

Nathan submitted

by Zack Evans

Submission date: 06-Jan-2026 09:52AM (UTC+0000)

Submission ID: 272242838

File name: 18253_Zack_Evans_Nathan_submitted_694650_42805828.docx (1.82M)

Word count: 4322

Character count: 25622



ACCA7023 - Agile Development and Software Dev Ops

Assignment – Evaluating Agile and Scrum Practices

Zack Evans
1908686@student.uwtsd.ac.uk

Table of Contents

ACCA7023 - Agile Development and Software Dev Ops	1
1. Introduction	4
2. Overview of Agile Principles	5
2.1 Face-to-Face Communication and Collaboration.....	5
2.1.1 Scrums Implementation.....	5
2.1.2 Sprint Planning	5
2.1.3 Sprint Review.....	5
2.1.4 Sprint Retrospective.....	5
2.1.5 Daily Scrum (Daily stand-up)	6
2.2 Multiple Short Iterations	6
2.2.1 Achieving Short Iterations Through Sprints.....	6
2.2.2 User story mapping within iterations	6
2.3 Adaptability to Changing Requirements	7
2.3.1 Scrum promotes change	7
2.4 Quality of Ownership Throughout the Product Lifecycle	8
2.4.1 Scrum Roles and Ownership.....	8
2.4.2 How ownership happens	8
3. Estimating User Story Implementation Time and Difficulty	9
3.1 Purpose of Estimation	9
3.2 Process of Estimating.....	9
3.3 Estimation Accuracy and Challenges	10
3.4 Burndown charts.....	10
3.5 Relationship Between Estimation and the Burndown Chart.....	10
4. Web Application Testing	12
4.1 Main Components for Testing	12
4.1.1 Frontend	12
4.1.2 Backend Logic	12
4.1.3 Backend API	13
4.2 Github Actions	13
4.2.1 How GitHub Actions Fits in CI/CD	13
4.2.2 GitHub Actions Example.....	13

4.3 API Testing with Postman.....	14
4.3.1 Postman Automated Testing.....	14
4.3.2 Postman Testing Example	14
4.4 User Interface Testing with Selenium.....	14
4.4.1 How it works.....	14
4.5 Benefits and Limitations of Adopting SwDevOps	15
4.5.1 Limitations of SwDevOps	15
5. Conclusion	17
6. References	18
7. Figures	19

1. Introduction

Software development demands methodologies capable of responding effectively to rapidly changing environments, strict delivery deadlines, and heightened expectations for reliability. Traditional software development strategies often struggle in these situations, leading to delays, inadequate stakeholder feedback, and diminished product quality. Consequently, Agile methodologies and their supporting practices have gained widespread adoption within the software industry.

This report examines key Agile principles and their practical application through the Scrum framework, emphasising how collaboration, short iterative development cycles, adaptability to change, and shared ownership contribute to successful project outcomes. Additionally, the report discusses the importance of user story estimation and the use of tools such as burndown charts to support planning, transparency, and continuous improvement.

Furthermore, the report investigates web application testing within Agile and DevOps contexts, focusing on frontend, backend, and API testing, and the integration of modern automation tools such as GitHub Actions, Postman, and Selenium. Finally, it evaluates the benefits and limitations of adopting Software DevOps (SwDevOps) practices, offering a balanced perspective on how automation, continuous integration, and continuous deployment can enhance software quality while introducing new challenges.

2. Overview of Agile Principles

Agile development is founded on a set of principles [1] intended to increase flexibility, collaboration, and product quality in complex projects and environments. This section outlines four key Agile concepts.

2.1 Face-to-Face Communication and Collaboration

One core principle is the strong emphasis on direct communication and collaboration among team members and stakeholders. Principle 6: “The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.” [1]. This enables practices like daily stand-ups, pair programming, and other collaborative review-based meetings, enabling progress to be measured and faster feedback and decision-making.

2.1.1 Scrums Implementation

In Scrum, face-to-face communication and collaboration are regarded as key for effective teamwork. While Agile emphasises interpersonal interaction, Scrum provides specific structures and events to aid this process [2]:

- Daily Scrum
- Sprint Planning
- Sprint Review
- Sprint Retrospective

2.1.2 Sprint Planning

The main goal of sprint planning is to organise work for the upcoming sprint. This event enables the Scrum team to discuss priorities, goals, and tasks. All members contribute to decomposing tasks into user stories, which are then sized and ordered by priority to facilitate active discussion and negotiation. For example, in the case of a note-taking application, features were divided into individual user stories [F1].

2.1.3 Sprint Review

The sprint review enables the entire team to inspect completed work and collect feedback from users and stakeholders. Immediate feedback supports reprioritisation and the development of new user stories, which can be added to the product backlog.

2.1.4 Sprint Retrospective

This allows the team to discuss on:

- What went well
- What did not go so well
- And how to improve

Zack Evans
1908686@student.uwtsd.ac.uk

This encourages honest dialogue, trust building and ownership.

2.1.5 Daily Scrum (Daily stand-up)

Scrum teams meet daily (preferably in person). This allows them to discuss completed and planned tasks for the day. This allows the team to assess the progress of tasks and identify potential obstacles. Due to the direct nature of these meetings, they improve the efficiency of problem-solving.

2.2 Multiple Short Iterations

A fundamental principle of Agile development is the delivery of software through short iterations. This is typically done by developing and testing sections of work. As stated in the Agile Manifesto's first principle, "Our highest priority is to satisfy the customer through early and continuous delivery of valuable software" [1]. The emphasis on "continuous" delivery enables Agile teams to operate in iterative cycles, producing work at regular intervals.

2.2.1 Achieving Short Iterations Through Sprints

In Scrum, short iterations of work done are classed as Sprints. These are fixed-length timeboxes intended to deliver a defined set of tasks. Sprints offer several advantages:

- Faster feedback from users and stakeholders
- Early detection of risks and issues
- Greater adaptability to changing requirements
- Continuous improvement

2.2.2 User story mapping within iterations

User story mapping is a visual technique for organising user stories. Typically, the horizontal axis represents the user journey or workflow, while the vertical axis indicates the priority or depth of functionality.

An example of user story organisation is provided in [F1], which is based on a note-taking application. Feature sections are mapped alongside their respective priorities and depths. This mapping can be divided into individual interactions or sprints by segmenting user stories. Horizontal splitting enables the creation of a minimum viable product in the first sprint [3], allowing subsequent sprints to achieve specific goals and milestones. This approach minimises overlooked user stories and mitigates feature creep.

Applying this approach to the original mapping results in three distinct sprints [F2]. The initial sprint delivers a basic note-creation system, while subsequent sprints introduce editing, deletion, and management functionality.

2.3 Adaptability to Changing Requirements

According to principle 2 of the Agile Manifesto. Agile allows the ability to adapt to changing requirements, even if the development lifecycle is in its later stages. Rather than viewing change as a disruption, Agile treats it as a potential source of the “customer's competitive advantage” [1]. By maintaining dynamic work priorities and continuous development, teams can better align products with stakeholder needs.

2.3.1 Scrum promotes change

Scrum expects change. In fact, it assumes that requirements, priorities, and even understanding of the product will evolve during development. Instead of resisting change, Scrum embraces it.

- **Iterative Delivery:** Work is delivered in short cycles known as Sprints, typically lasting two to four weeks. Each Sprint results in a potentially shippable product increment, enabling the team to respond rapidly to new information or feedback.
- **Inspection and Adaptation:** Scrum emphasises frequent inspection through events such as the Sprint Review and Daily Scrum, allowing the team to identify issues early and implement necessary adjustments.

Practice	How it Helps Manage Change
Product Backlog	The backlog is dynamic, meaning new requirements can be added, priorities changed, or items removed at any time. The Product Owner continuously refines the backlog.
Sprint Planning	At the start of each Sprint, the team selects the most valuable items from the backlog. This allows flexibility teams to shift focus as priorities evolve.
Daily Scrum	A daily check-in helps the team adapt their plan for the day based on progress and new insights.
Sprint Review	Stakeholders review the increment and provide feedback. The team can then adapt the backlog and plan future work accordingly.
Sprint Retrospective	The team reflects on what went well and what didn't. This continuous improvement loop makes the team more adaptable over time.

2.4 Quality of Ownership Throughout the Product Lifecycle

Agile also promotes collective ownership, which gives individuals responsibility for producing a high-quality product. Using practices like daily standups, code reviews, and short iterations. It helps to promote ownership of tasks throughout the scrum team.

2.4.1 Scrum Roles and Ownership

Scrum likes to break down jobs and ownership into roles [4].

Role	Responsibility
Product Owner	- Owns the product vision and ensures features align with user needs. - Responsible for prioritising the backlog by value not urgency. - Ensures the team focuses on delivering a high-quality product
Scrum Team	- Assigns themselves current tasks from the sprint - Decides on how to implement tasks
Scrum Master	- Pushes team to the correct direction - Fosters collaboration within team - Promotes continuous improvement

2.4.2 How ownership happens

Typically, the stakeholder provides basic user stories that outline the features they want. While everyone else participates in the refinement process, breaking user stories into multiple stories that hold the individual details needed. These refined stories are then allocated to individual sprints (according to priorities), assigning ownership to the Scrum Master and development team.

When the sprints start, team members pick the stories they want to work on. Members of the team take the responsibility of implementing the user stories within an agreed-upon acceptance criteria. Quality practices (like code reviews and peer feedback) ensure the team owns the end-to-end quality of each user story.

Upon completion of a sprint, the Scrum team demonstrates the completed work to stakeholders during the sprint review. Feedback is collected, and the team is then responsible for adapting future work based on this input to ensure continuous improvement.

3. Estimating User Story Implementation Time and Difficulty

In Agile development, estimating the time and difficulty of user stories is critical for planning, prioritisation, and progress tracking.

3.1 Purpose of Estimation

The purpose of user story estimation is to support planning and decision-making within an Agile process. Estimation allows teams to determine how much work can realistically be completed within a sprint, ensuring a measurable workload which can be achievable. By assigning an estimate based on the level of effort to each user story, teams can reduce the risk of over-commitment and overflow of user stories to the next sprint.

Estimation also plays an important role in managing stakeholders' expectations. While Agile methodologies like Scrum accept that change is likely, having estimated values provides a baseline for communicating progress. This helps everyone understand the timeline for which tasks will be completed.

However, an estimation is not a guarantee when it comes to timelines. Instead, it provides an informed approximation based on shared knowledge, experience, and assumptions. As a result, estimates should be viewed as planning aids rather than fixed deadlines. When used in the right context, it enables teams to adapt to change and improve their accuracy in completing work on time.

3.2 Process of Estimating

When it comes to estimating, time and difficulty typically need to be included. Teams need to gather their understanding of a user story and share their unified opinion on its complexity. Factors that could affect this include user story complexity, related dependencies, and potential missed risks.

Teams commonly use techniques like story points to assess effort rather than fixed time values [5]. Techniques such as Planning Poker or T-Shirt sizing are used to support this process.

Once estimates have been agreed, they are used to plan sprint capacity and determine which user stories can be realistically completed in each sprint. As work progresses, actual completion data is compared with original estimates. This feedback allows teams to refine future estimates and improve accuracy over time. However, the process remains inherently uncertain, as changes in requirements or unexpected technical issues can still impact implementation time and difficulty.

3.3 Estimation Accuracy and Challenges

Although estimation is essential for planning, achieving a high level of accuracy is often difficult. One of such challenges is uncertainty. This is due to not all technical details or possible problems being known in advance [6]. This can lead to underestimation, particularly when teams are overly optimistic about how quickly tasks can be completed.

One more challenge is the impact of changing requirements. In Agile projects, feature creep can occur, leading to additional user stories and increased project complexity. Such feature creep could be new features being proposed, third-party systems being needed, or delayed input from stakeholders.

Despite such challenges, estimation is still valuable when used appropriately. Accuracy tends to improve over time as teams gain experience and use data from previous sprints to direct future estimates. Regular reviews, such as sprint retrospectives, allow teams to identify patterns in estimation errors and change their approach.

3.4 Burndown charts

The Burndown chart [F3] is an Agile project management tool that visually tracks the amount of work remaining in a sprint over time. It displays remaining effort, often measured in story points or hours, against the sprint timeline. This allows both the development team and stakeholders to quickly assess whether progress is on track.

The primary purpose of the Burndown chart is to help assess the current sprint's progress. By frequently updating the chart, scrum masters can check if the team is current within or outside the sprint timebox. Additionally, the Burndown chart provides feedback on the accuracy of initial estimates. If the chart consistently shows slower progress than planned, it may indicate that user stories were underestimated or that unexpected complexity was encountered. As a result, Burndown charts are not only useful for tracking progress but also for improving future estimation and planning practices [7].

3.5 Relationship Between Estimation and the Burndown Chart

The effectiveness of a Burndown chart is closely linked to the accuracy of user story estimates. When story points are estimated realistically, the Burndown chart provides a meaningful representation of progress and remaining work. It allows the team to see whether they are on track to complete the sprint, supporting decisions about workload alterations or reprioritisation.

However, "The effectiveness of a burndown chart heavily relies on the accuracy of initial time or effort estimates. If these estimates are off, the chart can misrepresent the team's actual progress. Overestimations may suggest the team is ahead of schedule,

Zack Evans
1908686@student.uwtsd.ac.uk

while underestimations can indicate delays, even if the team's performance is consistent.” [8]. This affirms the importance of using estimation not as a fixed prediction but as a guide that can be used with team experience and historical data.

Burndown charts also support continuous improvement in estimation. By comparing the planned effort with actual progress, teams can spot recurring themes in under ([F4]) or overestimation and improve their methods for future sprints. This feedback loop strengthens both planning and monitoring, making sure that estimation and Burndown tracking work together to improve the overall effectiveness of Agile project management.

4. Web Application Testing

Web applications have become a fundamental part of software systems, supporting a wide range of everyday activities, including communication, productivity, and information management. One common example of such systems is a web-based To-Do List application. The To-Do List application being used within this report allows users to create, view, and delete tasks through a web interface. Despite its apparent simplicity, this type of application relies on multiple interconnecting components, making it a suitable case study for discussing software testing.

4.1 Main Components for Testing

Testing the web application involves several components:

- Frontend web ui
- Backend logic
- Backend API

4.1.1 Frontend

The frontend is the user-facing side of the web application. As a result, it requires thorough testing because it directly affects user experience (UX) and usability. Key features to test include the note creation form, which must ensure it accepts valid input, submits correctly, and displays new notes instantly. The ability to check off To-Do notes also needs testing to confirm that task completion updates the UI consistently and provides clear visual feedback. Additionally, the delete note functionality must be tested to ensure the correct note is removed without unintended behavior. Input validation is another critical aspect, as the frontend should properly handle empty or invalid entries with clear error messages. Finally, frontend logic that is not directly visible, such as state management and conditional rendering, must be tested to prevent bugs and ensures that it runs smooth, reliable interactions throughout the application.

4.1.2 Backend Logic

The backend of the To-Do application is responsible for handling logic, database interactions, and API communication, making it another crucial component to test. Backend testing confirms that To-Do items are correctly created, retrieved, and deleted within the database. This consists of verifying that data is stored accurately, changes to task status (such as checking off a To-Do) are properly saved, and deleted items are permanently removed.

4.1.3 Backend API

API endpoints must also be tested to confirm they return correct responses and error handling. Thorough testing is needed to help ensure data integrity, reliability, and smooth interaction between the server, database, and frontend of the application.

4.2 Github Actions

GitHub Actions is a GitHub feature that allows developers to automate workflows directly within a GitHub repository. It is commonly used in CI/CD (Continuous Integration and Continuous Deployment) pipelines to automatically run tasks when specific events occur, such as pushing code, creating a pull request, or merging changes [9].

In a CI/CD pipeline, GitHub Actions helps automate testing, building, and deployment of code. This makes sure that new code changes are automatically checked for errors, compiled or packaged correctly, and delivered to the appropriate environment without manual effort.

4.2.1 How GitHub Actions Fits in CI/CD

Continuous Integration (CI) focuses on frequently integrating code changes into a shared repository and verifying them through automated testing. GitHub Actions supports CI by running workflows every time code is pushed, or a pull request is created [9]. These workflows can execute unit tests and integration tests to ensure that new changes do not break existing functionality. By detecting issues early, developers can fix bugs quickly and maintain code quality.

Continuous Deployment (CD) extends CI by automatically deploying code after it successfully passes all tests. With GitHub Actions, once the CI phase completes without errors, the workflow can proceed to deploy the application to a production environment. For instance, after a pull request is merged into the main branch, GitHub Actions can deploy the latest version of a web application to cloud platforms such as AWS or Azure [9]. This ensures that users always have access to the most up-to-date version of the software.

4.2.2 GitHub Actions Example

Using GitHub as a reference [10], we can create an actions workflow file [F5]. In this workflow, GitHub Actions automatically runs tests whenever code is pushed to the main branch. If the tests pass, the deployment job runs next. This simple setup shows how GitHub Actions can efficiently manage testing and deployment, making it a necessary tool for modern CI/CD pipelines.

Zack Evans
1908686@student.uwtsd.ac.uk

4.3 API Testing with Postman

Postman is a widely used tool for testing Application Programming Interfaces (APIs). It provides a user-friendly interface that allows developers and testers to send HTTP requests and analyse responses without writing code. Using Postman, users can easily interact with API endpoints by specifying request types (GET, POST, PUT, or DELETE) and supplying parameters, headers, and request bodies. This makes it an easy-to-use tool for verifying that APIs work as expected during development and testing. For example, Postman can be used to test the To-Do web application, by testing the create note feature to ensure that it correctly creates the note and returns it [F6].

4.3.1 Postman Automated Testing

Postman also supports automated API testing through built-in scripting features. “You can write tests for your Postman API requests in JavaScript in the Post-response tab. You can also use test code to aid the debugging process when something goes wrong with your API project. For example, you might write a test to validate your API’s error handling by sending a request with incomplete data or wrong parameters.” [11], Postman can be used to test the POST /todos endpoint by sending mock transaction data and verifying whether the API returns a success or failure message based on the input, and if it does, its transaction within an acceptable timeframe.

4.3.2 Postman Testing Example

Using Postman, we can test out the create To-Do note endpoint. To do this, a request was created, which followed the same rules as the backend server. With this, it allows me, as the developer, to see if the API is working without the need for the frontend. This process can even be expanded upon by adding unit tests to the API request [F7]. With this, the results can be output [F8].

4.4 User Interface Testing with Selenium

Selenium is a powerful collection of tools created for emulating browser usage. It is primarily used for user interface (UI) testing by emulating real user actions, such as clicking, filling in forms, or navigating pages. Selenium helps validate the functionality, usability, and reliability of web applications. Its popularity derives from its capacity to automate repetitive testing tasks efficiently, reducing human error and saving time during software development.

4.4.1 How it works

Selenium is fundamental to UI testing through multiple important capabilities:

Zack Evans
1908686@student.uwtsd.ac.uk

- **Test Web UI Components:** Selenium enables automated interaction with various UI components of a web application. This automation ensures that every aspect of the interface behaves correctly under different scenarios, helping detect possible bugs before the software reaches users.
- **Cross-Browser Testing:** One of Selenium's strongest key advantages is its compatibility for multiple web browsers, such as Chrome, Firefox, and Safari. Ensuring that a web application works consistently across all browsers.
- **Supports Multiple Programming Languages:** Test scripts can be written in Java, Python, C#, and other popular languages, allowing flexibility when it comes to implementing for development teams.
- **Integration with CI/CD Pipelines:** Selenium can be integrated into pipelines such as GitHub Actions, allowing automated testing within continuous integration and deployment workflows. This allows tests to run automatically in the background without manual intervention.

4.5 Benefits and Limitations of Adopting SwDevOps

Adopting Software DevOps (SwDevOps) provides several major advantages for development teams and organisations. One of the main benefits is faster development cycles. By implementing continuous integration (CI) and continuous deployment (CD), teams can automatically build, test, and deploy software, reducing the time between writing code and releasing it to production.

Additionally, SwDevOps plays a role in raising code quality. Making sure that issues and errors are caught outside of an individual's machine, and catching them in a controlled environment, reducing the likelihood of critical failures in production. Continuous monitoring and feedback loops support maintaining standards for software reliability.

4.5.1 Limitations of SwDevOps

Although it has many advantages, adopting SwDevOps has some limitations. The first challenge is the complexity of the initial setup. Establishing CI/CD pipelines, automated testing frameworks, and other DevOps practices involves attentive planning and a significant upfront investment of time and effort. For smaller teams or projects, this initial setup can feel overwhelming, especially for inexperienced teams, which can lead to an extra cost of hiring a DevOps engineer.

Another limitation is excessive dependence on automation. While automated tools can catch many issues, they cannot replace manual testing in all scenarios. Exploratory testing, usability evaluations, and other human-driven tests remain vital for recognising outliers and subtle user experience issues.

Finally, there is the cost of tools. Many DevOps solutions, especially those offering advanced scalability, automation, or monitoring, come with subscription fees or

Zack Evans
1908686@student.uwtsd.ac.uk

infrastructure costs. Introducing these tools to large teams or high bandwidth applications can become expensive, particularly for startups or organisations with limited budgets.

Overall, while SwDevOps offers faster releases, better collaboration, and higher code quality, organisations must carefully manage its difficulties to setup, the limits of automation, and associated costs to maximise its benefits.

5. Conclusion

This report has examined the core principles of Agile development and their practical application through the Scrum framework. Key concepts such as face-to-face communication, short iterative development, adaptability to change, and shared ownership were shown to support effective collaboration, transparency, and continuous improvement. Scrum events, including sprint planning, daily stand-ups, reviews, and retrospectives, provide the structure needed to implement these principles in practice, ensuring development stays aligned with stakeholder needs while helping teams respond quickly to feedback and evolving requirements.

The report also highlighted the role of estimation, testing, and DevOps practices in supporting successful Agile delivery. Estimating user stories using techniques such as story points, alongside tools like Burndown charts, helps teams plan realistically and monitor progress while acknowledging uncertainty. In addition, comprehensive testing across the frontend, backend, and APIs, supported by automation tools such as Selenium, Postman, and GitHub Actions, ensures software quality throughout the lifecycle. While adopting SwDevOps introduces challenges such as setup complexity and costs, its benefits in improving collaboration, release speed, and code quality reinforce its value within modern Agile software development.

6. References

- [1] Beck et al, "Principles behind the Agile Manifesto," Agilemanifesto.org, 2001.
<https://agilemanifesto.org/principles.html>
- [2] K. Schwaber and J. Sutherland, "The Scrum Guide," Nov. 2020. Available:
<https://scrumguides.org/docs/scrumguide/v2020/2020-Scrum-Guide-US.pdf>
- [3] J. Patton, P. Economy, M. Fowler, A. Cooper, and M. Cagan, User story mapping : discover the whole story, build the right product. Beijing ; Sebastopol, Ca: O'reilly, 2014.
- [4] D. West, "Agile scrum roles," Atlassian, 2019.
<https://www.atlassian.com/agile/scrum/roles>
- [5] Atlassian, "What are story points in Agile and how do you estimate them?," Atlassian, 2024. <https://www.atlassian.com/agile/project-management/estimation>
- [6] K. Soosalu, "Improving the sprint review and delivery process by enhancing the monitoring system", 2025. <https://dspace.ut.ee/server/api/core/bitstreams/09d87a48-71f5-40cb-b7b0-b6393c378ac3/content>
- [7] B. Maloney, "Burndown chart - Scrum Alliance agile glossary," Scrumalliance.org, 2025. https://www.scrumalliance.org/glossary/burndown-chart?utm_source=chatgpt.com (accessed Jan. 05, 2026).
- [8] P. Griffon, "What Is A Burndown Chart," Purple Griffon, May 09, 2025.
<https://purplegriffon.com/blog/what-is-a-burndown-chart>
- [9] GitHub, "Understanding GitHub Actions - GitHub Docs," GitHub Docs, 2025.
<https://docs.github.com/en/actions/get-started/understand-github-actions>
- [10] "Building and testing Node.js - GitHub Docs," GitHub Docs, 2025.
<https://docs.github.com/en/actions/tutorials/build-and-test-code/nodejs>
- [11] "Write scripts to test API response data in Postman | Postman Learning Center," Postman Learning Center, Aug. 24, 2024. <https://learning.postman.com/docs/tests-and-scripts/write-scripts/test-scripts/>

7. Figures

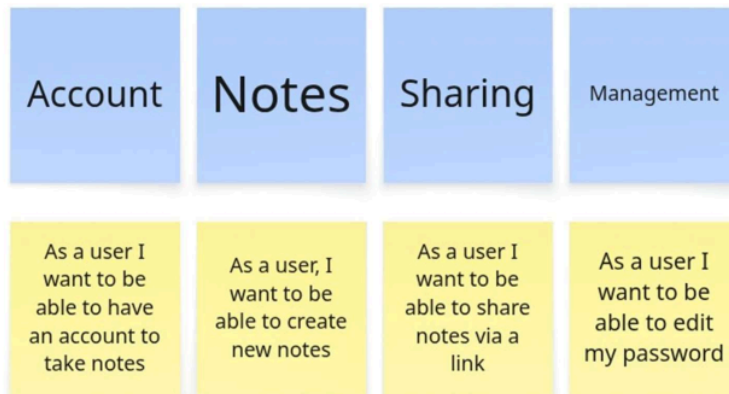
[F1] User story mapping for note taking app



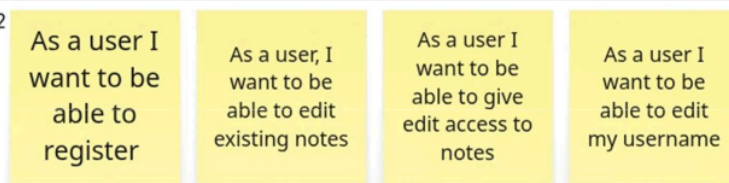
[F2] User story sprints

Zack Evans
1908686@student.uwtsd.ac.uk

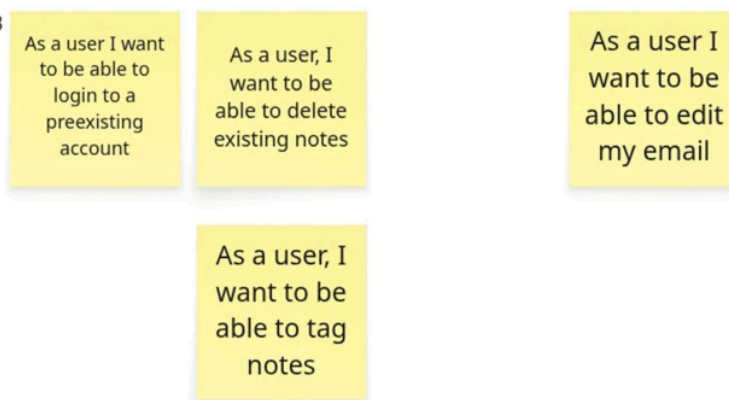
Sprint 1



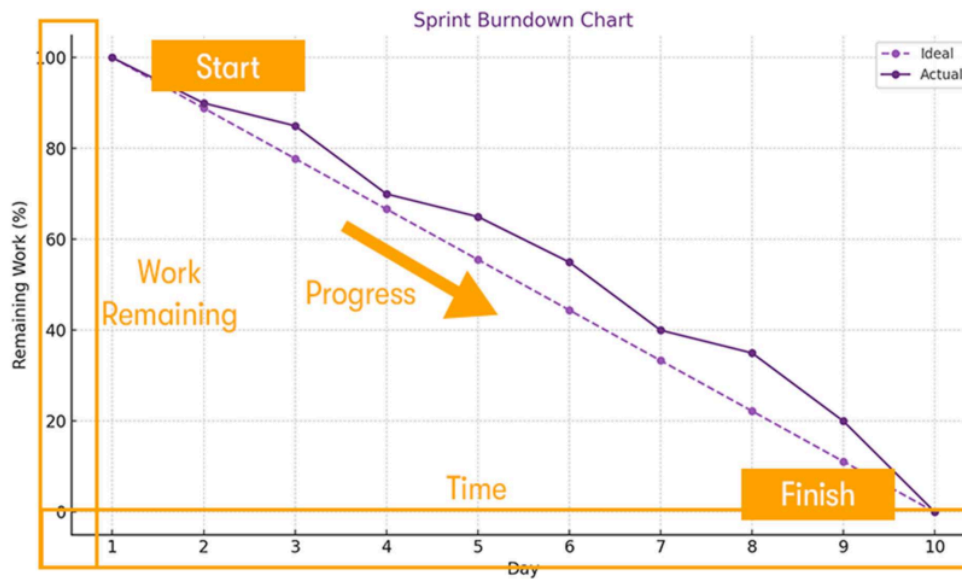
Sprint 2



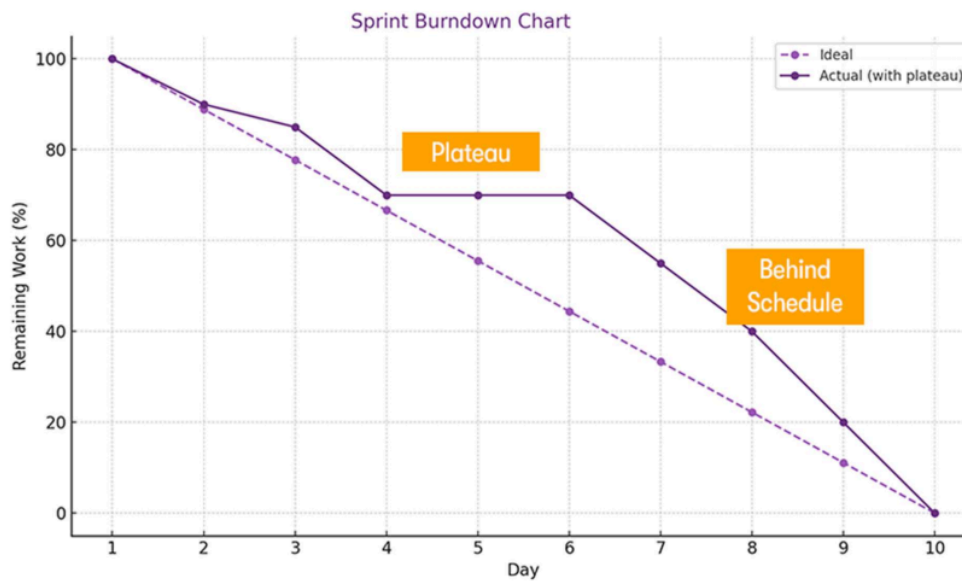
Sprint 3



[F3] Burndown chart example reference [8]



[F4] Burndown chart showing underestimation, example reference [8]



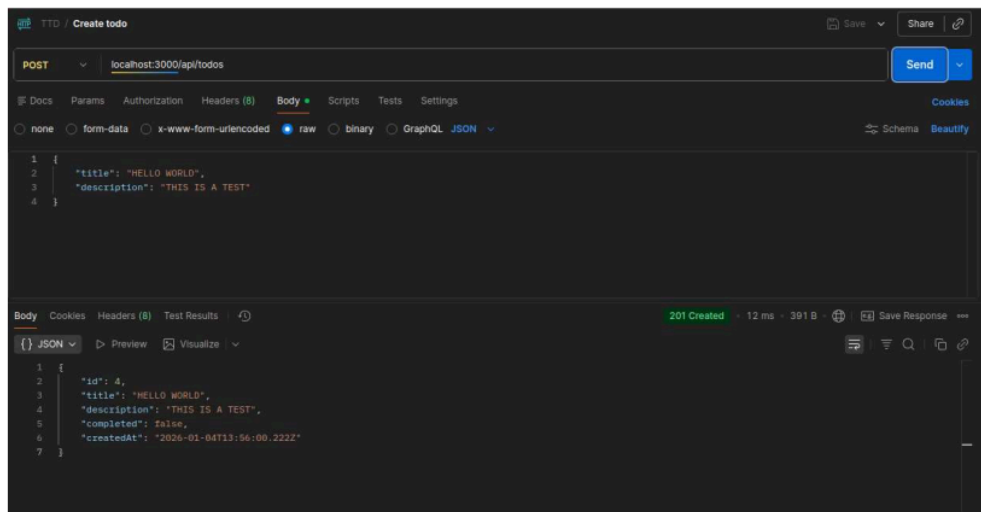
[F5] GitHub action workflow

```

1  name: todo-app-ci
2  run-name: CI test for Todo App
3  on:
4    push:
5      branches:
6        - main
7  jobs:
8    check-bats-version:
9      runs-on: ubuntu-latest
10     steps:
11       - uses: actions/checkout@v5
12       - name: Use Node.js ${{ matrix.node-version }}
13       - uses: actions/setup-node@v4
14         with:
15           node-version: ${{ matrix.node-version }}
16       - run: npm ci
17       - run: npm run build --if-present
18       - run: npm test

```

[F6] Postman create To-Do note



[F7] Postman test script

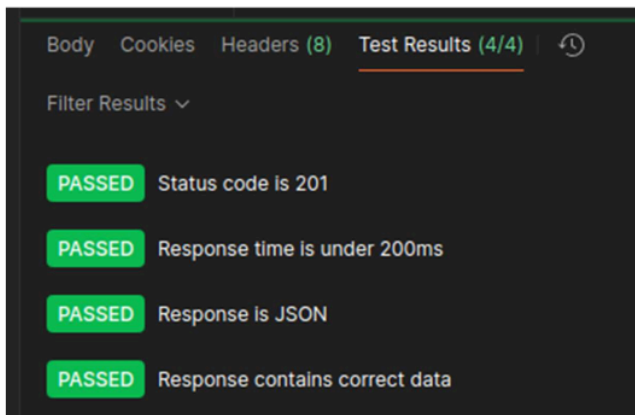
Zack Evans
1908686@student.uwtsd.ac.uk

```

1  pm.test("Status code is 201", function () {
2      pm.response.to.have.status(201);
3  });
4
5  pm.test("Response time is under 200ms", function () {
6      pm.expect(pm.response.responseTime).to.be.below(200);
7  });
8
9  pm.test("Response is JSON", function () {
10     pm.response.to.be.json;
11 });
12
13 /*
14  Example response body:
15  {
16      "id": 6,
17      "title": "HELLO WORLD",
18      "description": "THIS IS A TEST",
19      "completed": false,
20      "createdAt": "2026-01-04T14:01:25.014Z"
21  }
22  */
23 pm.test("Response contains correct data", function () {
24     var jsonData = pm.response.json();
25     pm.expect(jsonData).to.have.property("id");
26     pm.expect(jsonData).to.have.property("title", "HELLO WORLD");
27     pm.expect(jsonData).to.have.property("description", "THIS IS A TEST");
28     pm.expect(jsonData).to.have.property("completed", false);
29     pm.expect(jsonData).to.have.property("createdAt");
30 });

```

[F8] Postman result



Zack Evans
1908686@student.uwtsd.ac.uk

FINAL GRADE

70/100

GENERAL COMMENTS

This submission demonstrates a solid understanding of Agile principles, Scrum methodology, and DevOps practices, with content that is well-organised and logically structured throughout.

The report makes good use of primary sources such as the Agile Manifesto and Scrum Guide, and the consistent application of the To-Do application as a case study helps to ground theoretical concepts in practical context.

Screenshots of practical workshops attended in class would have been beneficial, as it was a requirement.

The Postman section is particularly well-evidenced with supporting screenshots. However, the work is predominantly descriptive rather than critically evaluative, meaning it explains what concepts and tools do without sufficiently analysing their strengths, limitations, or situations where they may be less effective.

The reliance on websites and blogs rather than peer-reviewed academic journals weakens the scholarly engagement, and the benefits and limitations section lacks any supporting citations.

A practical Selenium example or equivalent framework and associated references would have helped corner the discussion on automated testing

Agile Concepts and Scrum Methodology

Section 1 Engagement with Literature 11 out of 16

Section 1 Knowledge and Understanding 13 out of 16

Section 1 Cognitive and Intellectual Skills 5 out of 8

Estimation and Burndown Charts

Section 2 Engagement with Literature 5 out of 8

Section 2 Knowledge and Understanding 7 out of 8

Section 2 Cognitive and Intellectual Skills 3 out of 4

SwDevOps Evaluation

Section 3 Engagement with Literature 9 out of 16

Section 3 Knowledge and Understanding 12 out of 16

Section 3 Cognitive and Intellectual Skills 5 out of 8

PAGE 1

PAGE 2

PAGE 3

PAGE 4

PAGE 5

PAGE 6

PAGE 7

PAGE 8

PAGE 9

PAGE 10

PAGE 11

PAGE 12

PAGE 13

PAGE 14

PAGE 15

PAGE 16

PAGE 17

PAGE 18

PAGE 19

PAGE 20

PAGE 21

PAGE 22

PAGE 23
