

kelly casey agile report

by Kelly Casey

Submission date: 11-Feb-2026 04:33PM (UTC+0000)

Submission ID: 275127996

File name: 31343_Kelly_Casey_kelly_casey_agile_report_694650_1251824182.docx (1.94M)

Word count: 6946

Character count: 41068



AGILE DEVELOPMENT AND SOFTWARE DEV OPS ASSIGNMENT

ACCA7023



Kelly Casey
2106451

Contents

1. Introduction.....	2
2. Agile Development and Scrum Overview	3
3. Critical Evaluation of Agile Concepts within the Scrum Methodology	5
3.1 Multiple Short Iterations.....	5
3.2 Emphasis on Face-to-Face Communication and Collaboration.....	6
3.3 Adaptability to Changing Requirements	6
3.4 Quality of Ownership Throughout the Product Cycle	7
3.5 Flow of User Stories and the Role of User Story Mapping	8
4. Estimation of User Stories and the Use of Burndown Charts	12
4.1 Estimating User Story Implementation Time and Difficulty	12
4.2 Purpose and Use of the Burndown Chart.....	12
4.3 Practical Application of User Story Estimation.....	13
5. DevOps Tools and Testing in a Web Application.....	17
5.1 Web Application Components Requiring Testing	17
5.2 GitHub Actions for Continuous Integration and Deployment.....	18
5.3 API Testing Using Postman	20
5.4 User Interface Testing Using Selenium	21
5.5 Benefits and Limitations of Adopting Software DevOps	21
5.6 Alignment Between Agile and DevOps Practices	22
6. Conclusion.....	23
Bibliography	24

1. Introduction

Agile software development has become the main approach in software engineering. It supports fast changes, continuous delivery, and close teamwork between developers and stakeholders. Traditional methods like the waterfall model often struggle when requirements change or the environment moves quickly, which can lead to delays and products that don't fully meet user needs. Agile takes a different path by focusing on iterative development, regular feedback, and flexible planning. This lets teams deliver working software in small steps while improving both the product and their process along the way [1].

Of the many Agile frameworks, Scrum is one of the most widely used in both industry and university settings. It organises work into short, fixed-length iterations called sprints, and includes defined roles, artefacts, and events that encourage transparency, regular reviews, and adaptation throughout the project [2].

Core Agile principles such as short iterations, direct communication, openness to changing requirements, and shared responsibility for quality are built into Scrum. How well these ideas work depends on how consistently the team follows the framework and how well they use supporting practices, such as managing user stories, estimating effort, and maintaining good feedback [3].

User stories are central to Scrum. They describe requirements from the end user's viewpoint and help prioritise work in the product backlog. The way user stories move from early refinement to sprint delivery is important for keeping the team focused on delivering real value. User story mapping adds to this by providing a visual map of the user's journey and showing how features connect, which helps teams plan effective sprints and decide what belongs in a minimum viable product [4]. When these techniques are used well, they improve requirement clarity, communication with stakeholders, and the team's ability to adapt to change.

At the same time, Software Development and Operations (DevOps) practices have developed to tackle issues around integrating, testing, deploying, and maintaining software reliably. DevOps emphasises automation, continuous integration and delivery, and collaboration between development and operations teams [5]. Tools like GitHub Actions and Postman support automated pipelines, API testing, and UI checks, making it easier to deliver stable software at scale. Combining Agile with DevOps often leads to higher quality, more frequent releases, and better responsiveness, though it also involves cultural and technical challenges [6].

This report examines how effectively Scrum puts into practice four core Agile concepts: short iterations, collaboration, adaptability, and shared ownership of quality. Particular focus is given to the flow of user stories and the use of user story mapping. The report also looks at common methods for estimating user story complexity and duration, including the use of burndown charts to track sprint progress. Finally, it considers how DevOps tools and practices can be applied in

developing a web application, discussing both the advantages and the limitations of this approach. Through this analysis, the report aims to show a clear understanding of how Agile and DevOps practices support the efficient delivery of reliable software, in line with the module learning outcomes.

2. Agile Development and Scrum Overview

Agile software development came about as a direct response to the shortcomings of traditional, plan-driven approaches. Those older methods relied heavily on detailed upfront planning, lots of documentation, and delivering the full system only at the end. They often couldn't keep up when requirements changed quickly, stakeholders shifted their expectations, or projects grew more complex. Agile offers a different way forward: by prioritising flexibility, delivering in small steps, and keeping stakeholders involved throughout. This helps teams handle uncertainty better while staying focused on producing working software that actually delivers value [7].

One of the defining features of Agile is its use of iterative and incremental processes. Iterative development means working in repeated cycles where features are refined based on feedback and regular evaluation. Incremental development builds the system gradually by adding completed pieces of functionality over time. Most Agile approaches combine the two, so teams can get usable software into users' hands early and keep adapting it as new needs or insights emerge [7]. This is quite different from traditional incremental models, where you usually don't see a complete, working system until much later in the project.

Within the broader Agile world, Scrum has become one of the most popular and widely used frameworks for organising software development. Scrum isn't a strict, step-by-step methodology; it's more of a lightweight framework that provides structure for collaboration, accountability, and tracking progress. It is built around the core Agile ideas of transparency, inspection, and adaptation across the entire development process [8].

Scrum defines three main roles, each with clear responsibilities. The Product Owner acts on behalf of stakeholders, managing and prioritising the product backlog to ensure the team focuses on the most valuable work. The Scrum Master serves as a facilitator and coach, helping the team follow Scrum properly and removing any obstacles that slow them down. The Development Team is a self-organising, cross-functional group that delivers potentially shippable increments of software by the end of each sprint. This setup encourages shared ownership and moves away from traditional top-down management.

Key to making Scrum work are its artefacts, which give everyone clear visibility into what's happening. The Product Backlog is an ordered list of everything that might be needed such as features, requirements and fixes which are often written as user

stories. During sprint planning, the team pulls a subset of this into the Sprint Backlog, which becomes their focus for the current sprint. At the end, the Increment is the total of all completed items that meet the team's Definition of Done, ensuring quality is built in from the start rather than checked only at the finish [2].

Scrum organises work into fixed-length iterations called sprints, usually lasting one to four weeks. Every sprint starts with Sprint Planning, where the team agrees on what to tackle and sets a sprint goal. Progress is coordinated daily through the Daily Scrum, a brief meeting that keeps everyone aligned and highlights issues early. When the sprint ends, the team holds a Sprint Review to show the completed work to stakeholders and collect feedback, then follows up with a Sprint Retrospective to reflect on how things went and plan improvements .

This iterative setup allows for regular inspection and adaptation, so teams can bring in feedback and adjust to new requirements at short, predictable intervals. Instead of locking requirements in place, Scrum welcomes change by letting the backlog be reprioritised between sprints. That flexibility fits well with Agile's emphasis on responding to change and collaborating with customers over rigidly following an initial plan [9]. At the same time, the time-boxed sprints provide focus and a degree of predictability.

User stories are especially important in Scrum because they capture requirements from the end user's point of view. They are kept simple and lightweight at first, then refined over time as the team discusses and clarifies details. The way user stories move through the product backlog, into sprint planning, and eventually to delivery supports steady, incremental progress while keeping the focus on user value. This flow also sets the stage for more advanced practices, such as user story mapping, which the report will cover later [2].

In summary, Scrum delivers a practical, structured framework that puts Agile principles into action through its roles, artefacts, and events. By combining iterative work, ongoing feedback, and collective responsibility, it provides an effective way to manage the complexity of today's software projects. That said, how well Scrum actually depends on those core Agile ideas and how the team applies these elements in practice, something this report will evaluate more critically in the sections that follow.

3. Critical Evaluation of Agile Concepts within the Scrum Methodology

Scrum is designed to put core Agile principles into practice through its structured roles, artefacts, and events. While the framework is widely adopted and usually effective, how well it realises the key Agile concepts of short iterations, collaboration, adaptability, and shared quality ownership depends on how rigorously and appropriately it is applied in practice. This section critically evaluates each of these concepts within Scrum, with particular emphasis on the flow of user stories and the role of user story mapping in defining iterations.

3.1 Multiple Short Iterations

One of the key features of Agile development is its reliance on multiple short iterations. This approach allows teams to deliver working software frequently and creates regular chances for feedback and adjustment. In Scrum, this principle is put into action through time-boxed sprints, which usually last one to four weeks. These short cycles help teams break down complex systems into smaller, manageable pieces and deliver value steadily instead of waiting until the project is nearly finished.

In practice, short iterations bring several clear benefits. They lower overall project risk by surfacing technical problems or unclear requirements early rather than later. Stakeholders get to see and comment on real progress at the end of each sprint, which improves visibility and trust. Sprint retrospectives also provide a structured way for the team to reflect and make ongoing improvements. Research has shown that this kind of iterative delivery tends to increase stakeholder satisfaction and make project status much clearer compared to traditional long-cycle methods [10]. Requiring each sprint to produce a potentially shippable increment also helps build discipline and keeps the team focused on delivering quality work.

That said, short iterations don't automatically succeed in Scrum. If the product backlog isn't well refined or user stories aren't properly broken down, teams can end up with incomplete or substandard increments. Unrealistic sprint goals or poorly understood stories often lead to work carrying over to the next sprint, which weakens the whole point of regular delivery. Very short sprints can also add extra pressure, increase administrative tasks, and raise cognitive load, especially for teams that are still building their process maturity [11]. Overall, while Scrum gives a solid framework for short, iterative development, making it work well depends strongly on good backlog management and the team's level of experience.

3.2 Emphasis on Face-to-Face Communication and Collaboration

Agile puts a strong emphasis on direct communication and collaboration, preferring face-to-face interaction over extensive documentation. Scrum puts this principle into practice mainly through its key events, the Daily Scrum, Sprint Planning, Sprint Review, and Sprint Retrospective. These meetings are intentionally structured to build transparency, create shared understanding, and help the team resolve issues quickly [2].

The Daily Scrum stands out as a simple but powerful way to keep communication flowing. Every day, team members quickly sync up on what they've done, what they'll do next, and any blockers they're facing. This short, focused check-in helps catch problems early and keeps everyone aligned. Sprint Reviews go further by bringing stakeholders into the conversation, showing them working software and gathering their feedback while the project is still in progress. Research supports this: regular, direct interactions like these tend to strengthen team cohesion and cut down on misunderstandings, particularly when dealing with complex or changing requirements .

That said, Scrum's reliance on face-to-face communication can run into real difficulties in distributed or remote teams. Tools like video calls, shared boards, and chat platforms can help bridge the gap, but they rarely match the natural flow and nuance of being in the same room. Studies of distributed Scrum teams point to higher chances of miscommunication, less spontaneous knowledge sharing, and a greater need to fall back on written documentation when people aren't co-located [2]. As a result, while Scrum provides excellent support for collaborative communication in ideal conditions, its success in this area often depends on the team's setup, tools, and ability to adapt to remote or hybrid working.

3.3 Adaptability to Changing Requirements

Adaptability is a core part of Agile development, based on the understanding that requirements almost always change as stakeholders get a better sense of what they really need. In Scrum, this adaptability is mainly supported through ongoing backlog refinement and the ability to reprioritise items between sprints. By keeping changes outside the current sprint, Scrum strikes a balance: the team gets the flexibility to adjust direction while still having a stable focus during the sprint itself [12].

User stories are particularly helpful here. Because they start out simple and lightweight, they let requirements develop gradually through team discussions and refinement sessions rather than needing everything nailed down at the beginning. Studies have shown that this way of working makes teams more responsive to

change and keeps the cost of handling late changes much lower compared to traditional approaches [13].

That said, Scrum's approach to adaptability does have its limits. While it's easy to bring in new or changed items between sprints, mid-sprint changes are actively discouraged to protect the team's focus and commitment. In very fast-moving or unpredictable environments, this can feel restrictive. Frequent unclear ownership of the product backlog can also create churn, disrupt momentum, and wear down team morale over time. For adaptability to work well in practice, the backlog needs consistent management and refinement, and the Product Owner must have clear authority to make prioritisation decisions.

3.4 Quality of Ownership Throughout the Product Cycle

Agile stresses shared responsibility for quality, rather than treating it as something to be checked only at the end in a separate testing phase. In Scrum, this principle is supported through the idea of collective ownership, which is strengthened by the Definition of Done. By making sure every backlog item meets the team's agreed quality standards before it can be marked as complete, Scrum builds quality checks into every sprint rather than leaving them until later [2].

This way of working encourages the whole team to take responsibility for both delivering features and maintaining quality. It helps prevent technical debt from building up and supports a more sustainable pace over time. Regular inspection during sprint reviews and retrospectives also keeps accountability alive and gives the team chances to learn and improve their practices [14].

That said, shared ownership isn't without its difficulties. If the Definition of Done isn't clearly defined or the team lacks enough testing knowledge and skills, people can end up interpreting "done" differently, which often leads to inconsistent or declining quality. Collective responsibility can also make it harder to pinpoint who should address specific issues if roles and expectations aren't well communicated. In the end, while Scrum provides a solid structure for shared quality ownership, how well it works in practice depends on consistent discipline, clear standards, and proper support from the organisation.

3.5 Flow of User Stories and the Role of User Story Mapping

The flow of user stories is at the heart of how Scrum turns requirements into actual working software. User stories start out in the product backlog, get refined and clarified through team discussions, are chosen and broken down during sprint planning, and if everything goes to plan end up delivered as part of the sprint increment. This steady progression supports incremental delivery and keeps the team's work closely tied to delivering real value to users [15].

User story mapping builds on this flow by giving a visual overview of the user's journey and how different features connect and depend on each other. By laying stories out along a user-centred sequence, this helps teams spot the minimum viable set of functionality and plan iterations that make sense as a whole rather than isolated pieces. Studies have shown that visual techniques like story mapping lead to better prioritisation decisions, clearer conversations with stakeholders, and more effective sprint and release planning [16].

However, user story mapping isn't perfect for every situation. Creating and keeping the map up to date can take a fair amount of time and effort, especially as the product grows or changes frequently. For very small teams, or in projects where the system is mostly backend/technical with little direct user interaction, the overhead might outweigh the benefits. Even so, when used appropriately, user story mapping is a valuable way to keep iteration planning grounded in the user experience and aligned with the broader goals of the project.

These concepts were put into practice during in-class practical sessions on requirements analysis and user-centred design. In a group exercise, we developed use-case diagrams together with detailed structured use-case descriptions to capture important system interactions, including report submission, viewing documents, and managing feedback. These artefacts helped build a common understanding of the system's boundaries and how users would interact with it, which tied directly into Scrum's principles of team collaboration and refining requirements step by step through discussion.

Figure 1 shows the high-level use-case model created during the in-class exercise, providing an overview of the main actors and their interactions with the system. Figures 2 and 3 then illustrate how specific interactions were detailed further, using include and extend relationships to show dependencies and optional behaviours, along with structured use-case descriptions that outline preconditions, main flows, alternative flows, and postconditions.

These diagrams and descriptions helped clarify the system's functionality step by step and supported a shared team understanding of user needs and system boundaries.

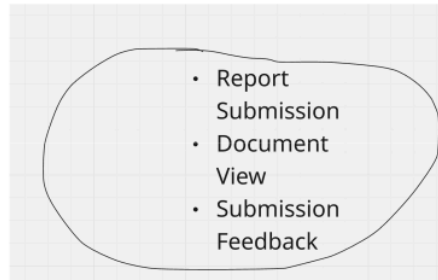


Figure 1 Use case overview identifying the core system functionalities defined during an in-class Agile requirements analysis exercise.

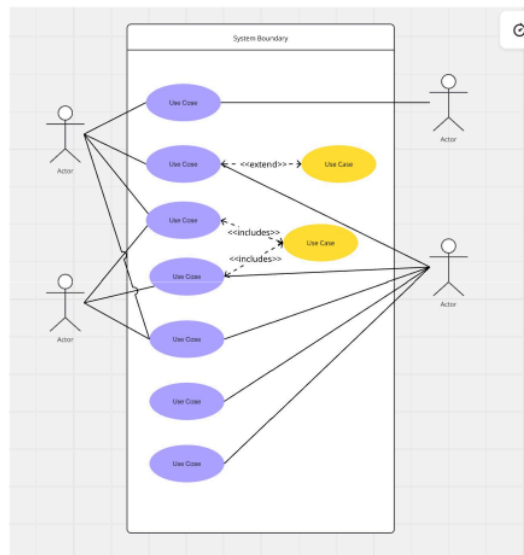


Figure 2 Detailed use case diagram developed during an in-class group exercise, illustrating actor interactions and the refinement of system behaviour using includes and extends relationships

Use case name	View module content	Use Case Code	UC2
Actors	User, Student		
Process Description			
User selects a module they have been enrolled in and be able to view / download content within module.			
Process Tasks			
1. Select module			
2. See sections			
3. Select sections			
4. See content within sections			
5. Click on content			
6. Download / View document			
Extended Tasks			
1. User does not have privileges to view module			
2. User cannot see hidden sections / content			
Post-Conditions			
Student view / downloaded content from module and section			

Figure 3 Example structured use case description created during an in-class group exercise to clarify user interactions, alternative flows, and post-conditions

Following the use-case analysis, Class Responsibility Collaboration (CRC) cards were produced during the practical session to support object-oriented design thinking. The CRC cards were used to identify key classes within the system, define their core responsibilities, and explore how classes collaborate to support the behaviours identified in the use cases. This lightweight design activity aligns with Agile principles by encouraging discussion and shared understanding rather than rigid upfront specification.

Class: Space invader game	
Responsibilities	Collaborators
The current score? State of Player State of enemies Enemy ships Player Lives Move Shoot Take damage Save Win/Lose conditions Current Wave Load game	Player Enemy Level Stage Score State Powerups? Save System

Figure 4 CRC card for the Space Invader Game class, identifying system-level responsibilities and collaborating classes.

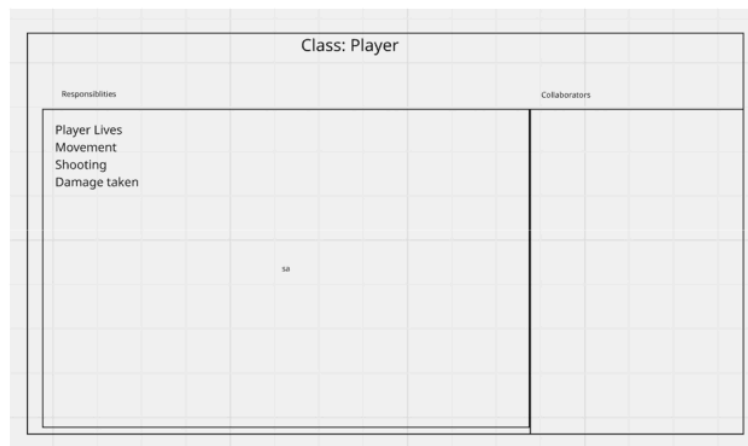


Figure 5 CRC card for the *Player* class, illustrating player-specific responsibilities and interactions within the system.

The requirements and responsibilities identified through use cases and CRC cards were then translated into user stories, forming the basis for estimation and sprint planning activities discussed in the following section. These artefacts provided a structured bridge between high-level requirements and the user stories later used for estimation and sprint planning.

In practice, creating these artefacts revealed a clear trade-off between keeping documentation lightweight which Agile encourages over building enough shared technical understanding among the team. Although use cases and CRC cards required some initial effort and time upfront, they significantly reduced ambiguity when the team later translated requirements into user stories. This clarity made estimation more straightforward, helped the group reach consensus during planning poker sessions, and supported smoother sprint planning overall.

4. Estimation of User Stories and the Use of Burndown Charts

Accurate estimation remains a major challenge in Agile software development, especially in settings where requirements change frequently and uncertainty is high. Rather than trying to forecast precise implementation times, Agile teams focus on estimating the relative effort and complexity of work items. This supports better planning, prioritisation, and decision-making throughout the project.

In Scrum, estimation is typically applied to user stories, which serve as the main units of user-centred functionality intended for delivery within a single sprint.

4.1 Estimating User Story Implementation Time and Difficulty

In Scrum, teams usually estimate using relative techniques like story points or T-shirt sizing instead of absolute units such as hours or days. This fits with the Agile view that requirements at the start are often unclear or incomplete, so trying to pin down exact time predictions is rarely realistic or useful. Relative estimation lets the team compare user stories to each other based on how much effort, complexity, and uncertainty they seem to involve, rather than guessing precise durations [17].

T-shirt sizing is one of the simpler relative methods, it groups stories into broad categories such as XS, S, M, L, or XL. The approach keeps things quick and light, encouraging brief discussion without getting lost in too much detail. It helps the team concentrate on grasping the overall scope and challenges of the work. In estimation sessions often using Planning Poker or similar team members look at things like technical difficulty, volume of work, unknowns, and any dependencies on other parts of the system or external teams. The process relies on open conversation and reaching consensus, which turns estimation into a team activity that builds common understanding and alignment [18].

4.2 Purpose and Use of the Burndown Chart

The burndown chart is a widely used visual tool in Scrum for tracking remaining work during a sprint. It plots the amount of work left, usually measured in story points or ideal hours against time, giving a clear picture of how progress is unfolding. The chart typically shows an ideal burndown line, which assumes steady, linear completion of work, alongside the actual line that reflects the team's real progress [2].

By comparing the actual line to the ideal one, the team and stakeholders can spot deviations early. A line that's flatter or steeper than expected might signal issues like

underestimated stories, unexpected technical challenges, or blockers preventing progress. This kind of visibility supports Scrum's emphasis on transparency and quick adaptation, allowing the team to inspect and adjust before small problems become bigger ones.

That said, burndown charts have clear limitations. They focus purely on the quantity of work remaining and say nothing about the quality, business value, or usefulness of what's been completed. A smooth downward trend doesn't guarantee that the team is delivering the most important features, nor does it reveal accumulated technical debt, bugs, or rework. There's also a risk that teams become too fixated on keeping the line on track, which can shift attention away from delivering real value and towards simply burning down points. For these reasons, burndown charts should always be used carefully, in combination with other sources of insight like sprint reviews, stakeholder feedback, and qualitative discussions [2].

The burndown chart shown below illustrates how remaining sprint work can be tracked over time against an ideal completion trajectory.

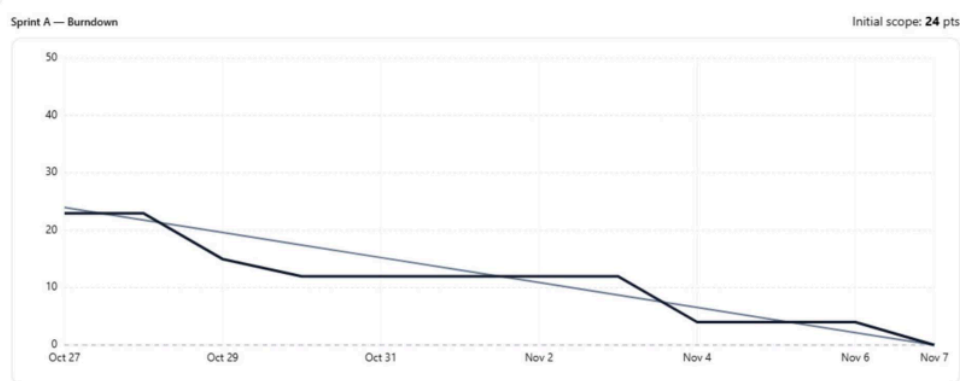


Figure 6 Sprint burndown chart showing planned versus actual progress during a sprint.

4.3 Practical Application of User Story Estimation

During the practical sessions, user stories were estimated using a relative estimation approach. These stories were drawn from the system requirements and use cases developed earlier in the module, then compared directly against each other based on perceived effort, complexity, and uncertainty. Instead of assigning exact time values, the team used relative sizing to encourage discussion and build a shared understanding of the work.

A straightforward estimation technique was used to place user stories into broad size categories that reflected their anticipated implementation effort. This exercise showed that the real value of estimation lies not in achieving perfect accuracy, but in

helping the team develop a common view of the work involved especially useful when requirements are still likely to change or evolve.

This approach was supported through practical exercises in which requirements were refined into user stories and grouped by relative size to support sprint planning.

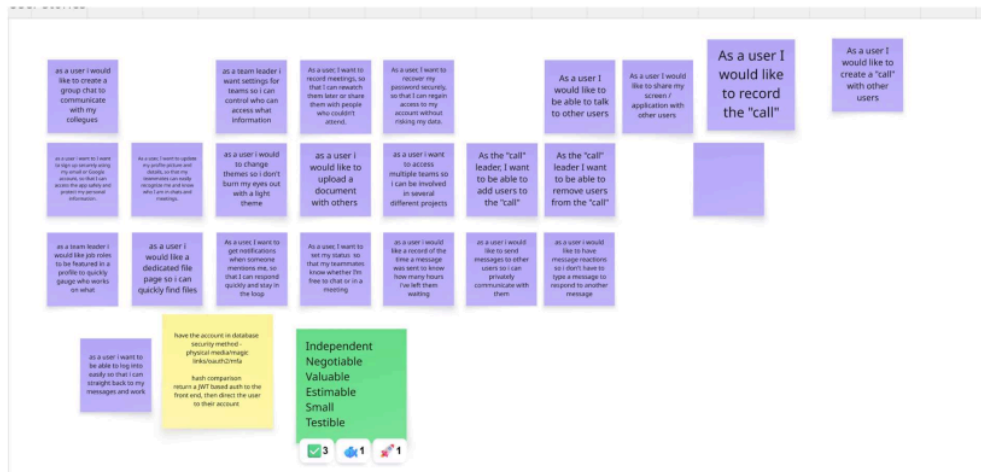


Figure 7 Early grouping and comparison of requirements prior to assigning relative size categories

This exercise reflects how relative estimation techniques such as T-shirt sizing or informal story point comparison are used in practice to support sprint planning decisions.

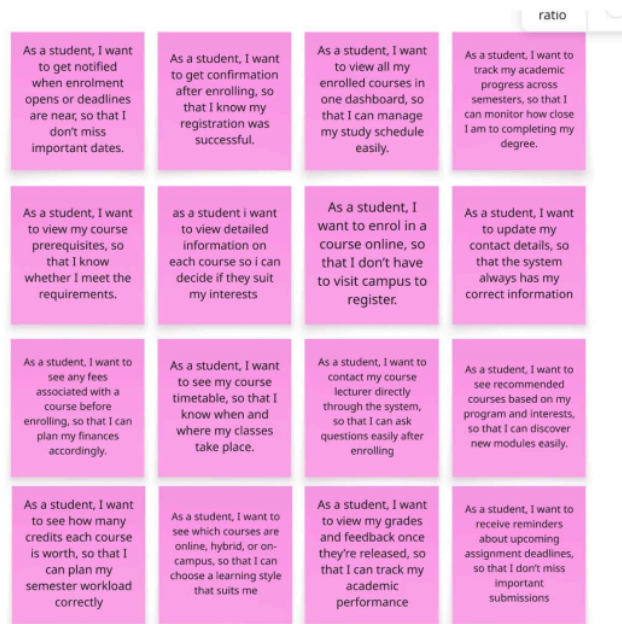


Figure 8 Collection of user stories created during an in-class Agile estimation exercise, capturing functional requirements from a student-centred perspective prior to relative sizing

The set of user stories shown in Figure 8 illustrates how requirements were consistently expressed using the standard “As a [user], I want [feature], so that [benefit]” format. This approach kept every story firmly focused on the end user and their needs, in line with core Agile best practices.

Creating and reviewing the user stories in this format made estimation noticeably more effective. Inconsistencies in scope and any remaining ambiguity stood out early in the process, allowing the team to address them before moving forward. Several stories turned out to be too broad at first and clearly would have taken more than one sprint to complete, which reinforced the need to keep stories small and negotiable right from the start.

This experience mirrors real-world Agile practice, where estimation is never just about predicting effort but also equally about refining and deepening the team's shared understanding of the requirements. Spotting and splitting oversized stories early helps avoid overloaded sprints and keeps planning realistic and productive.

Before moving to estimation and sprint planning, the stories were reviewed against the INVEST criteria to ensure they met the key qualities, Independent (could stand alone without heavy dependencies), Negotiable (open to discussion and refinement), Valuable (delivered clear benefit to the user or business), Estimable (clear enough for the team to size them reasonably), Small (appropriately sized to fit within a

sprint), and Testable (had clear acceptance criteria to verify completion). This check helped confirm the stories were well-formed and ready for effective planning and delivery.

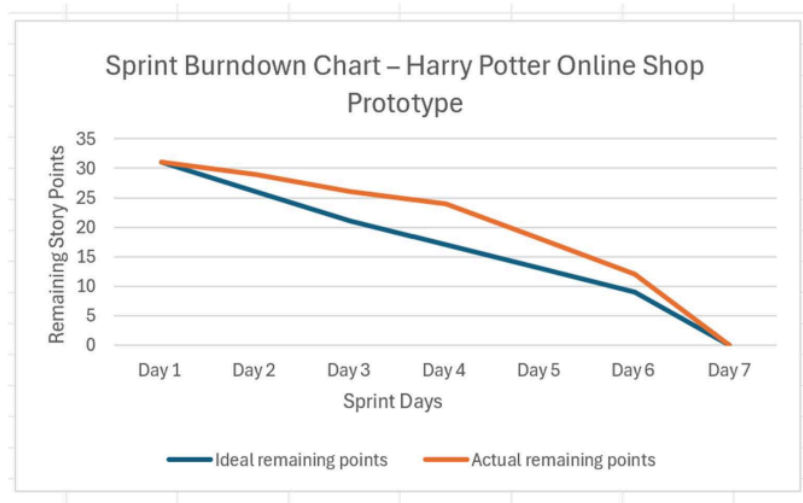


Figure 9 Sprint burndown chart for the Harry Potter online shop prototype, showing ideal versus actual remaining story points across a seven-day sprint based on an initial sprint backlog of 31 story points.

Figure 9 presents the sprint burndown chart for the Harry Potter online shop [19]prototype, illustrating the relationship between ideal progress and actual completion of user stories across the sprint.

The burndown chart shows that progress fell slightly behind the ideal line during the early and middle parts of the sprint. This suggests some underestimation of complexity, slower start-up on tasks, or unexpected blockers that slowed initial momentum.

However, the actual line caught up and converged with the ideal line toward the end of the sprint, showing that the team managed to recover well and complete the planned backlog by the close. This kind of pattern, starting slower but accelerating as the team gains clarity and removes impediments, is fairly normal in Agile projects. It highlights how understanding of the work deepens and velocity often improves as the sprint moves forward.

5.DevOps Tools and Testing in a Web Application

To illustrate the application of DevOps practices and testing tools, this section uses the Official Harry Potter Shop [19] as an example of a modern e-commerce web application. The site provides a realistic case study because its main functionality, product browsing, user accounts, basket management, and checkout , is publicly accessible. The discussion stays conceptual and focuses on typical architectural components and testing strategies rather than any branded details.

5.1 Web Application Components Requiring Testing

An e-commerce platform like the Official Harry Potter Shop is made up of several interconnected software components that all require thorough testing to guarantee reliability, security, and usability.

At the frontend level, the user interface needs testing to make sure pages render properly across different browsers and devices, navigation works as expected, and interactive elements such as product filtering, sorting, image galleries, and basket updates respond correctly to user inputs. Problems here can quickly harm the user experience and erode customer confidence.

The backend layer usually provides APIs that manage product data retrieval, user authentication, order processing, and pricing calculations. These APIs must be tested to confirm accurate request handling, strong data validation, and appropriate error responses under various conditions.

The database layer, which holds product details, user information, order records, and stock levels, requires testing to ensure data integrity, correct entity relationships, and stable performance during concurrent access from multiple users.

Finally, integration points with external services especially payment gateways and delivery providers need to be tested to verify that data exchanges work correctly, transactions complete securely, and any failures are managed gracefully without disrupting the overall flow.

5.2 GitHub Actions for Continuous Integration and Deployment

GitHub Actions provides a powerful way for Software DevOps teams to set up continuous integration and continuous deployment (CI/CD) pipelines directly within the repository. In a CI pipeline, any code push or pull request automatically starts workflows that build the application and execute automated tests. This catches defects early in the process and helps stop broken code from moving forward in the development cycle [20].

For an e-commerce application like the Official Harry Potter Shop, a typical CI workflow might involve installing project dependencies, building the frontend, running unit tests on business logic and components, and executing automated API tests or UI tests. If all these steps pass successfully, the CD part of the pipeline can then automatically deploy the updated application to a staging environment for further checks, or with appropriate approvals or conditions straight to production.

This automation enables teams to release updates frequently and with confidence, which is especially valuable for an e-commerce site during busy periods like holiday sales, new product launches, or promotional events, when rapid fixes and new features can make a real difference to performance and user satisfaction [21].

To demonstrate the practical application of this CI pipeline, Figures 10 – 12 present evidence of the implemented GitHub Actions workflow. These figures capture the successful execution of the pipeline after a push to the main branch, showing key stages including environment setup, dependency installation, automated test execution, and the build process. Together, they illustrate how continuous integration can be applied in practice to deliver rapid feedback and enhance overall build reliability.

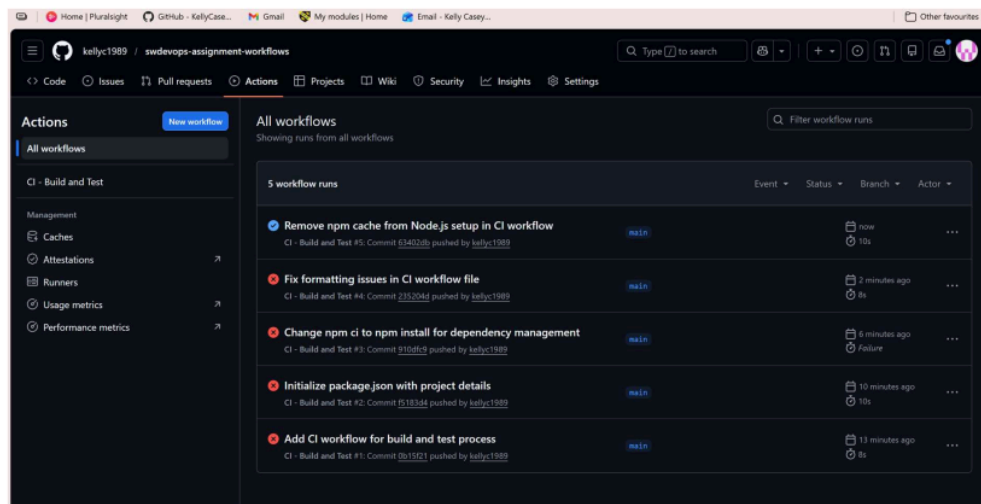


Figure 10 Successful execution of a GitHub Actions continuous integration workflow triggered on push to the main branch.

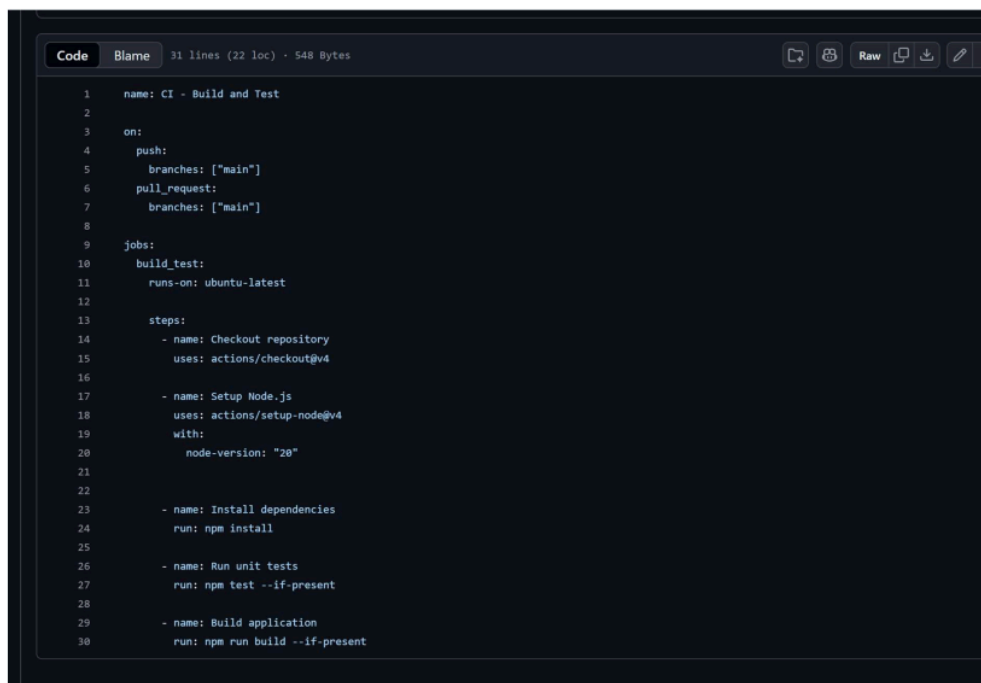


Figure 11 GitHub Actions CI workflow definition used to automate build and test processes.

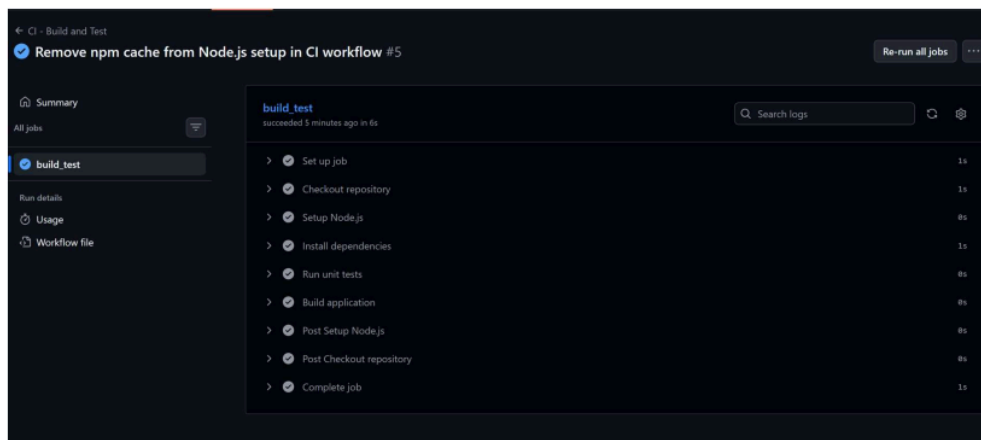


Figure 12 Successful execution of the GitHub Actions CI job showing automated build and test stages

5.3 API Testing Using Postman

API testing is crucial for checking backend services on their own, separate from the frontend interface. Tools like Postman let teams send HTTP requests directly to API endpoints and check key aspects: response status codes, the structure and content of returned data, and how errors are handled. This helps confirm that the backend behaves correctly and stays reliable, even if the user interface is updated or redesigned [22].

For an e-commerce site like the Official Harry Potter Shop, Postman could be used to test important APIs for example, those that fetch product details, handle user login and account management, or process order submissions. Teams can organise these tests into Postman collections, which can then be run automatically and integrated into CI pipelines. This setup ensures API tests are executed consistently every time code changes are pushed, forming a reliable part of the broader DevOps process [23].

5.4 User Interface Testing Using Selenium

While API testing checks the backend logic in isolation, Selenium is used to automate end-to-end testing of the user interface from a real user's point of view. Selenium scripts can mimic typical user actions, navigating between pages, logging into an account, searching for products, adding items to the basket, applying discounts or selecting options, and going through the full checkout process. This type of testing verifies that the entire user journey works smoothly after code changes or deployments [24].

For an e-commerce platform like the Official Harry Potter Shop, UI testing with Selenium is especially useful for catching problems introduced by frontend updates such as broken navigation links, unresponsive buttons, misaligned layouts, or features that fail in certain browsers. Although Selenium tests tend to run more slowly and can be more brittle compared to API or unit tests, they give strong assurance that the application delivers a consistent and reliable experience to actual customers [25].

5.5 Benefits and Limitations of Adopting Software DevOps

The adoption of DevOps practices brings clear advantages for web applications. Automated CI/CD pipelines speed up deployments and make them more consistent, cutting down on human error and allowing teams to release updates more frequently. Continuous testing catches defects early in the process, which improves overall software quality. At the same time, tighter collaboration between development and operations teams builds better visibility, shared responsibility, and a smoother flow from code to production [26].

On the other hand, adopting DevOps comes with real challenges. Setting up and maintaining automated pipelines demands solid technical skills and ongoing commitment from the team. Automated tests can grow complicated or become fragile over time if they're not well designed and regularly maintained. There's also often cultural resistance in organisations accustomed to separate development and operations silos, as shifting to a more collaborative model requires changes in mindset, processes, and sometimes even team structures. Despite these hurdles, when DevOps is implemented thoughtfully and with proper support, it delivers substantial long-term gains in reliability, scalability, and the ability to respond quickly to business or user needs [27].

5.6 Alignment Between Agile and DevOps Practices

Agile development and DevOps practices align closely through their common focus on rapid feedback, collaboration, and continuous improvement. While Agile frameworks like Scrum concentrate mainly on handling requirements, running short iterations, and fostering team collaboration, DevOps extends these ideas into the delivery process by bringing development and operations closer together. Combined, they create a complementary way of working that enables faster and more reliable software releases.

A key area of overlap is the use of automation to support short iterations. Continuous integration and automated testing let Agile teams check changes frequently, keeping each increment stable and ready to ship. This directly supports Scrum's aim of producing working software at the end of every sprint and helps prevent integration problems from building up unnoticed [21]. By tightening feedback loops, DevOps strengthens core Agile values such as responding quickly to change and prioritising technical excellence.

That said, adopting DevOps brings its own challenges. Automation improves consistency and reduces errors, but it also adds complexity like, maintaining pipelines, managing test coverage, and configuring infrastructure all require time and skill. For teams with limited experience or resources, these demands can sometimes slow things down rather than speed them up. This shows an important point, DevOps tools on their own don't automatically lead to better results, they need the right skills, supporting processes, and organisational backing to work well [28].

From a cultural standpoint, DevOps builds on Agile's emphasis on collaboration by promoting shared responsibility for quality and delivery across the entire lifecycle. It breaks down traditional divides between development and operations, leading to greater transparency and collective accountability. However, making this shift can be tough in organisations with long-standing silos or strict role boundaries, which means DevOps adoption is often as much about changing people and ways of working as it is about implementing new tools [29].

In summary, DevOps is best seen as an extension of Agile rather than something separate or competing. When applied carefully, it enhances Agile development by enabling quicker feedback, higher reliability, and true continuous delivery. Success depends on matching the level of automation to the team's capacity and ensuring that the tools and practices fit the organisation's context and maturity.

6. Conclusion

This report has critically examined how Scrum puts key Agile concepts into practice, with a particular focus on short iterations, collaboration, adaptability to change, and shared ownership of quality. By looking closely at Scrum's roles, artefacts, and events, it has shown how the framework offers a structured but flexible way to support these principles in real projects. The flow of user stories from the product backlog through to sprint delivery stands out as essential for keeping the team aligned with user value, and user story mapping has been highlighted as a valuable technique for shaping effective iterations and handling growing complexity.

The report also looked at estimating user stories in Scrum, noting that relative methods like story points and T-shirt sizing are preferred over exact time predictions. This approach fits well with Agile's empirical, adaptive planning and accepts the uncertainty typical in software work. Burndown charts were evaluated as a straightforward visual tool for tracking progress and promoting transparency, though their value depends on using them alongside an understanding of work quality and not in isolation.

In the second part, the report explored DevOps practices through the example of a real-world e-commerce application. It showed how tools like GitHub Actions can drive continuous integration and deployment, automating builds and tests to give fast feedback and boost reliability. API testing with Postman and UI testing with Selenium were presented as complementary methods that help verify backend accuracy and the overall user experience. While DevOps brings clear benefits such as quicker releases, stronger collaboration, and better software quality, the discussion also recognised challenges such as the complexity of setting up and maintaining tools, ongoing maintenance demands, and the cultural and organisational shifts required for success.

The practical artefacts produced such as the user stories, CRC cards, and sprint burndown charts, reinforced the theoretical concepts by showing how Agile and DevOps techniques work in realistic scenarios. User stories kept the focus on user value, CRC cards clarified early design thinking, and burndown charts revealed typical progress patterns like early lags and late recovery. Together, these hands-on outputs bridged theory and practice, highlighting the role of discipline, communication, and iteration in effective delivery.

In conclusion, Agile development and DevOps practices deliver the best results when applied thoughtfully and with discipline rather than followed rigidly. Scrum gives a practical structure for managing uncertainty and delivering value, while DevOps extends this by automating quality checks and deployment processes. Together, they enable teams to build reliable, adaptable software systems that respond effectively to the realities of modern software engineering.

Bibliography

- [1] A. M. Authors, “Agile Manifesto,” Agile Manifesto, 2001. [Online]. Available: <https://agilemanifesto.org>. [Accessed 12 1 2026].
- [2] K. S. a. J. Sutherland, “Scrum Guides,” 2020. [Online]. Available: <https://scrumguides.org/scrum-guide.html>. [Accessed 12 1 2026].
- [3] Atlassian, “Agile,” [Online]. Available: www.atlassian.com/agile. [Accessed 10 1 2026].
- [4] J. Patton and P. Economy, User Story Mapping: Discover the Whole Story, Build the Right Product, Sebastopol, CA: O’Reilly Media, 2014.
- [5] L. Leite, C. Rocha, F. Kon, D. Milojicic and P. Meirelles, “A Survey of DevOps Concepts and Challenges,” *ACM Computing Surveys*, vol. 52, no. 6, p. 127, 2019.
- [6] T. Offerman, R. Blinde, C. J. Stettina and J. Visser, “A Study of Adoption and Effects of DevOps Practices,” *arXiv preprint arXiv:2211.09390*, 2022.
- [7] Atlassian, “Atlassian,” [Online]. Available: <https://www.atlassian.com/agile>. [Accessed 12 1 2026].
- [8] A. Alliance, “Agile Alliance,” [Online]. Available: <https://agilealliance.org/agile101/>. [Accessed 12 1 2026].
- [9] A. Alliance, “12 Principles Behind the Agile Manifesto,” [Online]. Available: <https://agilealliance.org/agile101/12-principles-behind-the-agile-manifesto/>. [Accessed 15 1 2026].
- [10] Atlassian, “Product Backlog,” Atlassian, [Online]. Available: <https://www.atlassian.com/agile/scrum/product-backlog>. [Accessed 15 1 2026].
- [11] Atlassian, “Sprint Backlog,” Atlassian, [Online]. Available: <https://www.atlassian.com/agile/project-management/sprint-backlog>. [Accessed 15 1 2026].
- [12] Scrum.org, “What Is a Sprint Review?,” Scrum.org, [Online]. Available: <https://www.scrum.org/resources/what-is-a-sprint-review>. [Accessed 15 1 2026].
- [13] A. Alliance, “Why Agile Is Not Enough and Why We Need to Expand Our Agile Horizon for Being Agile Truly,” [Online]. Available:

<https://agilealliance.org/resources/videos/why-agile-is-not-enough-and-why-we-need-to-expand-our-agile-horizon-for-being-agile-truly/>. [Accessed 15 1 2026].

- [14] M. Fowler, "Technical Debt," [Online]. Available: <https://martinfowler.com/bliki/TechnicalDebt.html>. [Accessed 15 1 2026].
- [15] T. Dingsøyr, S. Nerur, V. Balijepally and N. B. Moe, "A Decade of Agile Methodologies: Towards Explaining Agile Software Development," *Journal of Systems and Software*, vol. 85, no. 6, p. 1213–1221, 2012.
- [16] G. Lucassen, F. Dalpiaz, J. M. E. van der Werf and S. Brinkkemper, "Improving Agile Requirements: The Quality User Story Framework and Tool," *technical Report / Workshop Paper*, 2016.
- [17] A. Alliance, "Improving Estimation with Story Points," Agile Alliance, [Online]. Available: <https://agilealliance.org/resources/experience-reports/improving-estimation-story-points/>. [Accessed 20 1 2026].
- [18] E. Agile, "Agile Estimation Techniques," Easy Agile, [Online]. Available: <https://www.easyagile.com/blog/agile-estimation-techniques#t-shirt-sizing>. [Accessed 20 1 2026].
- [19] H. P. S. UK, "Harry Potter Shop UK," Harry Potter Shop UK, [Online]. Available: <https://harrypottershop.co.uk/>. [Accessed 20 1 2026].
- [20] I. GitHub, "Understanding GitHub Actions," GitHub Docs, [Online]. Available: <https://docs.github.com/en/actions/get-started/understand-github-actions>. [Accessed 20 1 2026].
- [21] I. Red Hat, "What is CI/CD?," Red Hat, 2025. [Online]. Available: <https://www.redhat.com/en/topics/devops/what-is-ci-cd>. [Accessed 29 1 2026].
- [22] I. Postman, "API Testing," Postman, [Online]. Available: <https://www.postman.com/api-platform/api-testing/>. [Accessed 29 1 2026].
- [23] I. Postman, "Intro to Collection Runs," Postman Learning Center, [Online]. Available: <https://learning.postman.com/docs/collections/running-collections/intro-to-collection-runs>. [Accessed 29 1 2026].
- [24] S. Project, "WebDriver," Selenium Documentation, [Online]. Available: <https://www.selenium.dev/documentation/webdriver>. [Accessed 29 1 2026].

- [25] M. Fowler, "Martin Fowler," The Practical Test Pyramid, [Online]. Available: <https://martinfowler.com/articles/practical-test-pyramid.html>. [Accessed 30 1 2026].
- [26] D. R. a. A. (DORA), "2021 Accelerate State of DevOps Report," DORA, 2021. [Online]. Available: <https://dora.dev/research/2021/dora-report/2021-dora-accelerate-state-of-devops-report.pdf>. [Accessed 30 1 2026].
- [27] M. Gokarna and R. Singh, "DevOps: A Historical Review and Future Works," *arXiv preprint arXiv:2012.06145*, 2020.
- [28] I. Corporation, "IBM Corporation," What Is DevOps?, [Online]. Available: <https://www.ibm.com/think/topics/devops>. [Accessed 29 1 2026].
- [29] Atlassian, "DevOps Tools for Each Phase of the DevOps Lifecycle," Atlassian, [Online]. Available: <https://www.atlassian.com/devops/devops-tools>. [Accessed 29 1 2026].
- [30] V. Mahnič and I. Vrana, "Using stakeholder-driven process performance measurement for monitoring the performance of a Scrum-based software development process," *Elektrotehniški vestnik*, vol. 74, no. 5, p. 241–247, 2007.
- [32] J. Yip, "Martin Fowler," It's Not Just Standing Up: Patterns for Daily Standup Meetings, 2016. [Online]. Available: <https://martinfowler.com/articles/itsNotJustStandingUp.html>. [Accessed 20 1 2026].

kelly casey agile report

GRADEMARK REPORT

FINAL GRADE

77 / 100

GENERAL COMMENTS

You've done great, fair play Kelly! I can tell you've really applied yourself to this! Tremendous effort!

Excellent integration of class exercise artefacts (use cases, CRC cards, user stories)

Strong critical evaluation throughout, not just description

Good practical application of burndown chart with insightful analysis

Professional writing quality and report structure

Effective synthesis of Agile and DevOps

Marks Per Section

Agile Concepts and Scrum Methodology

Section 1 Engagement with Literature 13 out of 16

Section 1 Knowledge and Understanding 12 out of 16

Section 1 Cognitive and Intellectual Skills 6 out of 8

Section 1

Total 31 out of 40

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

Section 2 Total 15 out of 20

SwDevOps Evaluation

Section 3 Engagement with Literature 12 out of 16

Section 3 Knowledge and Understanding 13 out of 16

Section 3 Cognitive and Intellectual Skills 6 out of 8

Section 3 Total 31 out of 40

Overall Total 77 out of 100

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

PAGE 24

PAGE 25

PAGE 26

PAGE 27

PAGE 28
