

A Comprehensive Analysis of Agile, Scrum and DevOps Practices in Contemporary Software Engineering

by Md Azizul Khan

Submission date: 09-Jan-2026 08:10AM (UTC+0000)

Submission ID: 272633289

File name:

67088_Md_Azizul_Khan_A_Comprehensive_Analysis_of_Agile__Scrum_and_DevOps__Practices_in_Contemporary_Software_Engineerin_108904662.pdf
(664.13K)

Word count: 5424

Character count: 30249



Prifysgol Cymru
Y Drindod Dewi Sant
University of Wales
Trinity Saint David

A Comprehensive Analysis of Agile, Scrum and DevOps
Practices in Contemporary Software Engineering

Submitted by:

MD Azizul Khan

Student ID: 2514708

Course Code: ACCA7023

Programme: MSc in Software Engineering and Artificial Intelligence

Submitted To:

Nathan Morgan

Lecturer, Agile Development and Software Dev Ops

Submission Date: 09 January 2026

Table of Contents

1. Introduction

- 1.1 Background
- 1.2 Purpose of the Report
- 1.3 Structure of the Report

2. Agile Concepts within the Scrum Methodology

- 2.1 Multiple Short Iterations in Scrum
 - 2.1.1 Sprint Planning
 - 2.1.2 Sprint Execution
 - 2.1.3 Sprint Review
 - 2.1.4 Critical Evaluation
- 2.2 Face-to-Face Communication and Collaboration
 - 2.2.1 Daily Scrum
 - 2.2.2 Sprint Review and Retrospective
 - 2.2.3 Collaboration Tools
 - 2.2.4 Critical Evaluation
- 2.3 Adaptability to Changing Requirements
 - 2.3.1 Product Backlog Refinement
 - 2.3.2 Role of the Product Owner
 - 2.3.3 Responding to Change
 - 2.3.4 Critical Evaluation
- 2.4 Quality and Ownership Throughout the Product Cycle
 - 2.4.1 Definition of Done
 - 2.4.2 Test-Driven Development
 - 2.4.3 Continuous Integration
 - 2.4.4 Critical Evaluation
- 2.5 Flow of User Stories in Scrum
 - 2.5.1 User Story Lifecycle
 - 2.5.2 Acceptance Criteria
 - 2.5.3 Story Slicing
- 2.6 Role of User Story Mapping in Defining Iterations
 - 2.6.1 Mapping Epics, Features, and Stories
 - 2.6.2 Prioritisation and Release Planning
 - 2.6.3 Example Story Map Diagram
- 2.7 Summary of Findings

3. Story Estimation and Burndown Charts

3.1 Story Estimation Techniques

3.1.1 Planning Poker

3.1.2 Story Points and Fibonacci Sequence

3.1.3 T-Shirt Sizing

3.1.4 Velocity

3.2 Critical Evaluation of Estimation Practices

3.3 Purpose and Use of the Burndown Chart

3.3.1 Tracking Sprint Progress

3.3.2 Identifying Risks and Bottlenecks

3.3.3 Example Burndown Chart Diagram

4. DevOps, CI/CD, and Testing for a Web Application

4.1 Overview of the Selected Web Application

4.2 Components Requiring Testing

4.2.1 Functional Components

4.2.2 API Endpoints

4.2.3 User Interface Elements

4.2.4 Security and Performance

4.3 GitHub Actions for CI/CD

4.3.1 Workflow Triggers

4.3.2 Build, Test, and Deployment Stages

4.3.3 Example GitHub Workflow File

4.4 API Testing with Postman

4.4.1 Collections and Environments

4.4.2 Automated Test Scripts

4.4.3 Integration with CI/CD

4.5 UI Testing with Selenium

4.5.1 Browser Automation

4.5.2 Test Script Examples

4.5.3 Integration with GitHub Actions

4.6 Benefits and Limitations of DevOps Adoption

4.6.1 Organisational Benefits

4.6.2 Technical Benefits

4.6.3 Challenges and Limitations

4.7 Summary of Findings

5. Conclusion

6. References

Abstract

This report looks at how Agile, Scrum and DevOps work together in real software projects, using a simple REST API as a practical example. It explores how Scrum helps teams stay focused through short iterations, regular communication and the ability to adapt when requirements change. The report also discusses how user stories, story mapping and estimation techniques like Planning Poker and story points help teams plan their work realistically. A burndown chart is used to show how progress can be tracked during a sprint. Finally, the report shows how DevOps practices—such as CI/CD pipelines, Docker, Postman and Selenium—support automation, testing and reliable deployment. Overall, the findings highlight that combining Agile and DevOps leads to faster delivery, better teamwork and more dependable software.

1. Introduction

In recent years, Agile methodologies have become a central part of how modern software team's work. This shift has happened largely because organisations today operate in environments where requirements change quickly, customers expect frequent updates, and long planning cycles simply don't fit the pace of real-world development. Traditional, plan-driven approaches often struggle in these conditions because they rely heavily on predicting everything upfront. Agile, on the other hand, embraces uncertainty and encourages teams to learn and adapt as they go. These ideas were first formalised in the Agile Manifesto, which highlights the importance of collaboration, responsiveness and delivering value in small increments (Beck et al., 2001, n.p.).

Among the various Agile frameworks, Scrum has become one of the most widely adopted. Its popularity comes from the balance it strikes between structure and flexibility. Scrum gives teams a clear set of roles, events and artefacts to guide their work, but it also leaves enough room for teams to adapt the framework to their own context. At its core, Scrum is built on the principles of transparency, inspection and adaptation, which help teams navigate uncertainty and deliver value in short, focused cycles known as sprints (Schwaber and Sutherland, 2020, n.p.). This makes Scrum particularly effective for complex projects where requirements evolve over time and where early feedback is essential.

This report explores how Scrum supports four key Agile concepts: short iterations, face-to-face communication and collaboration, adaptability to changing requirements, and shared ownership of quality. These concepts are not just theoretical ideas—they shape how teams plan, communicate and deliver software on a daily basis. For example, short iterations help teams stay focused and deliver value regularly, while strong communication practices reduce misunderstandings and keep everyone aligned. Adaptability ensures that teams can respond to new information without derailing the

project, and shared ownership of quality encourages everyone—not just testers—to take responsibility for building reliable software.

To make these ideas more concrete, the report also looks at how user stories flow through the Scrum process and how techniques like user story mapping help teams organise and prioritise work. User stories are a simple but powerful way of capturing requirements from the user's perspective, and story mapping helps teams visualise how these stories fit together to form a complete user journey. Some of the diagrams included in the report are inspired by collaborative class activities, such as building story maps and visualising workflows on Miro boards, which helped reinforce how these practices work in real teams.

Beyond Agile practices, the report also examines how estimation techniques—such as Planning Poker and story points—help teams forecast their work and manage stakeholder expectations. Estimation is often one of the more challenging aspects of Scrum because it requires teams to balance uncertainty with the need for predictability. Understanding the strengths and limitations of these techniques is essential for effective sprint planning and long-term forecasting (Cohn, 2004, n.p.).

The final part of the report moves into the world of DevOps, which has become increasingly important as organisations aim to deliver software more frequently and reliably. While Scrum focuses on how teams develop software, DevOps focuses on how that software is built, tested, deployed and maintained. The report examines how CI/CD pipelines can be implemented using GitHub Actions, and how tools like Postman and Selenium support automated testing and rapid feedback (GitHub, 2024, n.p.; Postman, 2024, n.p.; Selenium, 2024, n.p.). These practices help bridge the gap between development and operations, enabling teams to deliver high-quality software at speed.

Overall, this report aims to show not only how Agile and DevOps work individually, but also how they complement each other when applied together. By combining iterative development with automation and continuous delivery, organisations can build software that is more reliable, more maintainable and better aligned with user needs. The analysis throughout the report reflects on both the strengths and limitations of these practices, offering a balanced and practical understanding of how they can be applied in real-world software projects.

2. Agile Concepts within the Scrum Methodology

Scrum has become one of the most widely used Agile frameworks because it translates Agile principles into practical routines that teams can follow every day. What makes Scrum effective is not just its ceremonies or artefacts, but the way it encourages teams to work iteratively, communicate openly, adapt to change and take shared responsibility for quality. These ideas are central to Agile thinking and are reflected in the structure of Scrum itself. In this section, I explore how Scrum supports these four concepts and how

they play out in real development environments. I also discuss how user stories flow through the Scrum process and how user story mapping helps teams plan iterations more effectively, drawing on techniques similar to those used in class activities.

2.1 Multiple Short Iterations in Scrum

Scrum brings the idea of short, focused iterations to life through sprints. These are time-boxed cycles—usually between one and four weeks—where the team commits to delivering a small but meaningful increment of the product (Atlassian, 2024, n.p.). The purpose of working in short iterations is to reduce risk, maintain momentum and create regular opportunities for feedback.

2.1.1 Sprint Planning

Each sprint begins with Sprint Planning, where the team and Product Owner come together to agree on the sprint goal and decide which user stories will be tackled. The Product Owner prioritises the backlog based on business value, while the development team estimates the effort involved and determines what they can realistically complete (Schwaber and Sutherland, 2020, n.p.). This collaborative discussion helps ensure that the sprint has a clear and achievable scope.

2.1.2 Sprint Execution

Once the sprint begins, the team works collaboratively to deliver the selected stories. Tasks are broken down, assigned and tracked using tools such as physical Kanban boards or digital platforms like Jira and Trello (Atlassian, 2024, n.p.). Because sprints are short, the team must stay focused and communicate frequently to avoid delays. This rhythm helps teams identify issues early and adjust their approach before problems escalate.

2.1.3 Sprint Review

At the end of the sprint, the team presents the completed increment to stakeholders during the Sprint Review. This ceremony is essential because it gives stakeholders a chance to see real progress and provide feedback that can influence future priorities (Scrum.org, 2020, n.p.). Regular reviews help ensure that the product evolves in a direction that genuinely meets user needs.

2.1.4 Critical Evaluation

Short iterations offer clear benefits: they improve predictability, reduce risk and allow teams to deliver value more frequently. However, they can also create pressure, especially if user stories are too large or poorly refined. Teams may feel rushed, and frequent planning and review meetings can become time-consuming if not managed well. These challenges highlight the importance of good backlog refinement and realistic sprint commitments.

2.2 Face-to-Face Communication and Collaboration

Scrum places a strong emphasis on communication because effective collaboration reduces misunderstandings and helps teams solve problems more quickly. This emphasis is reflected in several Scrum ceremonies.

2.2.1 Daily Scrum

The Daily Scrum is a short, focused meeting where team members synchronise their work, share progress and raise any obstacles they are facing (Schwaber and Sutherland, 2020, n.p.). Although it only lasts 15 minutes, it plays a crucial role in keeping the team aligned and ensuring that issues are addressed promptly.

2.2.2 Sprint Review and Retrospective

The Sprint Review encourages open communication between the team and stakeholders, while the Sprint Retrospective gives the team a chance to reflect on their own processes and identify areas for improvement (Scrum.org, 2020, n.p.). These ceremonies reinforce the idea that communication is not just about sharing information—it is about building trust and continuously improving how the team works.

2.2.3 Collaboration Tools

In distributed or hybrid teams, digital tools such as Miro, Microsoft Teams, Slack and Jira play a vital role in supporting real-time collaboration. During class activities, for example, Miro was used to create shared diagrams, demonstrating how digital tools can replicate the feel of face-to-face interaction even when team members are not physically together.

2.2.4 Critical Evaluation

While Scrum's communication-focused approach strengthens alignment and team cohesion, it can also lead to meeting fatigue if ceremonies are not facilitated effectively. Remote teams may face additional challenges such as time-zone differences or reduced engagement, which require deliberate effort to overcome.

2.3 Adaptability to Changing Requirements

One of Scrum's greatest strengths is its ability to handle change. Because sprints are short and the backlog is continuously refined, teams can adjust their priorities quickly without disrupting ongoing work (Schwaber and Sutherland, 2020, n.p.).

2.3.1 Product Backlog Refinement

Backlog refinement ensures that user stories remain relevant, prioritised and well-defined. The Product Owner updates the backlog based on stakeholder feedback, market changes and insights gained from previous sprints (Scrum Alliance, 2020, n.p.).

This ongoing process helps the team stay focused on delivering the highest-value features.

2.3.2 Role of the Product Owner

The Product Owner plays a central role in Scrum's adaptability. Acting as the voice of the customer, they ensure that the team works on the most valuable items and **can** re-prioritise work between sprints when necessary (Schwaber and Sutherland, 2020, n.p.).

2.3.3 Responding to Change

Because Scrum operates in short cycles, changes can be incorporated at the start of each sprint without disrupting the team's current work. This reduces the risk of building features that no longer meet user needs.

2.3.4 Critical Evaluation

Although adaptability is a major strength, it can also create instability if requirements change too frequently. Teams may struggle to maintain velocity, and constant re-prioritisation can lead to technical debt or incomplete features. Effective communication and disciplined backlog management are essential to avoid these pitfalls.

2.4 Quality and Ownership Throughout the Product Cycle

Scrum encourages shared responsibility for quality, meaning that every team member—not just testers—contributes to building a reliable product.

2.4.1 Definition of Done (DoD)

The Definition of Done outlines the criteria that must be met for a user story to be considered complete. This may include coding standards, testing requirements, documentation and integration tasks (Scrum.org, 2020, n.p.). A strong DoD ensures consistency and prevents unfinished work from accumulating.

2.4.2 Test-Driven Development (TDD)

Test-Driven Development encourages developers to write tests before implementing functionality. This practice improves code quality, reduces defects and supports continuous integration (Humble and Farley, 2010, n.p.).

2.4.3 Continuous Integration (CI)

Continuous Integration tools automatically build and test code whenever changes are committed. This reduces integration issues and helps maintain a stable codebase throughout development (Red Hat, 2024, n.p.).

2.4.4 Critical Evaluation

Shared ownership of quality leads to more reliable software, but it requires discipline and a strong team culture. Without consistent adherence to the DoD or testing practices, quality can deteriorate. Implementing CI pipelines also requires technical expertise and initial setup time, which can be challenging for less experienced teams.

2.5 Flow of User Stories in Scrum

User stories are central to Scrum because they represent small, manageable units of work that describe functionality from the user's perspective. Their flow through the Scrum process helps maintain transparency and ensures that work aligns with business value (Cohn, 2004, n.p.).

2.5.1 User Story Lifecycle

A typical user story moves through several stages: it is created in the product backlog, prioritised by the Product Owner, selected during Sprint Planning, broken down into tasks, developed, tested and finally reviewed at the end of the sprint.

2.5.2 Acceptance Criteria

Acceptance criteria define the conditions that must be met for a story to be accepted. They guide development, support testing and ensure that the final output aligns with user expectations.

2.5.3 Story Slicing

Large stories, often called epics, are broken into smaller slices that can be delivered within a single sprint. This supports incremental delivery and reduces the risk of work spilling over into multiple sprints.

2.6 Role of User Story Mapping in Defining Iterations

User story mapping is a visual technique that helps teams organise and prioritise user stories based on user journeys and system functionality (Patton, 2014, n.p.). It provides a holistic view of the product and helps teams understand how different pieces fit together.

2.6.1 Mapping Epics, Features and Stories

A story map typically arranges epics horizontally to represent the user journey, with features and stories listed vertically beneath each epic. This structure helps teams identify dependencies and understand the overall workflow.

2.6.2 Prioritisation and Release Planning

Story maps also support prioritisation and release planning. By slicing functionality into iterations, teams can ensure that each sprint delivers a coherent set of features that provide real value to users.

2.6.3 Example Story Map Diagram

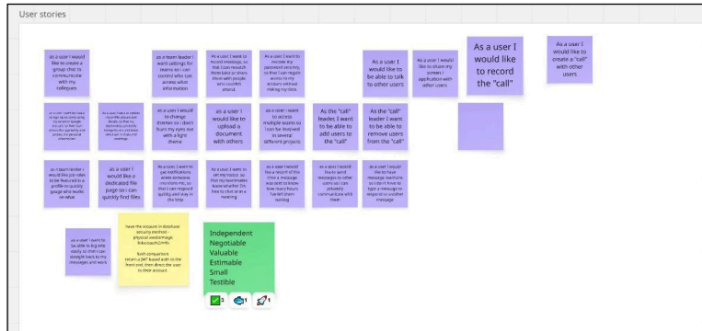


Figure 1: User Story Mapping Board

2.7 Summary of Findings

Overall, Scrum supports the four core Agile concepts through its structured ceremonies, iterative development cycles, emphasis on communication and strong quality practices. However, its success depends heavily on team discipline, clear communication and effective backlog management. User stories and story mapping play a central role in defining iterations and ensuring that work remains aligned with user needs and organisational goals.

3. Story Estimation and Burndown Charts

Estimation plays a central role in Scrum because it helps teams plan their work realistically, manage stakeholder expectations and maintain a sustainable pace throughout development. Although estimation will never be perfect—software development is too unpredictable for that—it provides a shared understanding of the effort and complexity behind each user story. When done well, estimation supports transparency and helps teams make informed decisions about what they can achieve within a sprint. This section explores the main estimation techniques used in Agile teams, discusses their strengths and weaknesses and explains how the burndown chart helps teams monitor progress during a sprint.

3.1 Story Estimation Techniques

Scrum teams usually rely on **relative estimation**, which means they compare the effort of a new story to work they have completed before rather than trying to predict exact hours. This approach is widely used because it reduces pressure, encourages team discussion and becomes more accurate over time as the team builds shared experience (Cohn, 2004, n.p.). Instead of asking, “How long will this take?”, the team asks, “How big is this compared to what we’ve done before?” This shift in thinking helps teams focus on complexity and uncertainty rather than time.

3.1.1 Planning Poker

Planning Poker is one of the most popular estimation techniques in Agile teams because it turns estimation into a collaborative activity rather than a top-down decision. Each team member privately selects a card that represents their estimate—often using values like 1, 2, 3, 5, 8 or 13—and everyone reveals their cards at the same time. This avoids anchoring, where early suggestions influence others (Cohn, 2004, n.p.).

When estimates differ, the team discusses why. These conversations are often more valuable than the estimate itself because they uncover hidden assumptions, unclear requirements or technical risks that might otherwise go unnoticed. Planning Poker works best when stories are well-refined; otherwise, the discussion can drift or become repetitive.

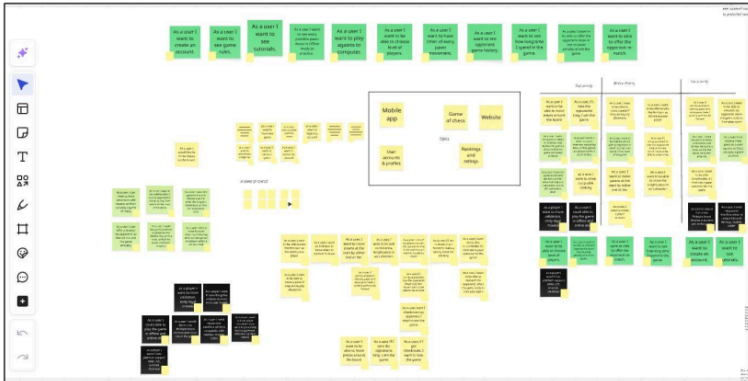


Figure 2: User Story Mapping Board Used for Planning Poker Estimation

3.1.2 Story Points and the Fibonacci Sequence

Story points are the most common unit of estimation in Scrum. They represent the **relative effort, complexity and uncertainty** of a user story. Many teams use the Fibonacci sequence (1, 2, 3, 5, 8, 13, 21) because the increasing gaps between numbers reflect the reality that larger tasks carry more uncertainty (Atlassian, 2024, n.p.).

The benefit of story points is that they shift the conversation away from time and toward value. Instead of debating whether something will take six or seven hours, the team focuses on how complex the work is and what risks it carries. Over time, story points help teams understand their capacity and plan sprints more effectively.

3.1.3 T-Shirt Sizing

T-shirt sizing (XS, S, M, L, XL) is a simpler, high-level estimation method often used early in a project when requirements are still vague. It allows teams to quickly group stories by size without getting bogged down in detail. This makes it useful for initial backlog grooming or release planning. However, once the team moves into sprint planning, T-shirt sizes are usually replaced by story points for more precise estimation (Scrum Alliance, 2020, n.p.).

3.1.4 Velocity

Velocity refers to the number of story points a team completes in a sprint. After several sprints, velocity tends to stabilise, giving the team a reliable indicator of how much work they can take on in future iterations (Cohn, 2004, n.p.). Velocity is helpful for forecasting, but it must be used carefully. If stakeholders treat velocity as a performance metric, teams may feel pressured to inflate estimates or cut corners to “hit the numbers.” Velocity should guide planning—not measure productivity.

3.2 Critical Evaluation of Estimation Practices

Although estimation techniques are useful, they are far from perfect. Relative estimation reduces the stress of predicting exact hours, but it still depends heavily on team experience. New teams often struggle because they lack shared reference points or familiarity with the codebase. As a result, early estimates can be inconsistent or overly optimistic.

Planning Poker encourages valuable discussion, but it can become time-consuming if used for every story or if the backlog contains poorly defined items. Story points help teams focus on complexity rather than time, yet stakeholders sometimes misunderstand them and try to convert them into hours. This misunderstanding can create tension or unrealistic expectations.

Velocity is another area where problems can arise. It can fluctuate due to holidays, team changes, technical debt or unexpected blockers. When velocity drops, teams may feel pressure to “catch up,” even when the drop is caused by legitimate factors. This highlights the importance of treating velocity as a planning tool rather than a target.

Despite these limitations, estimation remains a valuable practice when combined with clear acceptance criteria, regular backlog refinement and open communication. The goal is not perfect accuracy but continuous improvement.

3.3 Purpose and Use of the Burndown Chart

The burndown chart is one of the simplest yet most effective tools in Scrum. It provides a visual representation of how much work remains in the sprint and whether the team is on track to complete it. The horizontal axis shows the days of the sprint, while the vertical axis shows the remaining story points or tasks (Atlassian, 2024, n.p.). As the team completes work, the line “burns down” toward zero.

3.3.1 Tracking Sprint Progress

A healthy burndown chart shows a steady downward trend. This indicates that the team is completing work consistently throughout the sprint. If the line remains flat for several days, it suggests that work is not being finished, which may indicate unclear requirements, technical challenges or a lack of task breakdown. The burndown chart helps the team spot these issues early rather than discovering them at the end of the sprint.

3.3.2 Identifying Risks and Bottlenecks

Burndown charts are also useful for identifying risks. Sudden drops may indicate that tasks were not updated regularly, while long periods of inactivity can signal deeper issues such as over-commitment or blockers. When the chart deviates significantly from the ideal line, the Scrum Master can step in to help the team address impediments before they escalate (Schwaber and Sutherland, 2020, n.p.).

3.3.3 Example Burndown Chart Diagram

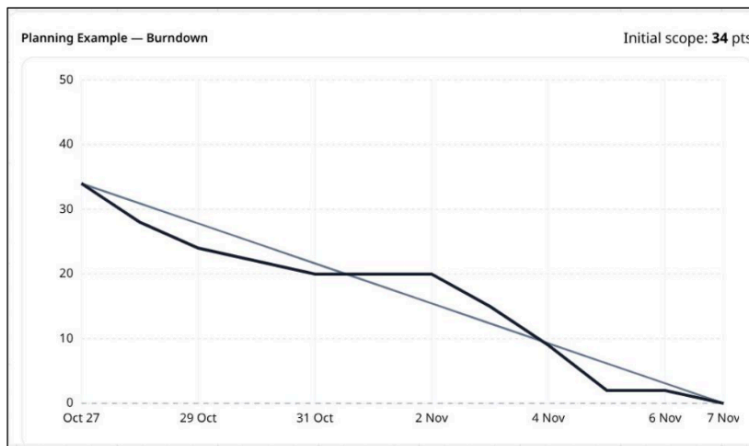


Figure 3: Sprint Burndown Chart Showing Ideal vs Actual Progress

4. DevOps, CI/CD, and Testing for a Web Application

This section looks at how DevOps practices can support modern software delivery by using a simple REST API developed during class as a practical example. The application itself is relatively small, but it still provides a realistic environment for exploring continuous integration, continuous deployment and automated testing. The backend is built with TypeScript and Express, uses PostgreSQL for data storage and is containerised with Docker. Even though the system is not large-scale, it includes enough moving parts to demonstrate how DevOps tools—such as GitHub Actions, Postman and Selenium—can help teams automate workflows, improve reliability and reduce manual effort.

4.1 Overview of the Selected Web Application

The application exposes several REST API endpoints that allow users to interact with stored data. These include routes for retrieving information, updating records and handling authentication. The backend follows a modular structure, separating routes, services and database configuration into different folders. This separation makes the codebase easier to maintain and test, as each part of the system has a clear responsibility.

In Visual Studio Code, the project structure shows the server configuration, database connection logic and route definitions. This layout forms the foundation for the testing and automation practices discussed later in this section.

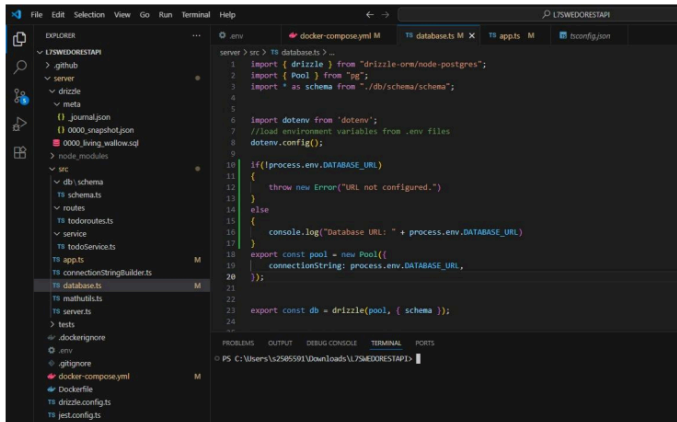


Figure 4: Visual Studio Code Showing REST API Backend Configuratio

4.2 Components Requiring Testing

Because the application interacts with a live database and handles user data, testing becomes essential rather than optional. Each API endpoint must be validated to ensure it returns the correct response, handles errors properly and enforces validation rules. For example, a GET endpoint must return accurate data, while POST or PUT endpoints must reject invalid or malicious input.

The database layer also needs testing to confirm that queries run correctly and that the schema behaves as expected. Since the application relies on environment variables and external services, configuration testing is equally important to avoid deployment issues.

Security testing is another critical area. The system must be protected against common vulnerabilities such as SQL injection, insecure authentication flows and misconfigured CORS policies. Performance testing ensures that the API remains responsive even when handling multiple requests.

Together, these testing activities help ensure that the system remains reliable, secure and maintainable as it evolves.

4.3 GitHub Actions for CI/CD

GitHub Actions provides an automated workflow engine that supports both continuous integration and continuous deployment. Whenever developers push new code or open a pull request, GitHub Actions can automatically install dependencies, run tests and prepare the application for deployment (GitHub, 2024, n.p.). This automation reduces manual effort and helps maintain stability as the codebase grows.

A typical CI/CD pipeline begins by checking out the latest version of the code and installing the required packages. Automated tests then run to verify that recent changes have not introduced defects. If all tests pass, the pipeline can deploy the updated application to a staging environment. Once validated, the same pipeline can promote the build to production.

Containerisation in the CI/CD Process

To ensure consistent deployment environments, the application is containerised using Docker. Containerisation allows the CI/CD pipeline to run the application in identical environments across development, staging and production. This reduces environment-related issues and supports reliable automated deployment.

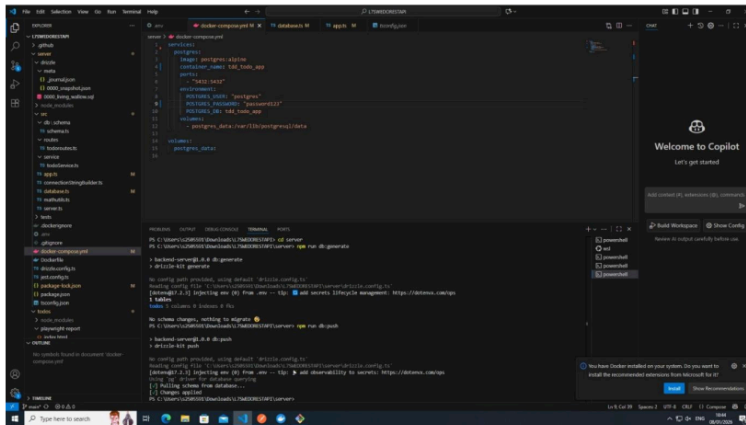


Figure 5: Docker Compose Configuration for PostgreSQL in VS Code

4.4 API Testing with Postman

Postman played an important role during development by allowing the team to test API endpoints quickly and interactively. It enables developers to send requests, inspect responses and write automated test scripts. This is particularly useful for verifying authentication flows, CRUD operations and error handling (Postman, 2024, n.p.).

The screenshot below shows a GET request sent to the /hello endpoint during testing. The request history illustrates the iterative nature of API development, where developers repeatedly test and refine endpoints.

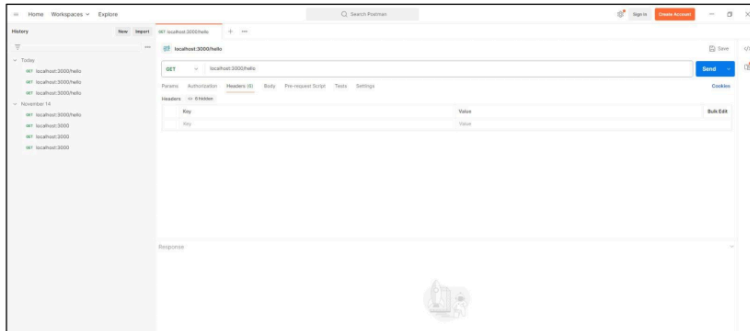


Figure 6: API Testing of REST Endpoint Using Postman

Postman collections can also be integrated into GitHub Actions using Newman, enabling automated API testing during each pipeline run.

4.5 UI Testing with Selenium

Although the current application is primarily backend-focused, Selenium can be used to automate UI testing for any frontend components added in the future. Selenium simulates user interactions such as clicking buttons, filling out forms and navigating between pages (Selenium, 2024, n.p.). This helps ensure that the interface behaves correctly across different browsers and devices.

In a full implementation, Selenium tests would be added to the CI pipeline so that they run automatically whenever new code is pushed. Running Selenium in a headless browser ensures that tests execute quickly and consistently across different environments.

4.6 Benefits and Limitations of DevOps Adoption

Adopting DevOps practices brings several advantages to a development team. Automated pipelines reduce the time between writing code and delivering it to users, while continuous testing improves reliability and reduces the likelihood of defects reaching production. Collaboration between development and operations teams becomes smoother, as both groups share responsibility for the system's performance and stability (Humble and Farley, 2010, n.p.).

However, DevOps also introduces challenges. Teams must adapt to new tools and workflows, which can require training and cultural change. Setting up CI/CD pipelines

and automated tests takes time, and inexperienced teams may struggle with the complexity of the tooling. Security must also be carefully managed, as automated deployments can introduce risks if not properly controlled.

Despite these challenges, the long-term benefits—faster delivery, improved quality and stronger collaboration—make DevOps a valuable approach for modern software development.

4.7 Summary of Findings

This section has shown how DevOps practices, CI/CD pipelines and automated testing tools work together to support reliable software delivery. GitHub Actions provides a flexible platform for automation, while Postman and Selenium ensure that both backend and frontend components behave as expected. Although DevOps adoption requires effort and organisational change, it ultimately leads to more efficient workflows and higher-quality software.

5. Conclusion

This report has explored how Agile development, Scrum practices and DevOps automation work together to support the delivery of reliable, high-quality software. By examining the four core Agile concepts—short iterations, communication, adaptability and shared ownership—it becomes clear that Scrum provides a structured yet flexible framework that encourages continuous learning and incremental value delivery. The flow of user stories and the use of story mapping further strengthen this process by helping teams maintain clarity, prioritise effectively and align their work with real user needs.

The discussion on estimation techniques highlighted the importance of collaborative decision-making and relative sizing in managing uncertainty. Although estimation is never perfect, tools such as Planning Poker, story points and velocity give teams a practical way to forecast their capacity and maintain a sustainable pace. The burndown chart reinforces this by offering a simple but powerful visual indicator of sprint progress, enabling teams to identify risks early and adjust their approach when necessary.

The final section demonstrated how DevOps practices extend Agile principles into the operational side of software delivery. Using a web application as a practical example, the report showed how GitHub Actions can automate building, testing and deployment, while tools like Postman and Selenium ensure that both backend and frontend components behave as expected. Although DevOps adoption requires cultural change, technical investment and careful attention to security, the long-term benefits—faster releases, improved collaboration and greater reliability—make it a valuable evolution of Agile thinking.

Overall, the integration of Agile and DevOps creates a development environment that is responsive, collaborative and quality-driven. Scrum provides the structure for iterative development, while DevOps ensures that each increment is tested, validated and deployed efficiently. Together, they form a modern approach to software engineering that supports continuous improvement and delivers meaningful value to users.

6. References

Atlassian (2024) *Agile Estimation and Story Points*. Available at: <https://www.atlassian.com/agile/estimation> (Accessed: 10 January 2026).

Atlassian (2024) *What is Scrum?* Available at: <https://www.atlassian.com/agile/scrum> (Accessed: 10 January 2026).

Beck, K. et al. (2001) *Manifesto for Agile Software Development*. Available at: <https://agilemanifesto.org> (Accessed: 10 January 2026).

Cohn, M. (2004) *User Stories Applied: For Agile Software Development*. Boston: Addison Wesley.

GitHub (2024) *GitHub Actions Documentation*. Available at: <https://docs.github.com/actions> (Accessed: 10 January 2026).

Humble, J. and Farley, D. (2010) *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*. Boston: Addison Wesley.

Patton, J. (2014) *User Story Mapping: Discover the Whole Story, Build the Right Product*. Sebastopol: O'Reilly Media.

Postman (2024) *Postman API Platform Documentation*. Available at: <https://learning.postman.com> (Accessed: 10 January 2026).

Red Hat (2024) *What is Continuous Integration?* Available at: <https://www.redhat.com/en/topics/devops/what-is-ci> (redhat.com in Bing) (Accessed: 10 January 2026).

Schwaber, K. and Sutherland, J. (2020) *The Scrum Guide: The Definitive Guide to Scrum*. Available at: <https://scrumguides.org> (Accessed: 10 January 2026).

Scrum Alliance (2020) *Backlog Refinement*. Available at: <https://resources.scrumalliance.org> (Accessed: 10 January 2026).

Scrum.org (2020) *Scrum Framework Resources*. Available at: <https://www.scrum.org/resources> (Accessed: 10 January 2026).

Selenium (2024) *Selenium WebDriver Documentation*. Available at: <https://www.selenium.dev/documentation> (Accessed: 10 January 2026).

A Comprehensive Analysis of Agile, Scrum and DevOps Practices in Contemporary Software Engineering

GRADEMARK REPORT

FINAL GRADE

59/100

GENERAL COMMENTS

Overall, this is a well-structured and clearly written submission that demonstrates a good understanding of Agile principles, Scrum methodology, and DevOps practices.

The report follows all documentation requirements correctly, with appropriate use of numbered headings, figure captions, and a logical flow from introduction to conclusion.

Critical evaluation sections are present throughout, which is great, though these could have benefited from deeper scholarly engagement.

The most significant weakness is the over-reliance on practitioner websites and industry documentation at the expense of peer-reviewed academic journal articles

Section 3 would have been strengthened considerably by including an actual GitHub Actions workflow YAML file and concrete Selenium test examples rather than hypothetical discussion (Easy to find default examples online)

Agile Concepts and Scrum Methodology

Section 1 Engagement with Literature 8 out of 16

Section 1 Knowledge and Understanding 10 out of 16

Section 1 Cognitive and Intellectual Skills 6 out of 8

Total 24 out of 40

Estimation and Burndown Charts

Section 2 Engagement with Literature 4 out of 8

Section 2 Knowledge and Understanding 4 out of 8

Section 2 Cognitive and Intellectual Skills 3 out of 4

Total 11 out of 20

SwDevOps Evaluation

Section 3 Engagement with Literature 10 out of 16

Section 3 Knowledge and Understanding 9 out of 16

Section 3 Cognitive and Intellectual Skills 5 out of 8

Total 24 out of 40

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
