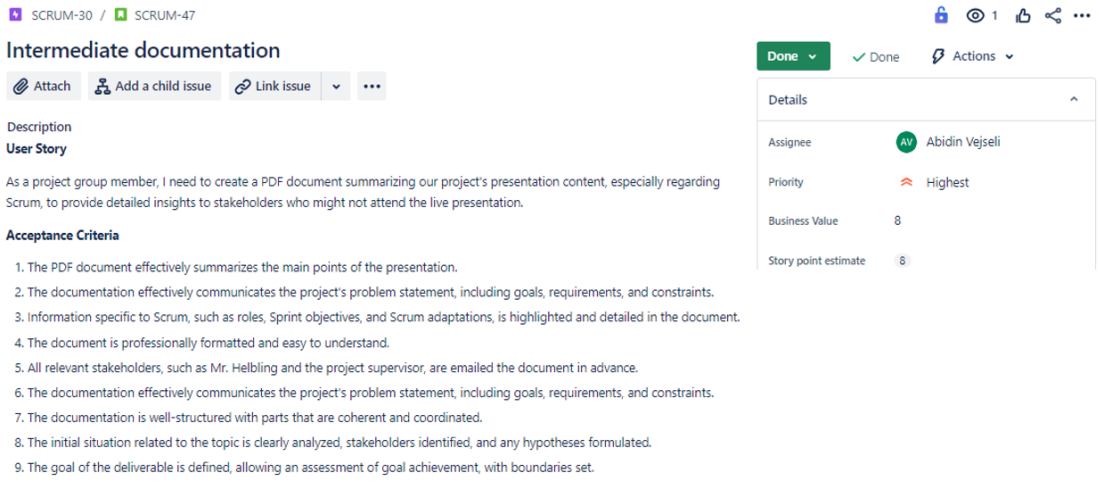


Figure 3.9.: User Story Detail for "Intermediate Documentation"

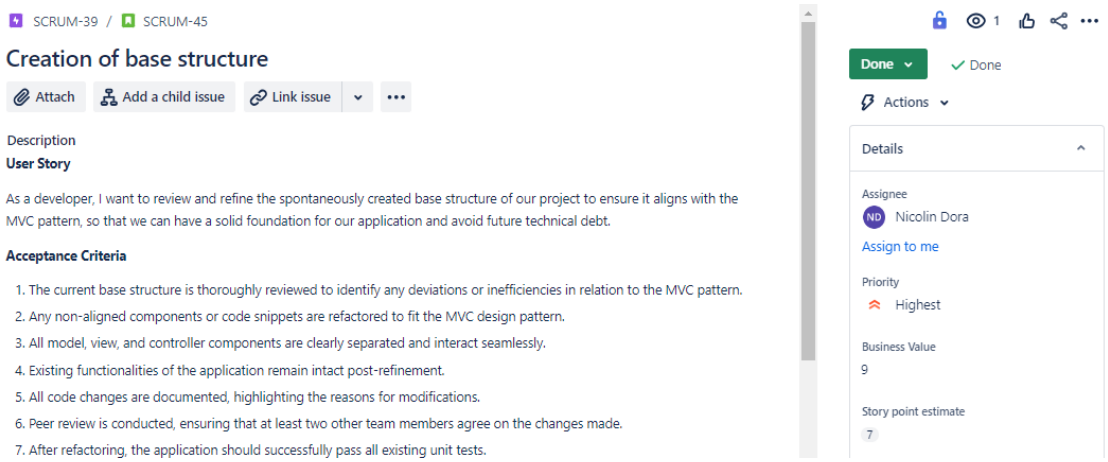


Figure 3.10.: User Story Detail for "Creation of Base Structure"

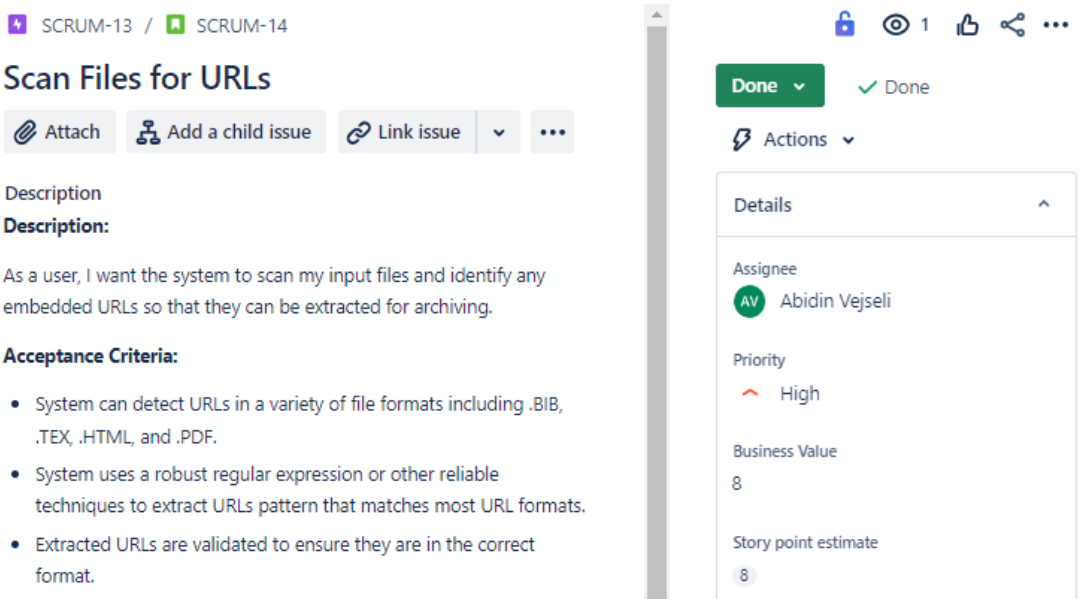


Figure 3.11.: User Story Detail for "Scan Files for URLs"

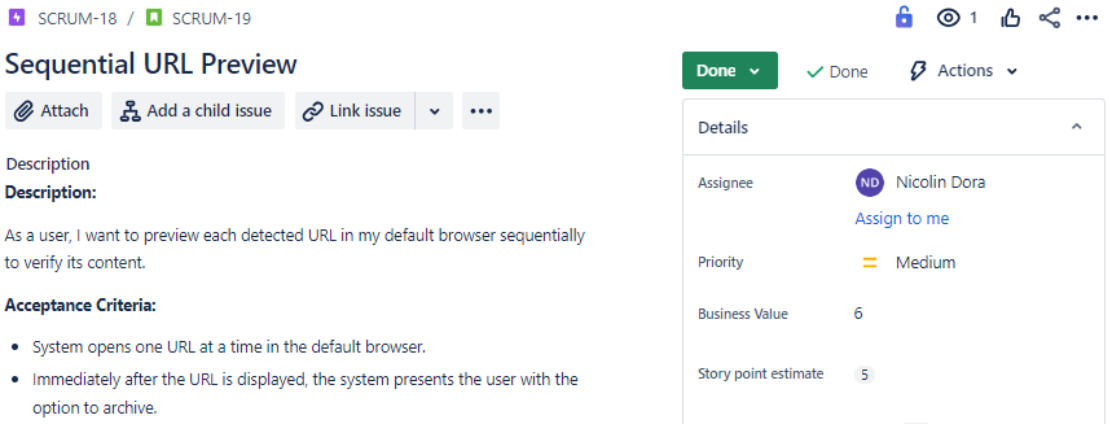


Figure 3.12.: User Story Detail for "Sequential URL Preview"

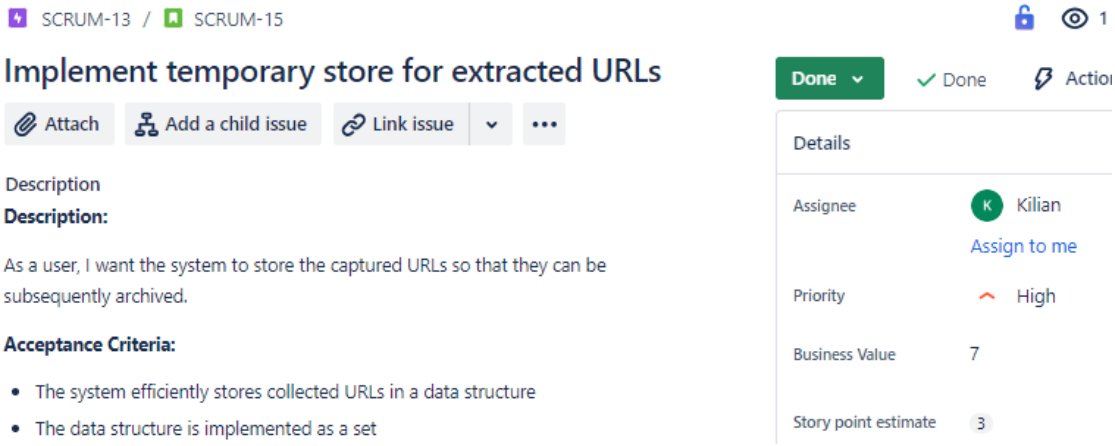


Figure 3.13.: User Story Detail for "Implement temporary store for extracted URLs"

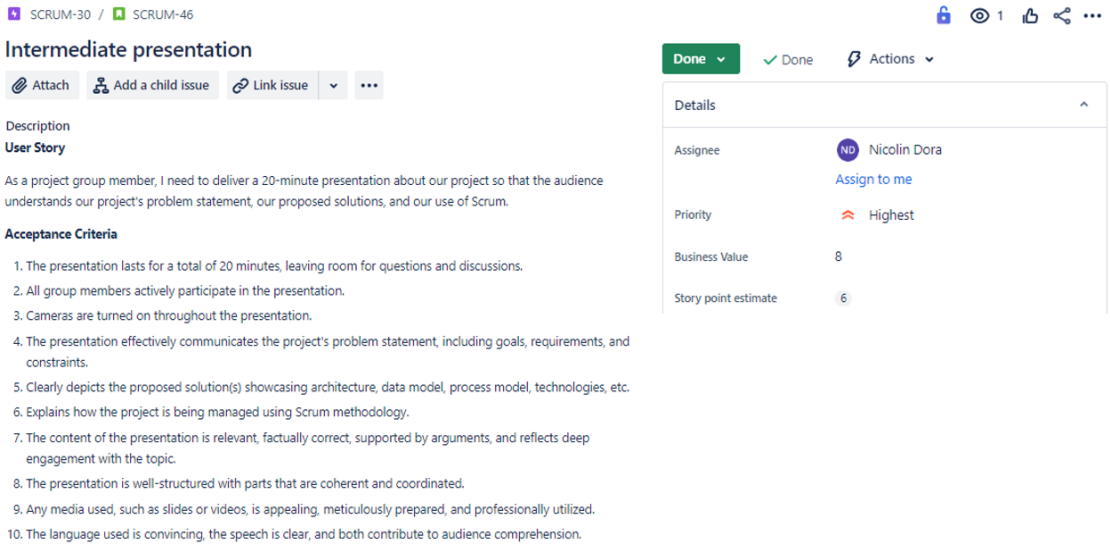


Figure 3.14.: User Story Detail for "Intermediate presentation"

In the second sprint, the burn down chart looks like this:

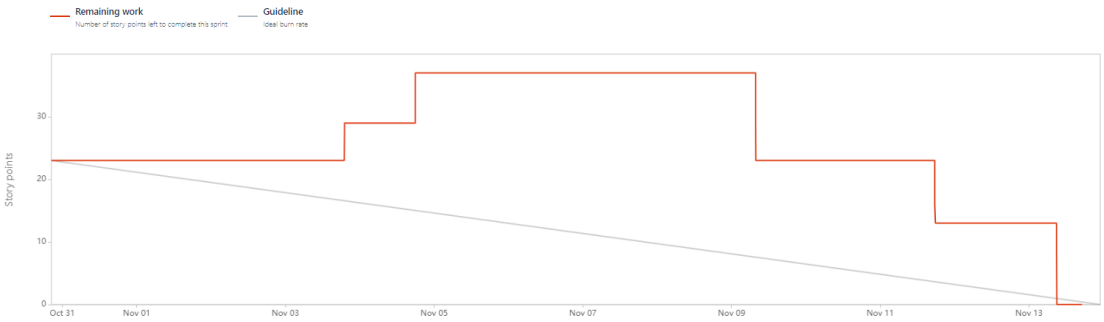


Figure 3.15.: Sprint 2 Burn Down Chart

Compared to the initial sprint, our workload significantly increased, and we introduced new user stories after the sprint began. Despite these additions, we maintained a strong pace and seamlessly managed the extra tasks that arose from the intermediate presentation and documentation requirements.

Sprint 3

Below is a screenshot of our board from the third sprint with the corresponding sprint goal.

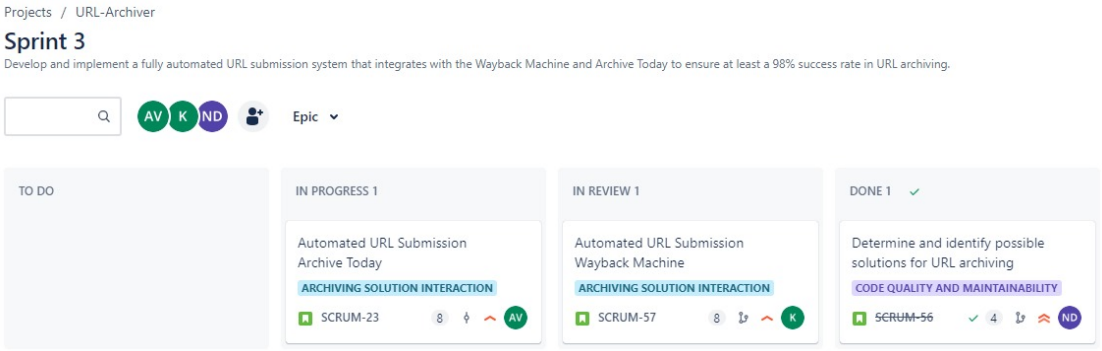


Figure 3.16.: Sprint 3 Backlog

The user stories from the third sprint are shown below. The stories have been estimated and prioritised.

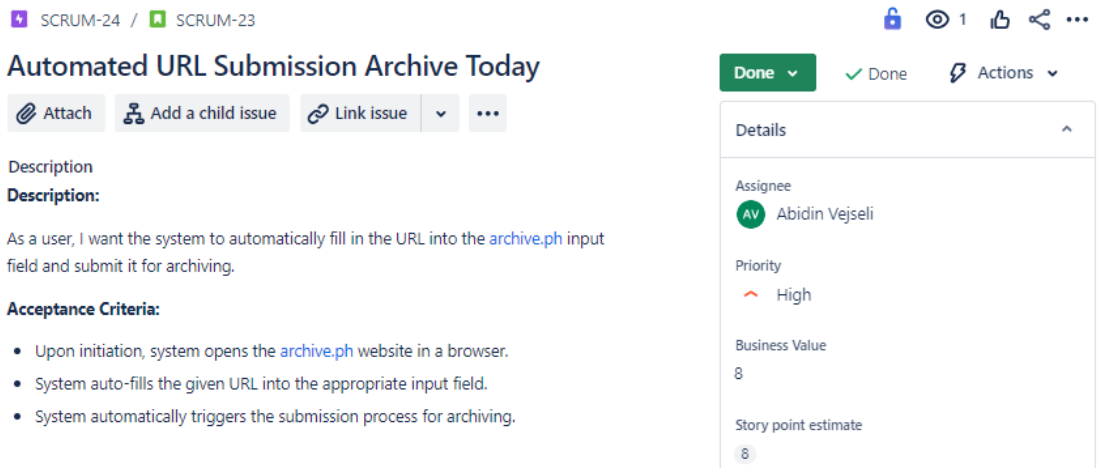


Figure 3.17.: User Story Detail for "Intermediate Documentation"

In the third sprint, the burn down chart looks like this:

During the third sprint, our progress on user stories was limited to the completion of only three due to the 'special week 3'. This meant that these stories had to be completed in the second half of the sprint due to the heavy workload of this special week. The impact of this adaptation to our sprint schedule due to 'special week 3' is reflected in the trends seen in our burndown chart.

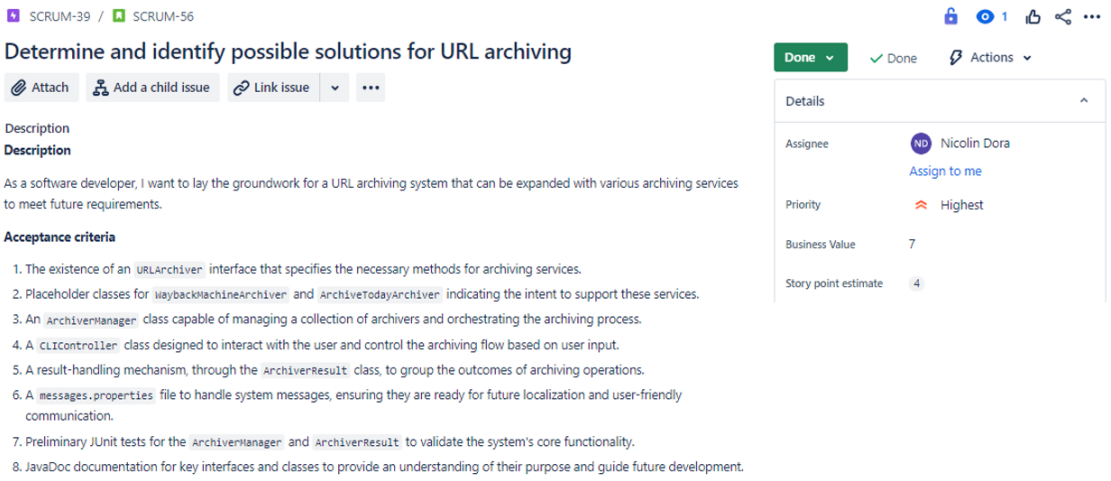


Figure 3.18.: User Story Detail for "Creation of Base Structure"

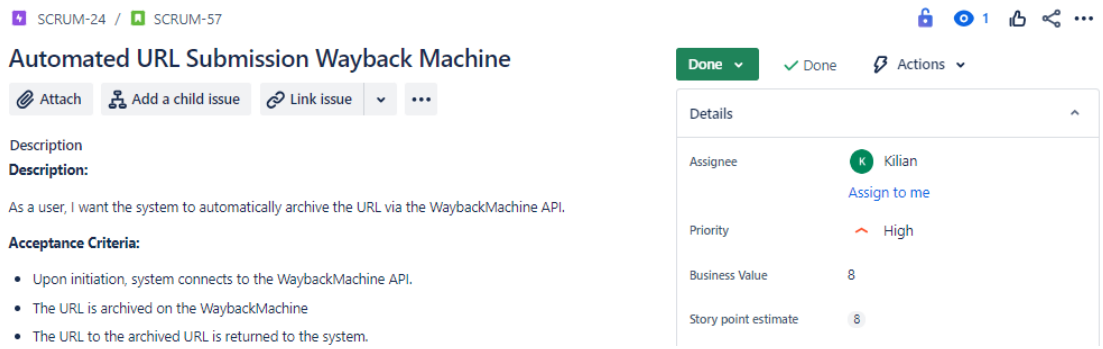


Figure 3.19.: User Story Detail for "Scan Files for URLs"

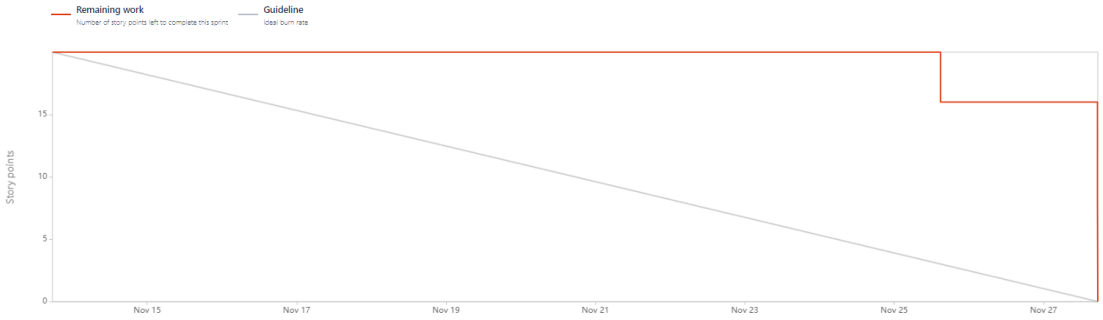


Figure 3.20.: Sprint 3 Burn Down Chart

Sprint 4

Below is a screenshot of our board from the forth sprint.

Key	Summary	Issue type	Epic	Status	Assignee	Story points
SCRUM-27	Generate CSV File	Story	OUTPUT AN...	DONE	K	4
SCRUM-58	No URL found in file is not handled	Bug	FILE INPUT ...	DONE	ND	5
SCRUM-60	Fix bug with selenium on linux	Story	ARCHIVING ...	DONE	ND	3
SCRUM-61	Fix bug with selenium on MacOS	Story	ARCHIVING ...	DONE	K	3
SCRUM-62	Fix bug with selenium and Edge browser	Story	ARCHIVING ...	DONE	AV	3
SCRUM-63	Document licences	Story	DOCUMENT...	DONE	ND	4
SCRUM-64	Periodic complete code review	Story	CODE QUALI...	DONE	ND	4
SCRUM-65	Document SCRUM sprint 3	Story	DOCUMENT...	DONE	AV	4
SCRUM-66	Creation of a config file	Story	CODE QUALI...	DONE	K	4
SCRUM-75	Progress indicator archiving	Story	ARCHIVING ...	DONE	ND	4

Figure 3.21.: Sprint 4 Backlog

The user stories from the forth sprint are shown below. The stories have been estimated and prioritised.

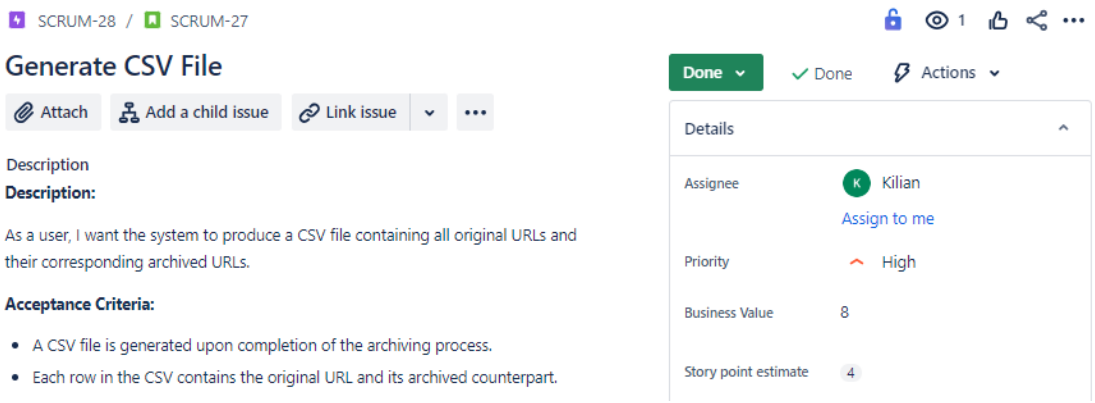


Figure 3.22.: User Story Detail for "Generate CSV File"

In the forth sprint, the burn down chart looks like this:

Compared to the previous sprint, we were more efficient and successfully accommodated additional user stories that were initiated after the sprint began. The completion of all

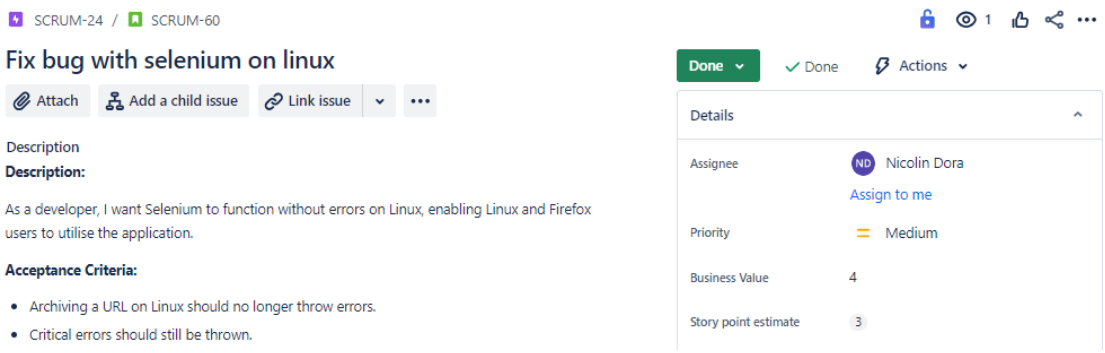


Figure 3.23.: User Story Detail for "Fix bug with selenium on linux"

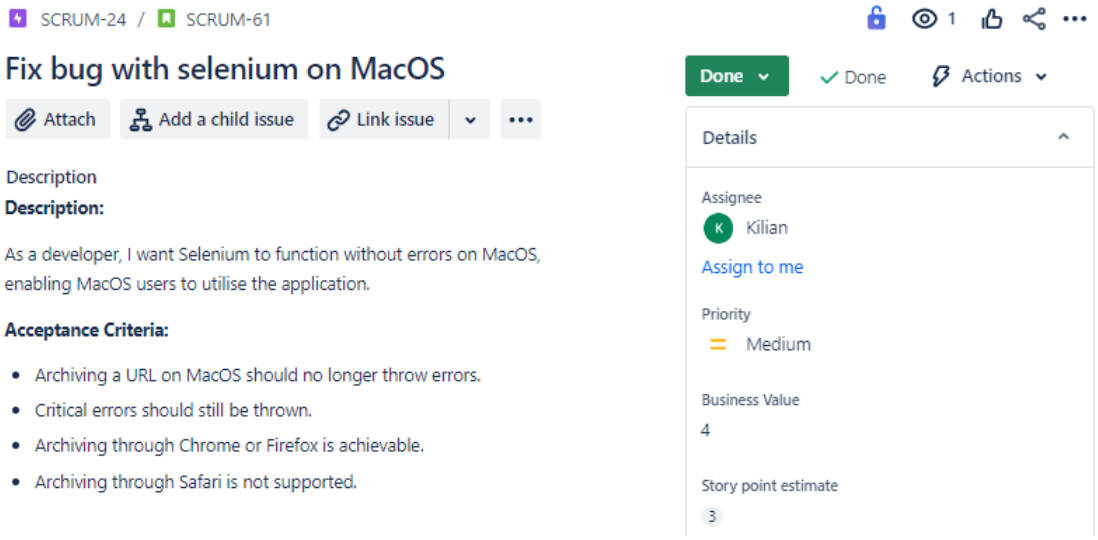


Figure 3.24.: User Story Detail for "Fix bug with selenium on MacOS"

allocated user stories within the sprint timeframe demonstrates our solid teamwork and sprint management capabilities.

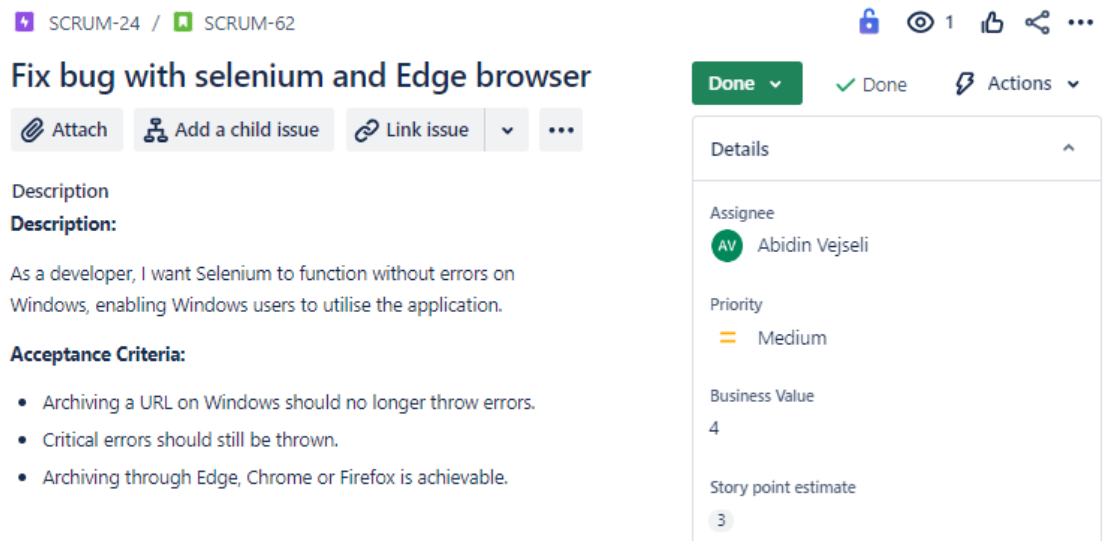


Figure 3.25.: User Story Detail for "Fix bug With selenium and Edge browser"

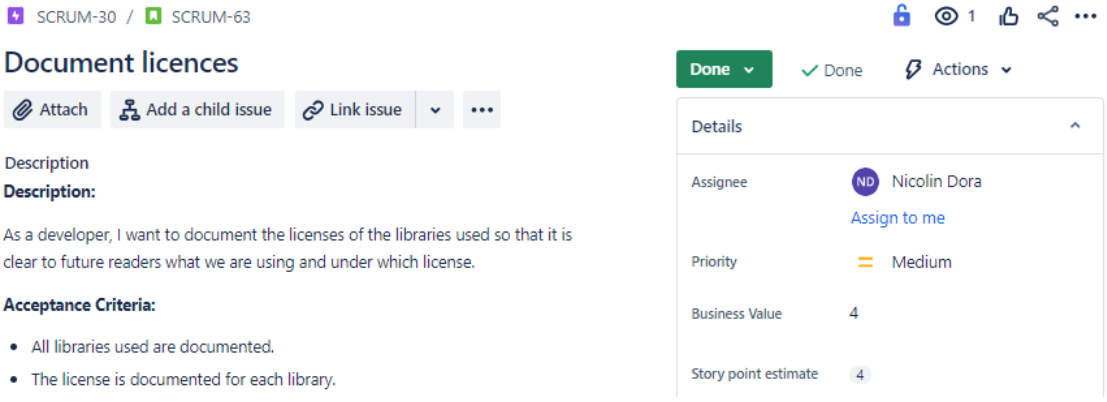


Figure 3.26.: User Story Detail for "Document licences"

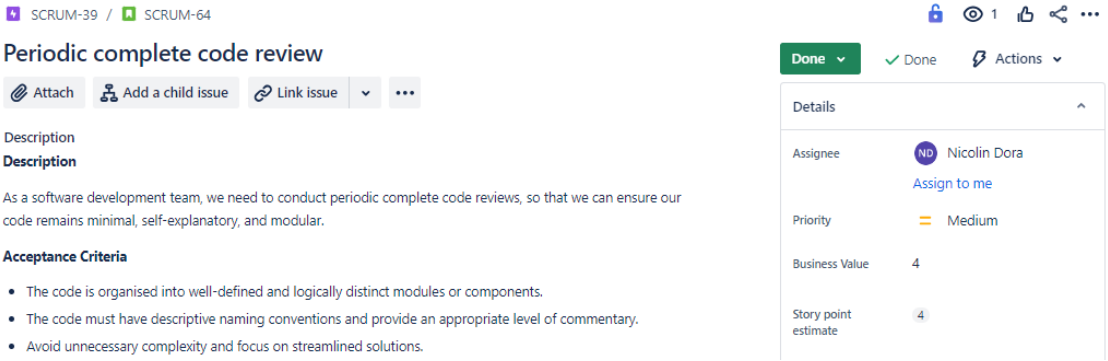


Figure 3.27.: User Story Detail for "Periodic complete code review"

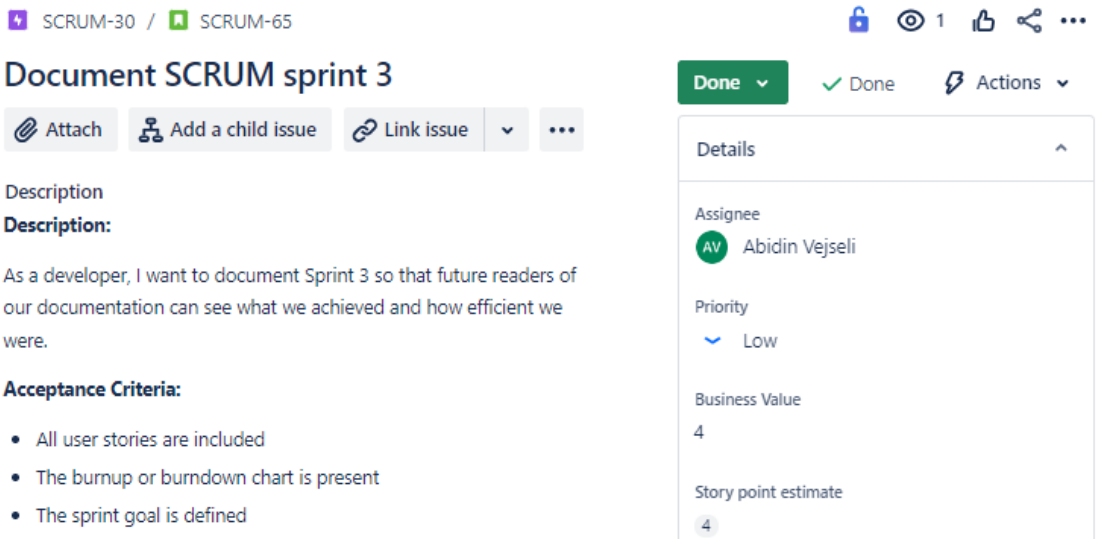


Figure 3.28.: User Story Detail for "Document SCRUM sprint 3"

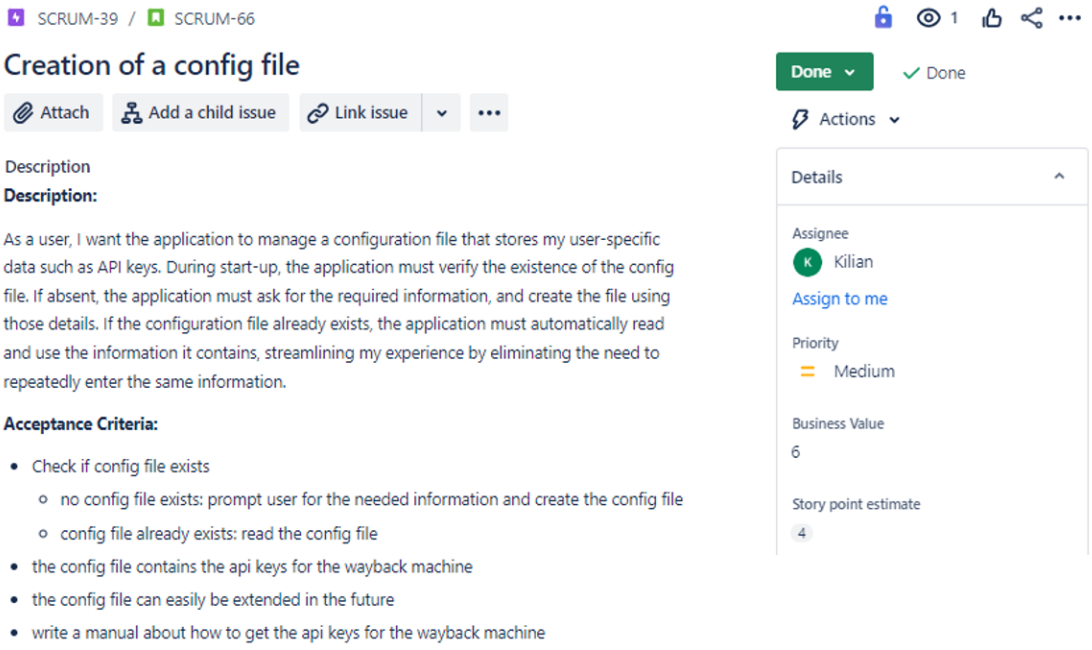


Figure 3.29.: User Story Detail for "Creation of a config file"

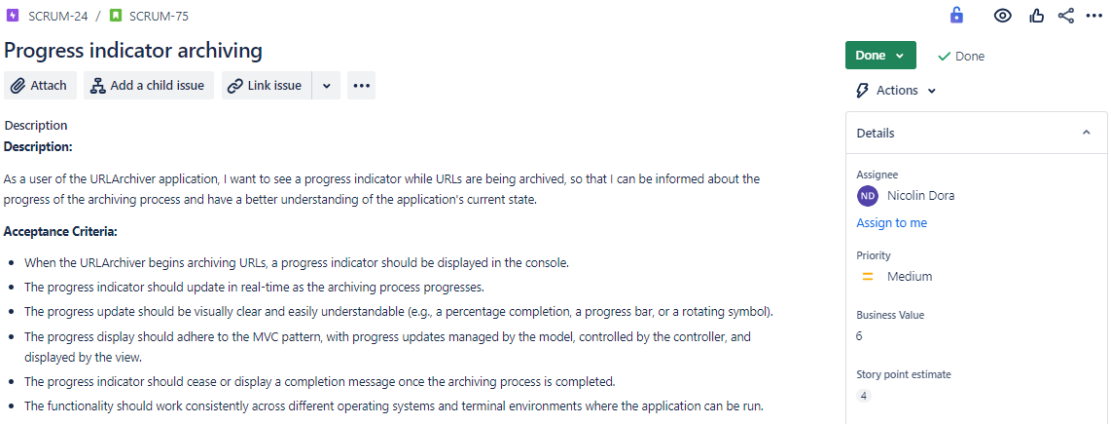


Figure 3.30.: User Story Detail for "Progress indicator archiving"

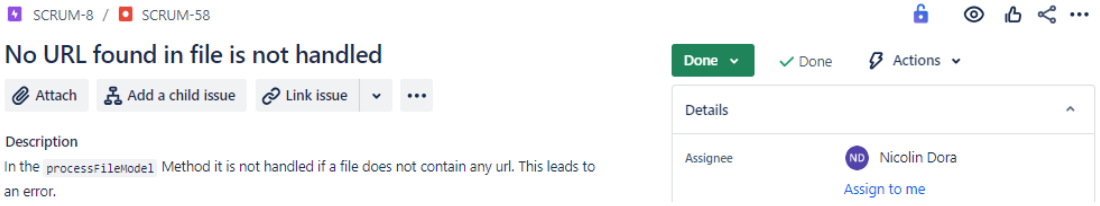


Figure 3.31.: Bug Detail for "No URL found in file is not handled"

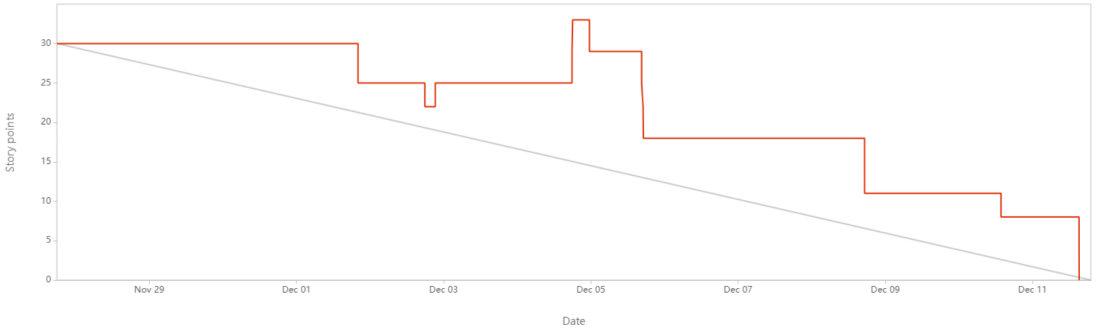
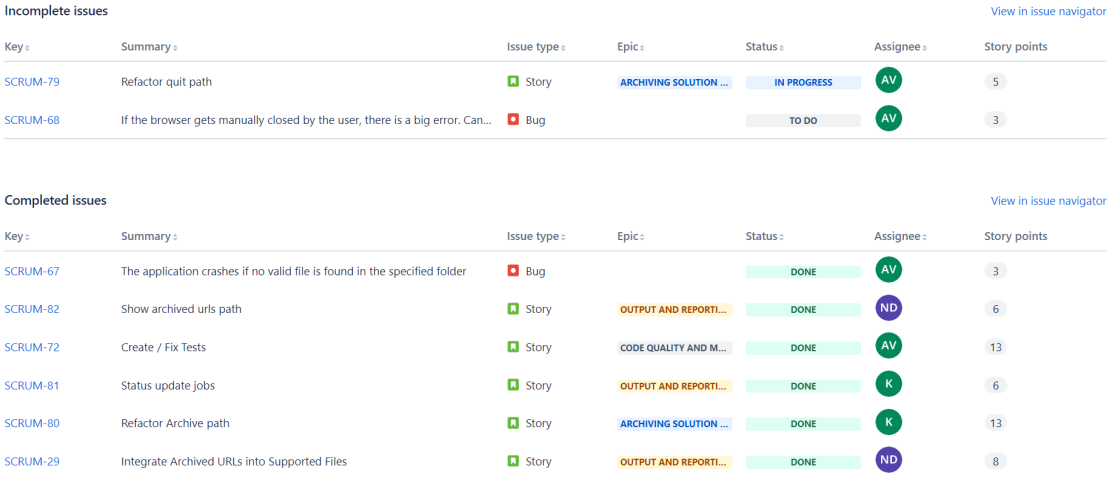


Figure 3.32.: Sprint 4 Burn Down Chart

Sprint 5

Below is a screenshot of our board from the fifth sprint.



Incomplete issues — View in issue navigator

Key	Summary	Issue type	Epic	Status	Assignee	Story points
SCRUM-79	Refactor quit path	Story	ARCHIVING SOLUTION ...	IN PROGRESS	AV	5
SCRUM-68	If the browser gets manually closed by the user, there is a big error. Can...	Bug		TO DO	AV	3

Completed issues — View in issue navigator

Key	Summary	Issue type	Epic	Status	Assignee	Story points
SCRUM-67	The application crashes if no valid file is found in the specified folder	Bug		DONE	AV	3
SCRUM-82	Show archived urls path	Story	OUTPUT AND REPORTI...	DONE	ND	6
SCRUM-72	Create / Fix Tests	Story	CODE QUALITY AND M...	DONE	AV	13
SCRUM-81	Status update jobs	Story	OUTPUT AND REPORTI...	DONE	K	6
SCRUM-80	Refactor Archive path	Story	ARCHIVING SOLUTION ...	DONE	K	13
SCRUM-29	Integrate Archived URLs into Supported Files	Story	OUTPUT AND REPORTI...	DONE	ND	8

Figure 3.33.: Sprint 5 Backlog

The user stories from the fifth sprint are shown below. The stories have been estimated and prioritised.

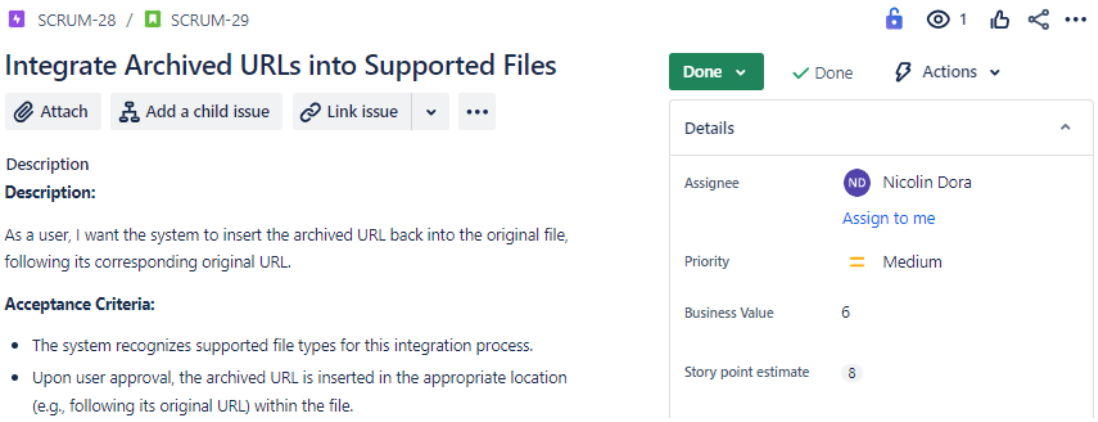


Figure 3.34.: User Story Detail for "Integrate Archived URLs into Supported Files"

In the fifth sprint, the burn down chart looks like this:

In sprint 5, our time was limited due to commitments in other modules and the upcoming Christmas period, which resulted in team member absences. We encountered an unexpected challenge with the extensive time required for test refactoring, as we prioritise quality, which often demands more time. As a result, we were unable to complete all planned user stories and had to carry them over to the next sprint. These challenges are reflected

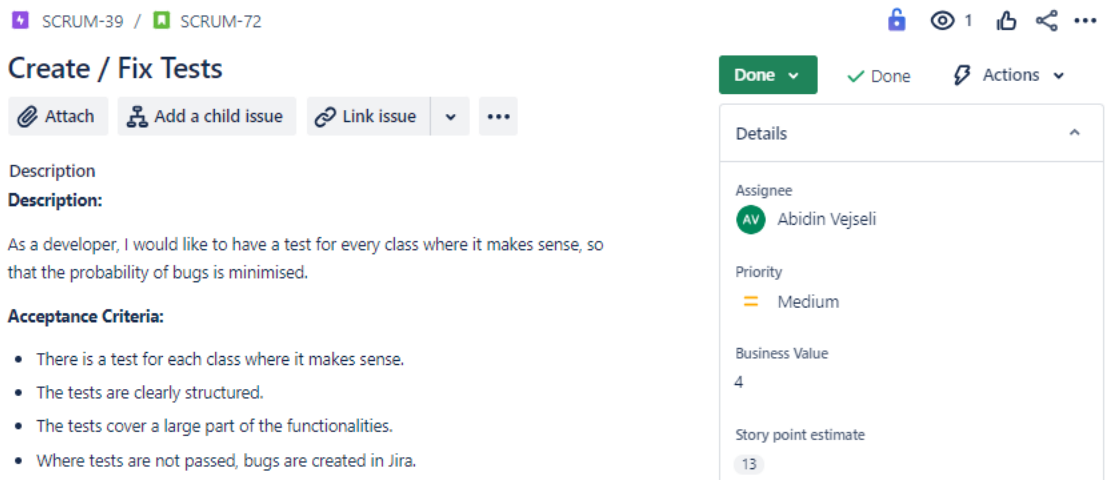


Figure 3.35.: User Story Detail for "Create / Fix Tests"

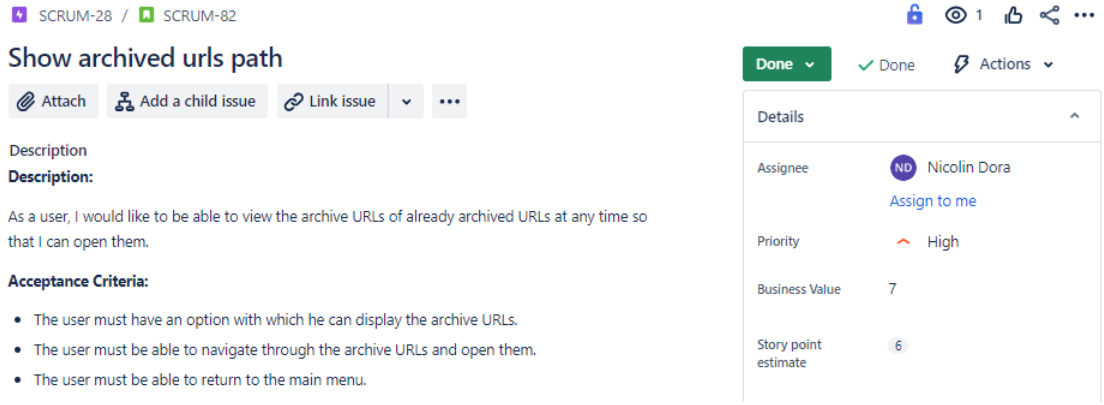


Figure 3.36.: User Story Detail for "Show archived urls path"

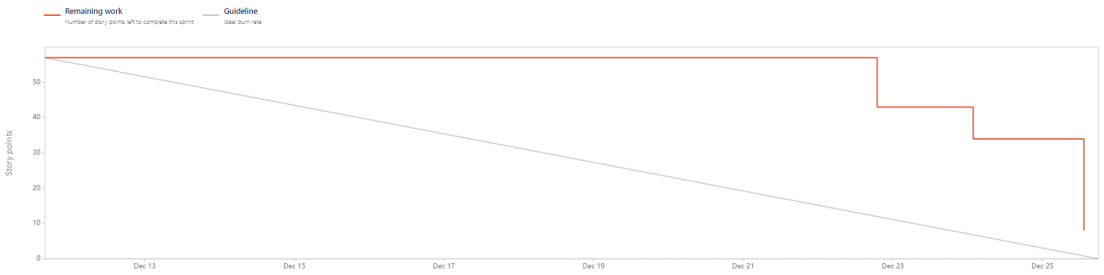


Figure 3.37.: Sprint 5 Burn Down Chart

in the burn down chart.

3.8. Scrum Adaptionen

As part of Project 1, we have adjusted Scrum in order to use it in the best possible way. The adjustments are explained in this chapter.

3.8.1. Definition of Ready (DOR)

Our DOR includes conditions that ensure that all team members understand the user stories and know when a user story can be included in a sprint. The DOR was set in line with the INVEST[2] criteria. A user story in the product backlog must meet the DOR before it can be included in a sprint.

Definition of Ready

- Ensure a clear definition
- Define the functionality or requirement to be implemented
- Clearly defined and testable acceptance criteria
- Ensure there are no or minimal dependencies
- Understood by the whole team
- The user story has been estimated
- The scope of the user story is small enough that it can be implemented in a single sprint.

3.8.2. Definition of Done (DOD)

Our DOD contains all the characteristics and standards that a user story must meet to be considered complete. Once it satisfies the necessary quality requirements (acceptance criteria), the story can be considered complete and can be closed. The goal of our DOD is to create transparency so that everyone has a common understanding of when a story can be closed. A story that does not comply with the DOD may not be finalised.

Definition of Done

- Coding standards and best practices are implemented
- Unit tests for the feature are written and passe
- Any changes to the code or functionality are documented
- The code and functionality are reviewed by peers
- The feature works across multiple platforms

[2] XP123 article: Invest in good stories and smart tasks

- Code is integrated with master branch
- Documentation has been updated
- Acceptance criteria are met

3.8.3. User Story Template

For the creation of a user story, we have defined a template so that the user stories contain all the necessary information. Below is a screenshot of our template. It includes all the relevant fields for us: Assignee, Priority, Business Value, Story Points estimate and assigned Sprint. Furthermore, we describe the user story in the ”Description” field in the format ”AS A ‹user role› I WANT TO ‹the goal› [SO THAT ‹reason›]” as well as the Acceptance Criteria. To ensure that we always have the DOR and DOD to hand, we also work with the Jira On-the-Fly add-on, which enables us to record both for each user story and tick off the individual points accordingly when they have been completed. This allows us to immediately recognise whether a user story can be included in a sprint and whether a story has been fully completed.

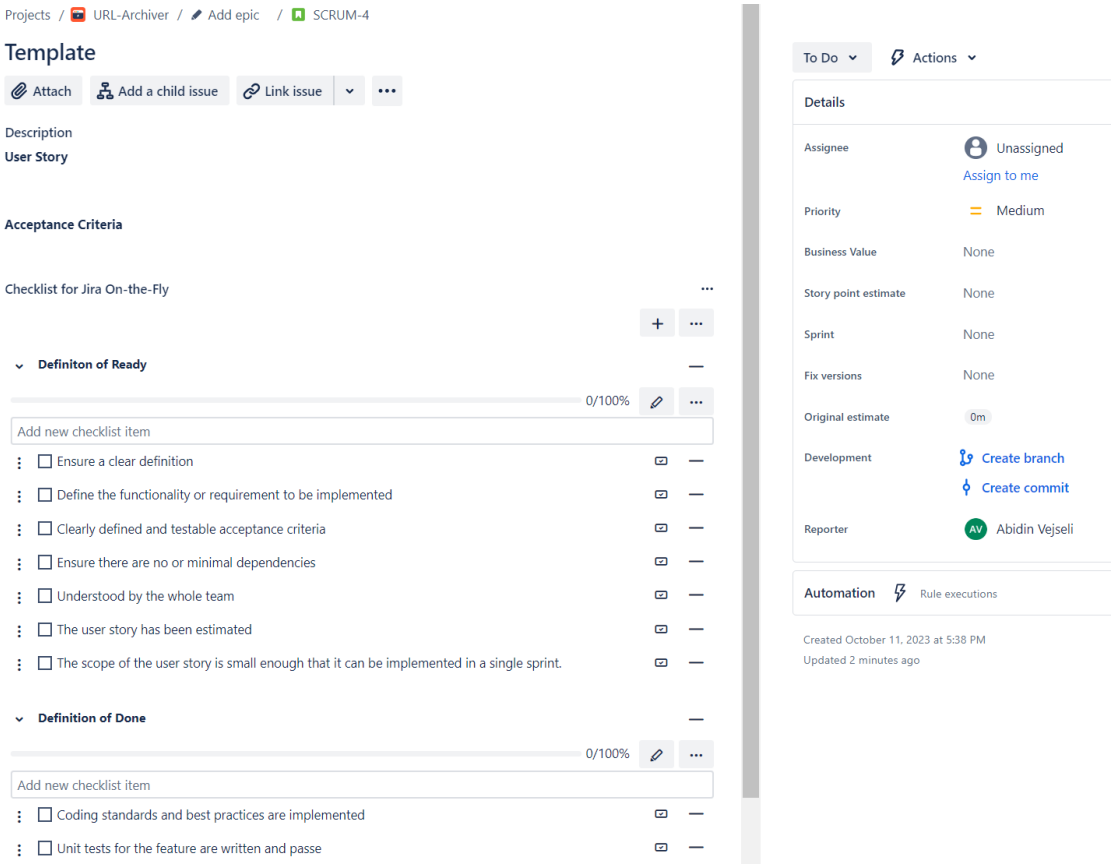


Figure 3.38.: Screenshot from the user story template.

3.8.4. Estimation method

We have chosen the ”T-shirt sizes” method because it is a simple way to estimate effort (story points). This method is based on the fact that everyone knows T-shirt sizes and that large sizes mean more work than small sizes. As a result, this method enables us to make efficient estimations, despite the lack of shared experience in the team.

Below is the scale we use:

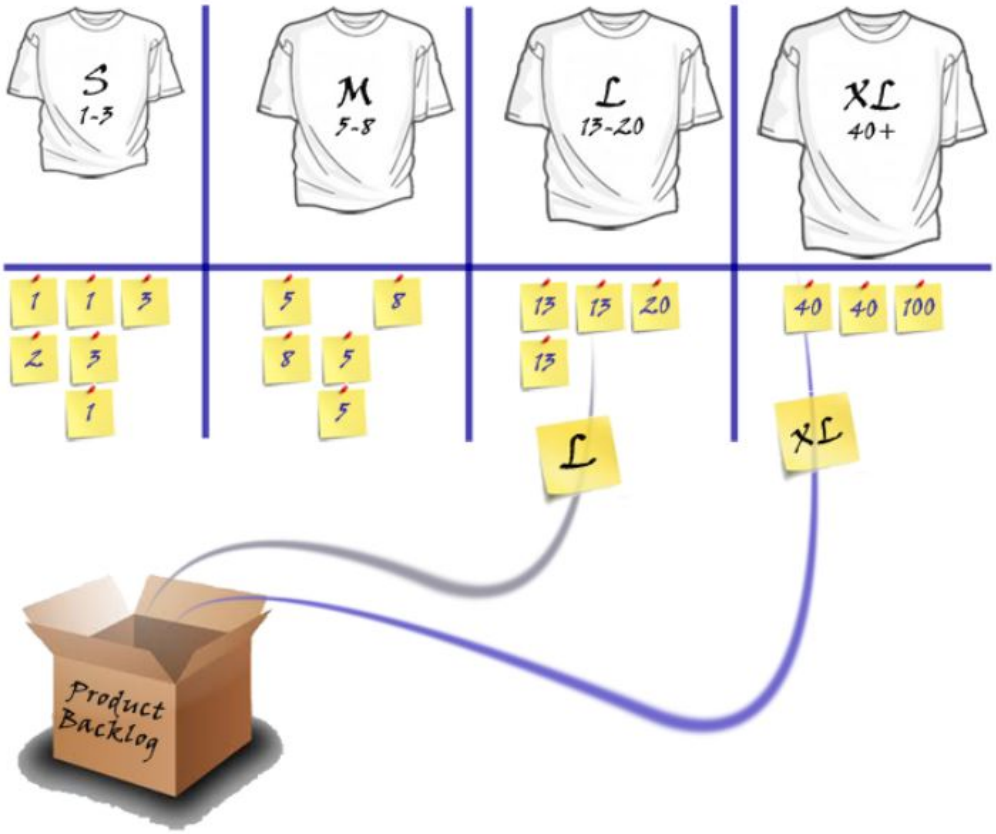


Figure 3.39.: Ilustration of T-Shirt Sizes

3.8.5. Velocity

To select the user stories and tasks to be worked on, a suitable criterion, velocity, is applied to estimate what can be completed in the upcoming sprint.

We have decided that we have a velocity of 30 story points per sprint. Therefore, our workload per sprint should not exceed this threshold. We consciously take this into account during sprint planning.

3.8.6. Sprint

As a team, we have decided that our sprints will take place at two-week intervals. For each sprint we define a SMART sprint goal, which specifies the relevant user stories.

We decided in favour of the two-week rhythm because regular feedback is important to us and thus creates a greater learning effect. Additionally, we ascertained that one-week sprints would result in excessive overheads due to the administrative work involved in Scrum. Likewise, we consider sprints longer than two weeks to be impractical, as the interaction would suffer.

Sprint Planning

As part of sprint planning, we make decisions about which user stories can be implemented based on the sprint goal, the story points and the velocity. Before a user story can be included in the sprint, it must be estimated by the Scrum team. This task is always carried out at the start of our sprint planning. A sprint goal is then defined based on the estimated and prioritised user stories. The user stories we select for the sprint are based on the business value, priority, story points and velocity of the team. The sprint planning takes place on the first Monday of each sprint. We have decided not to have a second sprint planning as we have already defined in our DOR that the user stories should be as small as possible. In addition, each developer has the opportunity to divide their user stories into tasks within the sprint. This allows a better overview of the progress in the sprint.

Daily Scrum

As a team, we have decided not to have daily Scrum meetings, as this is not possible because all team members do work part times. Instead, we have two weekly meetings (weeklys), on Wednesday and Friday at 17:00, which last a maximum of 15 minutes. In addition, we have chosen to hold the meetings through Microsoft Teams as it is easier to organise. The goal of these meetings is to share the current progress, address issues, update the team and briefly discuss the next steps.

Sprint Review

In the sprint review, we check the intermediate result of the processed user stories. We check whether all the stories that should be completed meet the DOD. Furthermore, we discuss in the team what went well, what problems we encountered and how we solved them. Based on these results, the product increment is created. In our case, the product owner, who represents the customer, tests the product increment against the requirements. The review is done from the customer's perspective by testing the product increment. The outcomes are utilized to update the product backlog. The sprint review meeting takes place on the last day of the sprint.

Sprint Retrospective

In the sprint retrospective, we gather information as a team about what went well and what didn't go as planned in the previous sprint. We then derive specific improvements and plan their implementation. Our goal is to improve the efficiency, quality, communication and speed within our team. To achieve this, we give ourselves constructive criticism and are open to feedback. The sprint retrospective takes place on the last day of the sprint.

4. Deployment/Integration

4.1. Licensing and Compliance

4.1.1. Licensing Overview

The project, along with its original source code, is licensed under the permissive and open MIT License. This license aligns with the project's goal of accessibility and ease of use, allowing for free use, modification, distribution, and private use of the software.

4.1.2. Compliance with Open Source Licenses

Although the project itself is licensed under the MIT License, it uses several open-source libraries that are subject to their respective licenses. Please refer to Appendix A.1 for more information. It is worth noting that this project employs libraries licensed under the Eclipse Public License v2.0 (EPL-2.0), including JUnit Jupiter API.

Using EPL-2.0 licensed libraries in a MIT-licensed project is compliant with open source licensing standards, as long as certain conditions are met.

1. **Attribution and Notices**: The project includes a 'Licenses and Attributions' section in the README file. This section acknowledges the use of open-source libraries, specifying their licenses and providing due credits. Additionally, a 'NOTICES.txt' file is included in the project's resources, detailing the open-source components used, their licenses, and where to find the full license texts. This approach is a crucial step to respect and recognize the work of open-source contributors and to maintain transparency about the software's composition.
2. **Separation of Licenses**: It is important to clarify that the MIT License applies to the original code developed for this project. In contrast, the libraries used retain their original licenses (EPL-2.0 in the case of JUnit).
3. **No Modification of EPL-2.0 Libraries**: This project uses the EPL-2.0 licensed libraries in their unmodified form. Any modification to such libraries would require adherence to the specific terms and conditions of the EPL-2.0.

4.1.3. Purpose of Compliance

Ensuring compliance with the licensing terms of used libraries is not only a legal requirement but also a commitment to the open-source community's ethical standards. It ensures that the project respects the rights and efforts of other developers and contributes to the sustainable and responsible use of open-source software.

4.2. Installation (Sysadmin) Manual & Script

The URL-Archiver enables the extraction of URLs from any Unicode text or PDF file and allows for interactive archiving on one of the supported archiving services.

The application was designed to be platform-independent . However, it has only been tested on the following systems, so it cannot be guaranteed to work without restrictions on other platforms.

- Windows 11 (Version 23H2)
- Windows 10 (Version 22H2)
- macOS (Sonoma)
- Ubuntu (20.04.3 LTS)

4.2.1. Requirements

To build and start the application, ensure that the following dependencies are installed on your system:

- Git: Latest stable version recommended.
- Maven: Version 3.8 or higher.
- Java: Version 21.

4.2.2. Clone the repository

To clone the repository, run the following command in a terminal:

```
git clone https://github.com/devobern/URL-Archiver.git
```

4.2.3. Build and run scripts

The build and run scripts are provided for Windows (`build.ps1`, `run.ps1`, `build_and_run.ps1`), Linux, and MacOS (`build.sh`, `run.sh`, `build_and_run.sh`). The scripts are located in the root directory of the project.

The scripts need to be executable. To make them executable, run the following command in a terminal:

- Linux / MacOS: `chmod +x build.sh run.sh build_and_run.sh`
- Windows:
  - Open PowerShell as an Administrator.
  - Check the current execution policy by running: Get-ExecutionPolicy.
  - If the policy is Restricted, change it to RemoteSigned to allow local scripts to run. Execute: Set-ExecutionPolicy RemoteSigned.
  - Confirm the change when prompted.
  - This change allows you to run PowerShell scripts that are written on your local machine. **Be sure to only run scripts from trusted sources.**

Windows

Build the application

To build the application, open a command prompt and run the following script:

```
./build.ps1
```

Run the application

To run the application, open a command prompt and run the following script:

```
./run.ps1
```

Build and run the application

To build and run the application, open a command prompt and run the following script:

```
./build_and_run.ps1
```

Linux and macOS

Build the application

To build the application, open a command prompt and run the following script:

```
./build.sh
```

Run the application

To run the application, open a command prompt and run the following script:

```
./run.sh
```

Build and run the application

To build and run the application, open a command prompt and run the following script:

```
./build_and_run.sh
```

4.3. User Manual

To follow the instructions in this section, the application must be built. See 4.2.

The URL-Archiver is a user-friendly application designed for extracting and archiving URLs from text and PDF files. Its intuitive interface requires minimal user input and ensures efficient management of URLs.

4.3.1. Getting Started

Windows

Open Command Prompt, navigate to the application's directory, and execute:

```
./run.ps1
```

Linux / MacOS

Open Terminal, navigate to the application's directory, and run:

```
./run.sh
```

4.3.2. Operating Instructions

Upon launch, provide a path to a text or PDF file, or a directory containing such files. The application will process and display URLs sequentially.

Navigation

Use the following keys to navigate through the application:

- **o**: Open the current URL in the default web browser.
- **a**: Access the Archive Menu to archive the URL.
- **s**: Show a list of previously archived URLs.
- **u**: Update and view pending archive jobs.
- **n**: Navigate to the next URL.

- **q**: Quit the application.
- **c**: Change application settings.
- **h**: Access the Help Menu for assistance.

Archiving URLs

Choose between archiving to Wayback Machine, Archive.today, both services, or canceling.

When opting to use Archive.today for archiving, an automated browser session will initiate, requiring you to complete a captcha. Once resolved, the URL is archived, and the corresponding archived version is then collected and stored within the application.

Configuration

Customize Access/Secret Keys and the default browser. Current settings are shown with default values in brackets.

Exiting

To exit, press **q**. If a BibTex file was provided, you'll be prompted to save the archived URLs in the BibTex file. Otherwise, or after saving the URLs in the BibTex file, you'll be prompted to save the archived URLs in a CSV file.

For BibTex entries:

- Without an existing note field, URLs are added as: `note = {Archived Versions: \url{url1}, \url{url2}}`
- With a note field, they're appended as: `note = {<current note>, Archived Versions: \url{url1}, \url{url2}}`

5. Conclusion

5.1. Discussion

5.1.1. Example from BFH Template - Delete

What is the significance of your results? – the final major section of text in the paper. The Discussion commonly features a summary of the results that were obtained in the study, describes how those results address the topic under investigation and/or the issues that the research was designed to address, and may expand upon the implications of those findings. Limitations and directions for future research are also commonly addressed.

5.2. Bottom Line

5.3. Future Work

Glossary

Archive Today **A web archiving service that stores snapshots of web pages** for preservation and retrieval. 2

BibTeX **Program for the creation of bibliographical references** and directories in TEXor LATEX documents 2

CLI A **Command Line Interface (CLI)** is a type of user interface navigated entirely using text commands. It allows users to interact with software or operating systems by typing commands into a console or terminal. 2

CSV **CSV (Comma-Separated Values)** is a file format used to store tabular data, such as a spreadsheet or database. Data fields in CSV files are separated using commas. 2

Factory Pattern A **design pattern in software development** used to create objects, allowing interfaces to define object creation but letting subclasses alter the type of objects that will be created. 4

FLOSS **Free/Libre and Open Source Software (FLOSS)** refers to software that is both free in the sense of freedom (libre) and open source. It allows users the freedom to run, study, modify, and distribute the software for any purpose. 2

GUI GUI, or **Graphical User Interface**, refers to a type of user interface that allows users to interact with electronic devices through graphical icons and visual indicators, as opposed to text-based interfaces, typed command labels, or text navigation. It simplifies the user experience by employing graphical representations of the commands and actions available. 4

Model-View-Controller (MVC) pattern Model-View-Controller, a **software design pattern** for implementing user interfaces, data, and controlling logic. 4

Unicode Text File A **Unicode Text File is a file that uses the Unicode standard** for encoding its content. Unicode enables the representation of a vast array of characters from various scripts and symbol sets, making it suitable for internationalization and localization. 2

URL A **URL (Uniform Resource Locator)** is an internet address that directs to a specific resource, like a webpage. 2

URL-Archiver **A software tool designed to extract, archive, and manage URLs found within digital documents.** It supports various file formats and integrates with services like the Wayback Machine for archiving web pages. 2

Wayback Machine **A digital archive of the World Wide Web**, allowing users to see older versions of web pages. 2

Index

Factory Pattern, 4, 5

internet connection, 4

MVC-Pattern, 4–6

platform-independent, 4, 44
priorities, 2
product goal, 2

Bibliography

A. Original Project Description

```
# URL-Archiver

## Description

The goal of this project is to deliver a FLOSS-licensed,
platform-independent Java-program (called "URL-Archiver") that
(1) takes as input (the path of) a directory or any Unicode-text-
    (e.g.: .BIB, .TEX; .HTML; etc.) or .PDF-file
     (https://www.baeldung.com/java-curl);
(2) scans it for any URLs
 (https://stackoverflow.com/questions/4026614/extract-text-from-pdf-files ,
    https://librepdf.github.io/OpenPDF , https://pdfbox.apache.org ; see
     also
    https://en.wikipedia.org/wiki/List_of_PDF_software);
(3) extracts all URLs (regular expression ;-) from the text;
(4) optionally spring-loads all URLs in a Web-browser;
(5) posts all URLs to https://archive.ph ;
(6) gets the resulting archived URLs;
(7) outputs a CSV-file of the resulting key-value (URL, archived URL)
 pairs; and
(8) optionally inserts the archived URLs into a .BIB-file.

The program code should be minimal, modular, and self-explaining.

The project report should be concise (maximally informative, minimally
 long).
It must contain this project description as a quotation.

## Technologies

Java, LaTeX

## Advisor

Dr. Simon Kramer
```

A.1. List of Used Libraries and Their Licenses

Below is the list of libraries used in the project, along with a short description and their versions.

Library	Version	Short Description	Used License
JUnit Jupiter API	5.9.2	Unit testing framework for Java applications.	Eclipse Public License v2.0
JUnit Jupiter Engine	5.9.2	The test engine for running JUnit tests.	Eclipse Public License v2.0
Selenium Java	4.15.0	Automation framework for web applications testing.	Apache-2.0
Selenium Logger	2.3.0	A wrapper for enhanced Selenium log management.	MIT
Mockito Core	5.4.0	Mocking framework for unit tests in Java.	MIT
Mockito JUnit Jupiter	5.4.0	Integration of Mockito with JUnit Jupiter.	MIT
System Lambda	1.2.1	Utilities for testing Java code that uses system properties and environment variables.	MIT
Apache PDFBox	3.0.0	Library for creating and manipulating PDF documents.	Apache-2.0
Jackson Core	2.16.0	Core part of Jackson that defines common low-level features.	Apache-2.0
Jackson Dataformat XML	2.15.2	Support for reading and writing XML encoded data via Jackson abstractions.	Apache-2.0

Table A.1.: List of Used Libraries in the Project

Declaration of Authorship

I hereby declare that I have written this thesis independently and have not used any sources or aids other than those acknowledged.

All statements taken from other writings, either literally or in essence, have been marked as such.

I hereby agree that the present work may be reviewed in electronic form using appropriate software.

January 2, 2024

N. Dora

A. Vejseli

K. Wampfler