Master Thesis
Methodology as a Service (MaaS)

Germans Aņikevičs

Faculty of Electrical Engineering, Mathematics and Computer Science

University of Twente

Supervisors

Dr. L. Ferreira Pires
Dr. G. Sedrakyan

Abstract

Platform as a Service (PaaS) platforms have become synonymous with fast time-to-market software development, substantially reshaping how applications are built, tested, and delivered. With a software development process fundamentally different from conventional environments, established software development methodologies are employed without adequate adaptation. Recognising this gap, this thesis introduces Methodology as a Service (MaaS), which is a hybrid software development framework tailored for PaaS development environments. This study demonstrates MaaS through an application case study on the metadata-driven PaaS ServiceNow platform to illustrate its practical application, pave the way for future PaaS-centric software development methodology innovation, and determine if the benefits of a PaaS-oriented software development methodology led to better outcomes compared to non-PaaS oriented methodologies. Evaluation conducted through expert surveys produced exploratory results indicating improvements across software development outcomes, such as collaboration, resource efficiency, end-user involvement, performance, and delivery, when compared to established traditional, Agile, or other hybrid methodologies, paving the way for future research and development of PaaS-centric software development methodologies.

Table of Contents

Table of Contents 3
1 Introduction 6
1.1 Motivation 6
1.2 Problem Statement 7
1.3 Research Questions 7
1.4 Approach 8
1.5 Expert Survey 8
1.6 Thesis Structure 9
2 Background 10
2.1 Platform as a Service 10
2.1.1 PaaS Platforms Type 10
2.1.2 PaaS Characteristics 11
2.1.3 Limitations 12
2.2 Software Development Methodologies 13
2.2.1 Waterfall 13
2.2.2 V-Model 14
2.2.3 Agile 15
2.2.4 SCRUM 15
2.2.5 Rapid Application Development 16
2.2.6 Rugby 17
2.2.7 Continuous Integration, Delivery, and Deployment 19
2.2.8 Hybrid Development Methodologies 20
3 PaaS Software Development Principles 21
3.1 Principle Identification 21
3.2 Benefits 22
3.3 Security Challenges 23
3.4 Gradual Introduction 23
4 Compatibility with Software Development Methods 24
4.1 Traditional 24
4.2 Agile 25

4.3 CI/CD ... 25
5 Methodology as a Service ... 27
5.1 Development Process ... 27
5.2 Planning ... 28
5.2.1 Initiation ... 28
5.2.2 Execution ... 29
5.3 Development ... 29
5.3.1 RAD ... 29
5.3.2 SCRUM ... 30
5.4 Maintenance ... 31
5.5 CI/CD ... 31
5.6 Project Management ... 31
6 Validation ... 32
6.1 Use Case ... 32
6.2 Applying MaaS to ServiceNow ... 33
6.2.1 Project Management ... 34
6.2.2 CI/CD ... 34
6.2.3 Planning ... 37
6.2.4 Development ... 38
6.2.5 Maintenance ... 39
6.3 Applying MaaS to Mendix ... 39
7 Results ... 41
7.1 Survey ... 41
7.2 PaaS and Traditional Software Development Differences ... 42
7.3 PaaS Methodology Compatibility ... 43
7.4 Benefits of a PaaS-oriented Software Development Methodology ... 43
8 Final Remarks ... 50
8.1 Discussion ... 50
8.2 Limitations ... 50
8.3 Benefits ... 51
8.4 Future Work ... 52
8.5 Conclusion ... 52
Acknowledgements ... 54

References 55
Appendix A: Survey Questions 58

1 Introduction

1.1 Motivation

The ever-increasing demand for software solutions applies to every industry. To remain competitive, organizations consistently demand high-quality software to be delivered quickly and efficiently. Amongst others, this motivated the rapid rise and use of cloud computing (CC) platforms, providing Infrastructure as a Service (IaaS), Software as a Service (SaaS), or Platform as a Service (PaaS) capabilities, as shown in Figure 1. With regards to software development, no single CC platform is inherently better than the other. CC platforms offer either a pay-as-you-go or a subscription-based business model, making them cost-effective and accessible to both large and small enterprises [2, 3]. From the customers perspective, the goal is to receive the ordered product as fast as possible and at the lowest cost; from the developer perspective, the goal is to deliver the highest possible value to the customer [1]. Therefore, the optimal choice depends on project requirements, budget, technical level of expertise, and the level of desired control over the software.

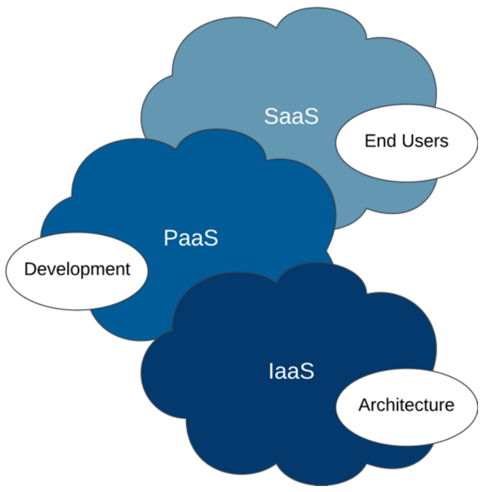


Figure 1: Cloud Computing Models

PaaS stands out in software development because it offers a streamlined and efficient environment for developers. While IaaS produces the primary computing services that allow for the deployment and management of virtual machines, operating systems, and applications [2], it is typically used when customers desire flexibility and full control over their computing resources—for example, to meet fluctuating capacity demands by adjusting computing power or storage [3]. SaaS provides the opportunity to use applications from a software provider that run on a cloud platform, usually accessed through a web application service or a dedicated interface with limited configurability tools [2, 3]. This allows for easy access to ready-to-use software applications without the complexities of installation, maintenance, and updates, thereby reducing the workload on digital talent but also limiting the potential for strategic differentiation.

1.2 Problem Statement

PaaS occupies the essential middle ground by supplying developers with scalable, multi-layer architectures to build, test, and deploy applications on the network. It utilises specialized resources such as ready-to-use functionalities, pre-built components, automated workflows, continuous integration, and continuous delivery (CI/CD) capabilities [2, 3, 5, 6, 14]. This means developers can focus on creating and improving applications without managing the complexity of the infrastructure, providing them with the necessary tools and frameworks to accelerate development and deployment processes. Additionally, facilities offered by PaaS drastically reduce the level of technological literacy required to develop software, which in turn increases the interest towards, and dependence on citizen developers, addressing the disproportionality between the demand for software solutions and the availability of IT talent [4]. Such diverse capabilities, matched with a high variance in technical expertise of developers, require a software development process fundamentally different from traditional (non-CC / non-PaaS) software development.

The methodologies used to manage and structure software development, however, have remained consistent across both traditional and PaaS environments. The same software development methodologies that were initially created for traditional environments are being applied to develop software on PaaS platforms without significant adaptation. For example, Agile remains a dominant methodology in both environments, as its core principles, such as iterative cycles, customer feedback loops, and team collaboration are universally applicable and largely depend on the underlying project. One contributing factor is the relative novelty of PaaS platforms. Academic literature has paid little attention to the impact of PaaS on software development, or its SDLC model [27], in contrast to the inner workings of services, business models, ecosystems, and characteristics of PaaS platforms [6]. In our literature survey, we could not find any scientific sources focusing on a PaaS-centric methodology for the software development process itself. Given that PaaS platforms require a fundamentally different development approach, yet lack a dedicated development methodology, an opportunity arises for innovation by proposing a methodology better suited to the characteristics, development principles, benefits, and limitations of PaaS platforms.

1.3 Research Questions

The goal of this thesis is to propose a hybrid software development methodology, which can support the software development principles of PaaS platforms more efficiently and effectively than their established counterparts and validate it via an application case study. This goal can be translated into the following research questions:

- **RQ 1:** What are the differences in software development principles and requirements between PaaS and traditional software development environments?
- **RQ 2:** Which established or novel methodologies and their components are compatible with PaaS software development principles?
- **RQ 3**: Do the benefits of a PaaS-oriented software development methodology lead to better outcomes (collaboration, performance, resource efficiency, end-user involvement, and delivery) compared to established traditional, agile, continuous, or hybrid methodologies?

1.4 Approach

To achieve our goal, we applied an approach based on Design Science consisting of the problem identification, solution design, demonstration, evaluation, and communication steps, partially adhering to the structures outlined by Peffers et al. and Offermann et al. [7, 8]. This means that not all steps from the defined nominal or iterative process sequences of Design Science could be fully completed due to lack of resources, time, and a dedicated real-world context in which the solution artifact would be evaluated.

Table 1: Source Distribution

Sources	**Topic**
[2, 3, 5, 6, 9, 10, 11, 12, 14, 15, 21, 26]	Principles and characteristics of PaaS platforms and PaaS software development.
[1, 4, 13, 16, 17, 18, 19, 20, 22, 23, 24, 25, 27]	Currently established, hybrid, or novel software development methodologies in the context of PaaS platforms.
[7, 8, 28]	Research methods

The problem identification consisted of both identification and motivation of the problem, which should have practical relevance [7]. Relevance of the problem was achieved through literature review**.** The solution artifact design phase consisted of developing a software development methodology for an application case study geared towards development principles of PaaS platforms, wherein the answers to **RQ 1** and **RQ 2** acted as guidelines in determining its design. To answer **RQ 1** and **RQ 2**, a systematic literature review (Table 1) was conducted to determine the differences in development principles of PaaS and non-PaaS software development environments, as well as differences in existing methods, to produce a PaaS-centric development methodology. This methodology focuses on integrating practises that leverage the specialised resources and capabilities offered by PaaS. The demonstration phase consisted of applying the constructed methodology on a PaaS environment, within the context of a mock software development project, to illustrate how the methodology can be implemented in practice. During the evaluation phase, we conducted expert surveys to assess the benefits, improvements, and effectiveness of the application case study methodology, answering **RQ 3**. Finally, the communication phase involves summarising all findings, discussing limitations, writing this thesis, proposing areas for future research, and drawing final conclusions.

The search for scientific sources was conducted across scientific databases, such as Google Scholar, Scopus, ResearchGate and IEEE, as well as from references of the selected papers. Search queries were performed by using relevant keywords, namely “Cloud Computing Platforms”, “Platform as a Service”, “PaaS”, “IaaS”, “SaaS”, “PaaS Software Development, “PaaS Development Characteristics”, “PaaS Development Principles”, “Software Development Principles”, “PaaS Software Development Methodology”, “Traditional Software Development”, “Software Development Methodology”, “SDLC”, “Agile”, “Innovative Software Development Methodologies”, “Hybrid Software Development Methodologies”, “Design Science”, “Likert Scale”, “Survey Methodology”.

1.5 Expert Survey

The experts were chosen from a pool of actors typically associated in either participating in or facilitating software development processes. Additionally, the chosen experts are expected to possess experience working with both non-PaaS and PaaS software development projects, as well as an understanding of the use, benefits, and application of software development methodologies and lifecycle models. For this thesis, experts with the following backgrounds participated in the survey: technical and functional consultants, software engineers, and product owners or project leads. Additionally, respondents are informally divided into *technical* and *functional* subgroups (Table 2) to explore any differences of perception of MaaS.

Table 2: Survey Expert Groups

Technical Experts	**Functional Experts**
Technical Consulting Specialist	Product Owner
AI Software Engineer	UX Designer
Technical Consultant	Lead App Engine & Creator
Software Engineer	Business Process Optimization Consultant
Full Stack Developer	
Integration PaaS Platform Developer	

1.6 Thesis Structure

This thesis is further structured as follows: Chapter 2 provides the literature review results of, and the theoretical background for, covered topics, such as PaaS platforms and their characteristics, as well as established, hybrid, or novel software development methodologies. Chapter 3 discusses the differences between traditional and PaaS software development and lays out PaaS-centric software development principles. Chapter 4 provides an analysis and comparison of PaaS software development principles and software development methods covered in Chapter 2. Chapter 5 introduces the Methodology as a Service (MaaS) approach and structure, which is a hybrid methodology tailored towards PaaS-specific software development, utilising traditional, Agile, Rugby, and CI/CD practices. Chapter 6 demonstrates MaaS on a mock project on the ServiceNow platform, showcasing phases of MaaS during planning, development, and maintenance activities. Chapter 7 evaluates MaaS based on expert Likert-scale survey responses and additional feedback across collaboration, end-user involvement, performance, resource efficiency, and delivery effectiveness. Chapter 8 presents the discussion of key findings, limitations, recommendations for future research, and the conclusion of the thesis.

2 Background

To give the necessary contextual background to understand the thesis, this chapter introduces the theoretical concepts relevant for this research.

2.1 Platform as a Service

In the category of modern Cloud Computing (CC) services, PaaS platforms support a game changing paradigm, offering their customers facilities to develop, run and manage software with the backing of Model-Driven Development [9] and without the complexity of building and maintaining the underlying infrastructure (Table 3). Different CC service models differ by providing a unique resource as a service and are defined by Wulf et al. [3] as:

- IaaS Offers an environment to host information systems (IS)
- PaaS Provides an environment for IS development.
- SaaS Delivers ready-to-use IS.

PaaS can be described relative of the other CC service models, as it is most often built on top of an IaaS and in the end can be used to produce SaaS products.

Table 3: PaaS definitions

Author	Definition
Shu-Qing et al. [10]	PaaS is a business model in the cloud computing era, which provides a server platform or development environment for developers.
Gass et al. [5]	PaaS furnishes a broad spectrum of elaborate application-level services and offers an execution and development environment on top of a cloud infrastructure.
Singh et al. [11]	PaaS is a virtualized platform that consist of many servers. It comprises a layer of various software and provides it as a service that can be used to make higher-level services.

From an end-user perspective, PaaS applications, or applications developed and supported using PaaS platforms and environments, resemble standard SaaS applications [5], which are accessible if an Internet connection is available. For developers, however, PaaS platforms open a world of possibilities. The entire infrastructure, consisting of hardware, databases, operating systems and their corresponding patches, is available as a paid-for service. Furthermore, when developing software, PaaS offers developers a variety of shared components, such as predefined objects or built-in access and security features that developers can leverage. Finally, development is often supported by wizards and point-and-click features, which might simplify application creation [5].

2.1.1 PaaS Platforms Type

PaaS platforms provide a development environment and can be further differentiated based on the following categories defined by Walraven et al. [12]:

- PaaS platforms that mimic and match the APIs of popular enterprise application servers and middleware platforms. Such platforms are great for developing mobile and web applications that are meant to integrate with existing enterprise systems, or applications which require an extensive use of the mimicked middleware features. Examples include Microsoft Azure using the .NET framework, Oracle Cloud running on top of the WebLogic Server, Red Hat OpenShift based on the JBoss platform, and Cloud Foundry using VMware and Spring technology [12].
- Focused PaaS platforms aimed at supporting specific types of cloud applications. These are known for their supported deployment of highly scalable middleware and storage facilities. Such platforms are typically used for applications that benefit from specific optimizations in performance and scalability, i.e., real time analytics services, social media and e-commerce platforms. Google App Engine and GigaSpaces' XAP Elastic Application Platform belong to this category [12].
- Metadata-driven PaaS platforms, which are similar to focused PaaS platforms, are designed with SaaS applications in mind. These introduce a higher-level composition and configuration interface and are best suited for development of applications where much of the functionality can be configured through metadata, rather than coded from scratch. Examples include ServiceNow, Salesforce, Mendix, WOLF, and TCS InstantApps [12].

The focus of this thesis is primarily targeted at metadata-driven PaaS platforms due to their popularity and accessibility to both citizen and seasoned software developers. Additionally, the configurational support of metadata-driven platforms naturally instils a culture of component reusability, decreases the time spent on the development of existing software features, and decreases the level of maintenance required.

2.1.2 PaaS Characteristics

Most PaaS platforms include a set of shared components, as shown in Table 4, to cover aspects such as security, data management, connectivity, and templating. Built-in security controls ensure the availability of robust role-based access to keep data secure [5]. This allows developers to determine the level of accessibility of users to different parts of the software. In addition, most of these platforms offer advanced data management capabilities that allow the developers to model, access, and change data within their applications [5, 14]. Connectivity is fundamental for creating modern and interoperable applications that can leverage external data and functionality. Finally, a recognizable feature of PaaS is the provision of templates and building blocks, which are reusable components that can be used to speed up the development of custom applications [5].

Another important PaaS characteristic is extensibility, which can be defined into two sets of capabilities: configuration and programming. Configuration capabilities help minimize coding time and achieve consistency among different projects. In contrast, programming capabilities define what can be achieved through custom code [5], which provides flexibility to build very customized and specific features that cannot be achieved through configuration. The development tools offered by PaaS platforms consist of both web-based and local tools. A web-based development environment is usually offered through an integrated development environment (IDE) which can be accessed through a web browser, whereas local tools cater to those who prefer to develop in their local environment, offering advanced features and integrations with other software [5]. Lastly, learnability is an important characteristic, encompassing the required knowledge to effectively use the platform and the provided knowledge through documentation and training materials [5]. This makes PaaS platforms accessible to both experienced and new developers.

Table 4: PaaS characteristics as defined by Gass et al. [5]

Feature	Characteristic	Description
Shared components	Access and security controls	User management and user rights management
	Data management capabilities	Features to model data and capabilities to access and modify data
	Platform connectivity	Support of protocols and preexisting connectors to integrate external services
	Templates and building blocks	Platform objects that can be reused for custom applications
Extensibility	Configuration capabilities	What can be achieved just by configuration
	Programming capabilities	What can be achieved with custom code
Development tools	Web-development environment	Functionality and usability of the browser-based Integrated Development Environment (IDE)
	Local tools	Functionality and usability of local tools
Learnability	Required knowledge	Knowledge and previous experience required
	Provided knowledge	Documentation and training material

2.1.3 Limitations

Gass et al. [5] recognized the PaaS capability of removing the cumbersome maintenance and setup responsibilities from developers. However, such a novel approach presents its own challenges, contrasting from those in traditional software development. In their discussion on the transformative potential of cloud-based software development solutions, specifically PaaS platforms, Aydin et al. [9] point out that the transition towards cloud-based development introduces various uncertainties. The abstraction of underlying infrastructure and the multitude of services and tools available can be intimidating considering that the end-user range of expertise is spread between experienced, novice, and citizen developers. Similarly, the issue is extended further as PaaS or other types of CC platforms employ a dedicated platform development programming language [11]. While the language itself can be the same as one used in traditional software development, such as JavaScript, PHP, Java, Ruby, or Python, it is often modified to accommodate its respective platform, using a specific and less publicly known framework. Platforms, such as Mendix and OutSystems, provide a visual development environment which utilises visual modelling as development language. Salesforce development operates on the Apex language, which is similar to Java, but is not used outside of its platform. ServiceNow utilises JavaScript, however, the heavy use of its own proprietary libraries drastically alters the code writing process. This means that even the most experienced developers are required to go through a learning curve when switching to a PaaS development environment. Cloud based IDEs often lack certain features compared to their desktop counterparts, such as comprehensive debugging tools, performance analytics, and modelling capabilities [14]. Quality assurance of an application produces a large set of challenges, as it can depend on external services [13]. Performance is often an issue as optimization requires a deep understanding of the platform inner workings and limitations. Application scalability is difficult to maintain, as the developers have no knowledge of the potential changes a future release may bring. Any customization or other deviation from out-of-the-box (OOTB) configuration automatically puts the application at risk. Similar issues were identified by Gass et al. [5], stating that vendors fail to understand and often understate potential productivity impediments. The complexity of PaaS platforms is difficult for newcomers to grasp due to its crucial technical and non-technical information being scattered across various sources.

2.2 Software Development Methodologies

Software development methodologies are systematic approaches that guide the planning, execution, and management of software projects. These methodologies provide frameworks for organizing tasks, coordinating teams, and delivering software products that meet customer requirements. Over the years, a variety of methodologies have emerged, each reflecting different philosophies and designs to address specific challenges inherent of software development. To help answer **RQ 2**, we lay the theoretical foundations with examples from traditional, agile, continuous, hybrid, and novel software development methodologies. We include popular models based on the traditional (sequential), prototyping, and iterative SDLC philosophies.

2.2.1 Waterfall

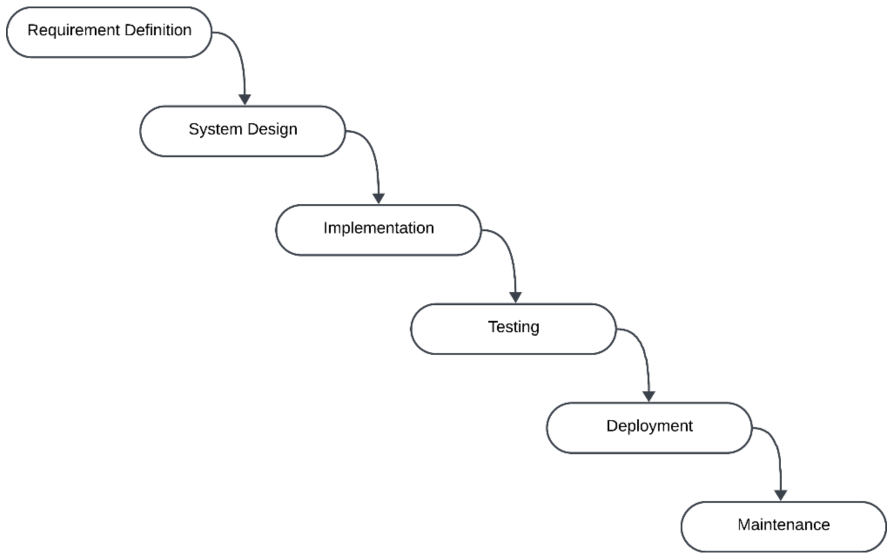


Figure 2: Waterfall model

Traditional software development methodologies, such as Waterfall (Figure 2) or V-Model (Figure 3), are referred to as heavyweight methodologies due to their structured sequential approach to software development. These are some of the older methodologies and thus are coined as traditional yet are still incredibly popular to this day [23]. Waterfall stages can be broken down to requirement definition, system design, implementation, testing, deployment, and maintenance [13]. Each stage must be completed before the next. Characterized by their adherence to predefined processes and an emphasis on extensive documentation, this approach facilitates an initial clarity regarding project expenses, timelines, and the distribution of resources [13]. However, it simultaneously poses significant challenges to integrate modifications during the ongoing phases of development. The efficiency of projects orchestrated under such methodologies depends on a deep understanding of all requirements prior to the commencement of the development phase. Waterfall is most appropriate for projects in which the requirements are stable and are not subject to frequent change, specifically during the development and implementation stages [16].

2.2.2 V-Model

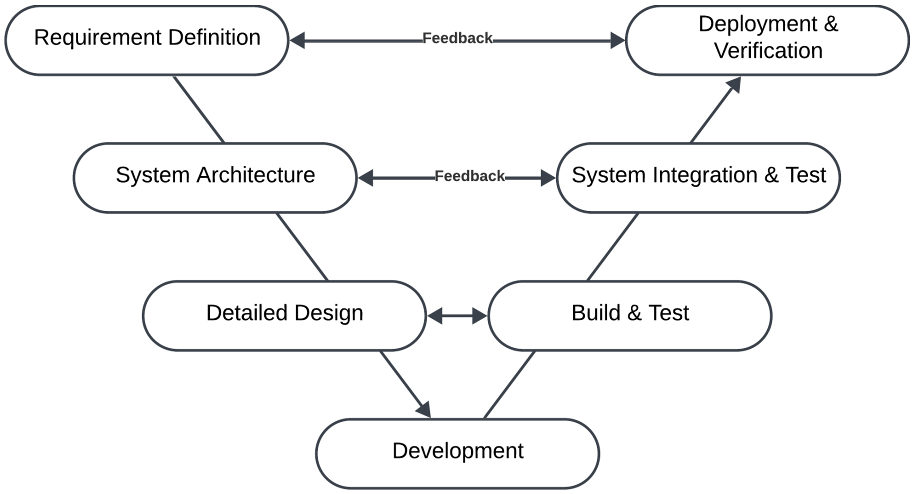


Figure 3: V-Model

The NASA developed V-Model (Figure 3), which is a variation of the Waterfall model, is represented by a V shape folded in half at the lowest level of decomposition [17]. The left leg consists of sequential phases, following a top-down approach, in which user requirements evolve into even smaller components through the process of decomposition and definition, until the development phase is reached. In parallel, the right leg consists of sequential phases, following a down-up approach, in which decomposed user requirements are integrated, tested, and verified into successful levels of implementation and assembly [17]. The models symmetric and structured approach assists in early detection of issues, as well as their timely resolution. It ensures that each development phase is paired with a corresponding testing phase, emphasising on validation and verification. This makes the method highly suitable for large and complex projects, including those with varying internal and external stakeholders, as alignment of each party at each phase is required before advancing to the next.

2.2.3 Agile

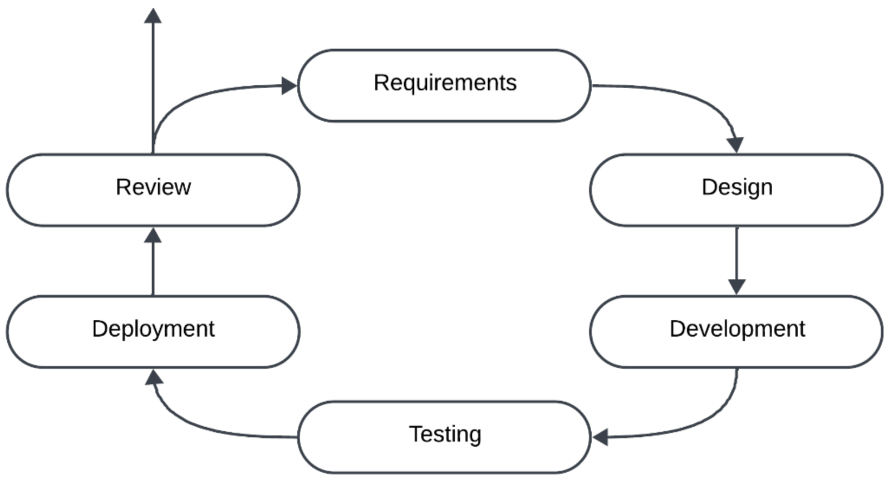


Figure 4: Agile model

The dynamic nature of modern software projects demands an increasingly greater level of responsiveness and flexibility, as opposed to the structured and linear approaches employed by traditional software development methodologies, such as Waterfall and the V-Model. As a result, the Agile philosophy emerged rooted on iterative and incremental development, where requirements and solutions evolve through collaboration between self-organising, cross-functional teams [13, 20]. Agile is distinguished by its repetitive and progressive character, emphasizing adaptability, teamwork, and client satisfaction during the development phase (Figure 4). This approach utilises self-directed, multifunctional teams to navigate through shifting demands and solutions through continuous communication, while retaining flexibility [13, 20]. This is achieved through iterative delivery of small, functional increments of software, in contrast to traditional models like the waterfall model, where software is delivered after comprehensive development cycles [17].

Agile itself is not a development method, but rather a project management philosophy based on a set of values outlined in the findings of Rohil et al. [13]. Actual development frameworks, such as RAD (Rapid Application Development) or SCRUM, are defined using the Agile philosophy. For example, the key aspects of RAD resolve around iterative development, user involvement, and demonstrable deliverables, aligning well with the demands of modern fast-paced software development. By compressing the phases of analysis, design, build and test into short, manageable iterations, the development teams are enabled to quickly adapt to changes. Furthermore, the ability to incorporate user feedback becomes possible from the early stages of development, increasing the chances of the product to meet user needs and preferences.

2.2.4 SCRUM

SCRUM is an Agile methodology designed to facilitate collaboration on complex projects in software development. Characterised by its flexibility and adaptability, it breaks down a product into small and incremental builds, which are then delivered in time-boxed iterations. Every iteration involves cross-functional teams working on planning, requirement analysis, design, implementation, and testing. Such iterations are called sprints and typically run for two to four weeks depending on the underlying projects requirements. While Agile is a philosophy with a set of principles, SCRUM provides a structured way for implementing them through well-defined practises, regularly delivering usable product increments. It is particularly effective in projects with constantly changing or unclear requirements, making it suitable for environments which benefit from regular feedback and rapid adaptation. Additionally, SCRUM was found to improve productivity in software projects, as well as positively impact customer satisfaction, quality, team motivation, and cost reduction [19].

The SCRUM methodology defines artifacts and events, which promote efficient project management. The product backlog is a centralized list of all anticipated or desired work to be completed throughout the project. The sprint backlog is a subset of items from the product backlog which are selected for implementation for the duration of the ongoing sprint. The sprint planning is a meeting where the team plans and selects from the backlog the work for the upcoming sprint. Stand-ups are daily team meetings, typically lasting no longer than 15 minutes and performed at the beginning of the workday, where the team synchronises on its progress and sets a plan for the rest of the day. The sprint review is held at the end of a sprint to inspect and review the completed increment and adapt the product backlog if required. The sprint retrospective is a meeting for the team to reflect on the progress of the previous sprint and identify improvements for the next one. For these cross-functional teams, artifacts, and events, SCRUM defines specific roles, each with their own distinct responsibilities. The product owner represents the stakeholders and is responsible for navigating, planning, managing, and maximising value of the product backlog. The scrum master acts as a facilitator to the team, ensuring that the SCRUM principles are adhered to, as well as removing any obstacles that might impede the sprint progress. Finally, the development team is a cross-functional group of professionals who design, build, and test the product increment during each sprint [20].

2.2.5 Rapid Application Development

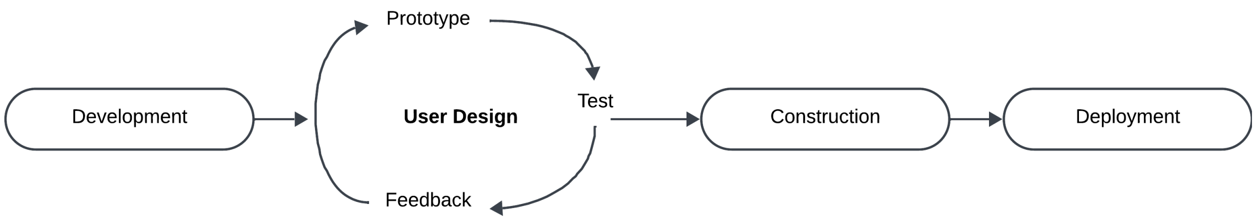


Figure 5: Rapid Application Development

Like SCRUM, RAD is an Agile methodology, initially developed as a response to the rigid and slow pace of traditional methodologies, like Waterfall or V-Model. It emphasizes quick and iterative release cycles, user involvement, and adaptability [17]. While SCRUM is a more detailed and better structured method with dedicated roles and events, RAD is less systematic and focuses on rapid prototyping and user feedback. Similarly to SCRUM, it operates on iterative development cycles and promotes self-organising teams, however, it does not define any specific team roles or events. Instead, it relies heavily on collaboration between developers and users [17]. RAD is typically employed when time-to-market and user feedback is paramount, for example, when working on projects with systems or applications that heavily rely on user interaction.

The process of RAD consists of four steps: requirement definition, user design, construction and deployment. During the requirement definition, RAD already sets itself apart from traditional software development methods, as it does not require a detailed list of specifications; rather, it asks for a general goal or key features, as well as any other business needs, to determine the project scope. The user design phase is where the prototyping and testing takes place. Developers create prototypes with different features and functions as fast as they can within the context of the broader goal or feature. This is performed in an iterative manner like that of SCRUM, where feedback is collected at the end of every cycle allowing the client to decide on what to keep and what to dismiss. During the construction phase, the accepted user design is then finally iteratively implemented by the development team, where prototypes are refined into working models. Lastly, the cutover phase, consists of final system testing, user training, and the subsequent deployment of the developed product.

2.2.6 Rugby

Krusche et al. [18] address the challenges faced by project-based organisations by presenting their novel Agile process model, Rugby, with the objective of improving software delivery processes through integration with continuous delivery practices to streamline the delivery of software updates. The effectiveness of Rugby was evaluated through its implementation in two large university courses with 100 participants working on 10 projects each year in collaboration with industry partners, closely simulating a real-world scenario [18]. To understand Rugby, one has to understand its environment and process model.

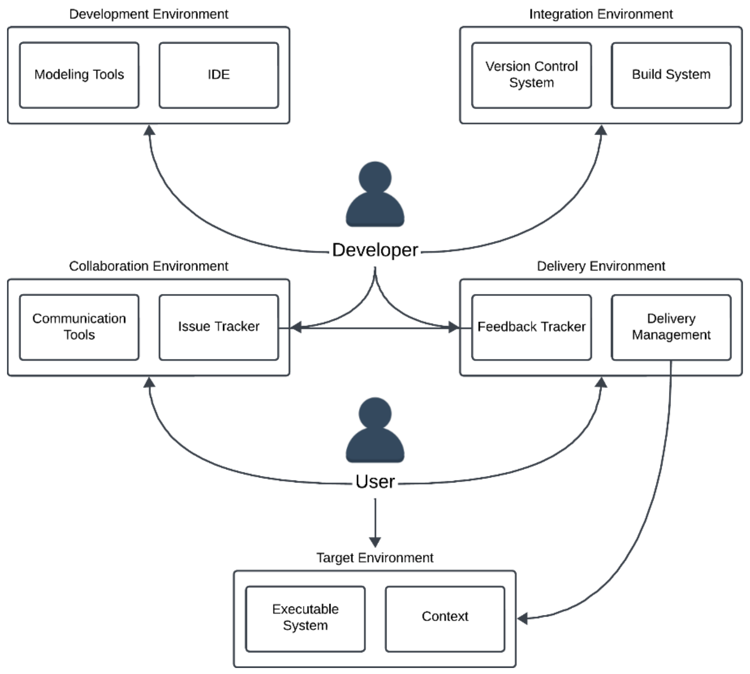


Figure 6: Rugby Eco-System (adapted from [18])

The Rugby eco-system is split in five environments, each with their own dedicated functions: development, integration, collaboration, delivery and the target environments. The project team, consisting of up to eight developers, a team leader and project leader, is expected to be self-organising, cross-functional and therefore responsible for all aspects of development and delivery of software [18]. The project team, as well as the end-user, interact with different environments based on their project responsibilities, as seen in Figure 6. An emphasis is put on the collaboration and delivery environments as per the principles of continuous delivery – a consistent communication and feedback loop between the developer and the end-user. A user is notified from the delivery environment if a new release is available and can then use the software in their target environment. Feedback of the user is stored in the delivery environment and then forwarded into the collaboration environment, i.e., as feature requests. A user can also vote certain features in the collaboration environment [18].

Rugby presents a unique combination of collaboration, flexibility and delivery of quality software, which can be characterized through principles and practices defined in the Rugby process model. During the sprint planning, teams outline visionary scenarios to create backlog requirements for the upcoming sprint [18]. This allows the customer to then choose the requirements to work on while providing sufficient detail for developers to begin work. A key difference when comparing to SCRUM is that Rugby allows requirements to be discussed and expanded during the sprint due to its event-based releases, utilising the continuous delivery workflow [18]. Additionally, Rugby proposes weekly meetings rather than daily, as it anticipates part-time developers to be present on the project [18]. In Sprint 0, lasting two to four weeks, teams focus on building unity, familiarising themselves with release management techniques, such as version control, continuous integration and continuous delivery, as well as acquiring necessary technical and functional knowledge depending on the project problem statement [18].

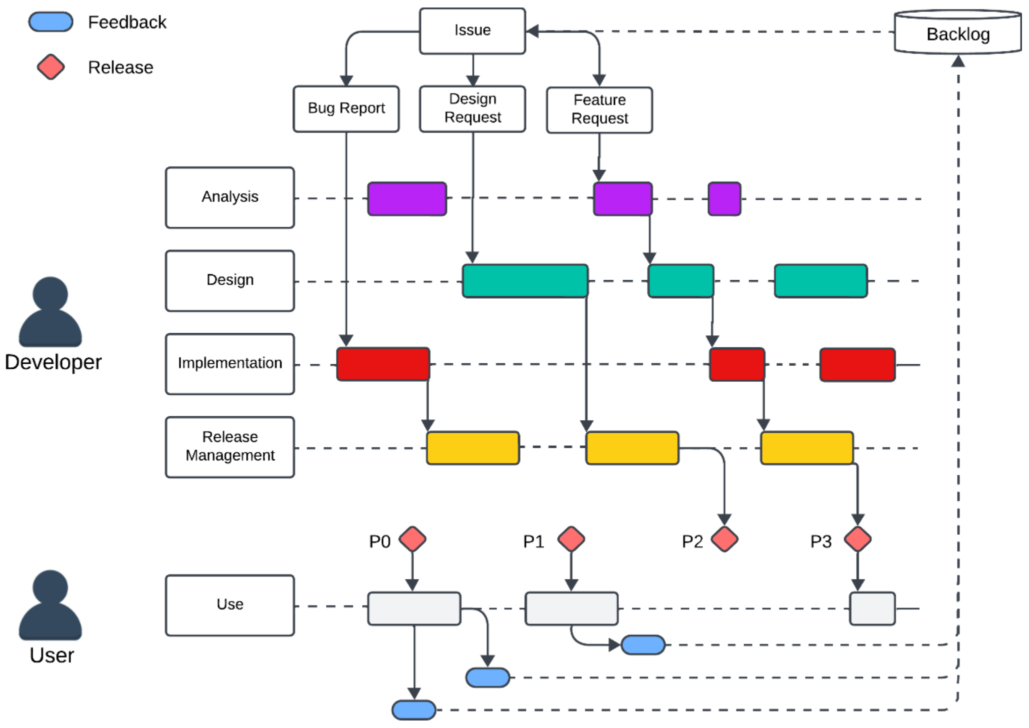


Figure 7: Rugby Feedback Cycle (adapted from [18])

The goal of Sprint 0 is to create an initial empty time-based release to demonstrate the availability of release management and feedback capabilities [18]. The subsequent sprints also maintain a two-to-four-week time frame depending on the requirements, where each sprint is aimed at producing a potentially shippable product increment. Releases in Rugby are vital as they facilitate communication between project stakeholders. While teams are expected to deliver at least one time-based release at the end of each sprint, they are also motivated to release their software whenever communication or feedback is required, or when it is requested by the manager or customer [18]. Upon receiving a new release, users (customers) begin interacting with the new product increment to provide feedback, which subsequently, is added to the backlog. Feedback is categorized into feature requests, design requests and bug reports, which are handled by the analysis's workflow, the design workflow and the implementation workflow, respectively. The release manager of the team is responsible for deciding when and to whom the build should be delivered. A way developers can communicate and inquire for specific feedback is through release notes of a product increment [18]. This continuous feedback loop is present throughout all sprints and leads to iterative development.

Preliminary results after the use of the Rugby model indicated an increased frequency and quality of interactions between the developers and the users, which in turn led to a higher accuracy in delivering the desired requirements, as well as their quality, suggesting the effectiveness of consistent developer-user communication. Furthermore, it enabled the ability to include requirement revision and manipulation during an ongoing Rugby sprint, rather than at the end of one [18]. Ultimately, the results displayed that Rugby's integration of version control, continuous integration and continuous delivery significantly improved collaborative development and development speed [18].

2.2.7 Continuous Integration, Delivery, and Deployment

Continuous integration involves several critical steps aimed at maintaining code quality and facilitating collaborative development efforts. The process is centred around source code management, requiring every team member to commit their changes to a centralized repository upon completing tasks or making new changes. Continuous delivery extends CI by automating the entire software delivery process, ensuring that code can be delivered at any time. This includes the automation of code testing, its integration into shared repositories, builds, and acceptance testing. Once these steps are complete, a manual action is required to deploy to production. Continuous Deployment (CD) is the automation of the release deployment to production after passing predefined tests.

This approach is highlighted as a fundamental aspect of Agile and efficient software development practices [21]. The practice of CI and CD is emphasized as a vital strategy for organizations aiming for frequent and reliable updates to their projects or products [22]. Benefits of transitioning towards a CI/CD pipeline (Figure 5) includes faster release cycles, increasing the rate at which the product is passed on to the hands of the user. Reduced risk is naturally achieved through the continuous release of updates and features. A higher quality in a product is achieved through continuous delivery of small updates, increased developer-user interaction, fewer bugs, and reduced occurrence of issues. For these reasons, CI/CD remains the main toolset within the general practice of DevOps, which is a philosophy encompassing broad range of practices, including culture, collaboration, automation, delivery and continuous improvement of software.

The biggest issue that comes with a transition towards CI/CD is the cultural change, specifically for organisations coming from a traditional methodology background, as the drastic practical shift might require that the staff is retrained, including the managers. The task of building an automated code repository is simply complex as it is, which in turn requires a testing suite of the highest integrity. In fact, integrity is a necessary condition of success, as without an appropriate testing suite, CI/CD introduces more risks than benefits if the tests are not effective.

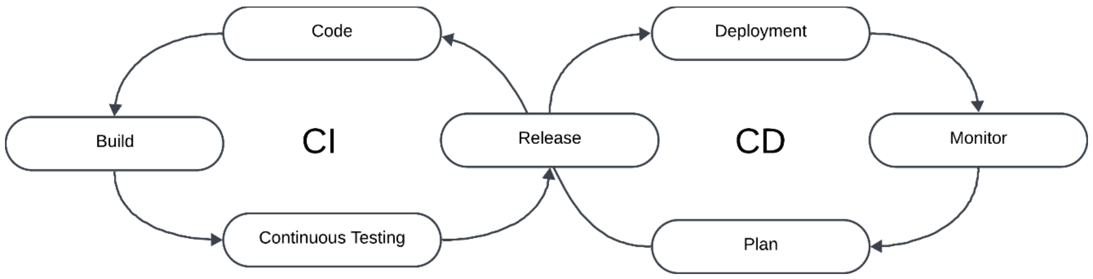


Figure 8: CI/CD Model

2.2.8 Hybrid Development Methodologies

Apart from the choice between Agile and traditional, hybrid methodologies are also prevalent and popular in modern projects. Hybrid development methodologies are manifested dynamically through blending of elements from different software development methodologies depending on the underlying project and its requirements. Such approaches are common because no single methodology can appropriately accommodate absolutely all project needs [24]. In fact, in their research, Vijayasarathy et al. [23] found that over 45% of software development projects ran on a hybrid methodology. The most common approach is the blend of traditional and Agile practices [24]. Hybrid approaches typically emerge organically, rather than through a formal process improvement program, where the top motivating factors include project/product management and commitment, evolution and pragmatism, and project operation and improved flexibility [24]. Hybrid approaches are seen as a way towards more stable yet flexible projects, where the stability of established traditional methodologies aims to appeal towards both management and customer commitment, and the flexibility of agile methodologies provide the necessary flexibility to the developer teams [24]. Lastly, hybrid approaches are used regardless of company size or industry sector [24].

An example of an applied hybrid methodology is the development of a HRIS (Human Resource Information System), where Singhto et al. [16] combined both waterfall and SCRUM methodologies. The planning and the analyses phases of the project were conducted under the waterfall model, defining the business process while assisting new developers in understanding the software development lifecycle [16]. Most of the design and the development phase are conducted according to SCRUM, breaking down requirements into stories and consequently into iterative SCRUM sprints.

Another example is the blend of SCRUM and extreme programming (XP), where Mushtaq et al. [25] address the gap between the two popular agile methods by creating their own novel hybrid model. While SCRUM is used as a lightweight, flexible, and adaptive project management framework, it provides little to no guidance on the engineering aspects of software development [25]. The hybrid model retained the iterative and incremental approaches of SCRUM, using sprints and SCRUM events, such as stand-ups, sprint reviews, sprint plannings, and sprint retrospectives. Engineering practices of XP were integrated to ensure higher code quality, specifically concepts such as simple design, pair programming, continuous integration, and test-driven development [25]. Through the incorporation of XP practices, development teams were guided not only by iterative planning and review mechanisms of SCRUM, but also by systematic coding standards, as well as early and frequent testing.

3 PaaS Software Development Principles

Traditional software development has evolved over decades, shaped by numerous methodologies and engineering practices. In these environments, development teams manage the full technology stack, requiring infrastructure set-up, middleware maintenance, data handling, and application deployment. This approach, predating cloud adoption, is typically characterized by manual, heavyweight processes and environments. Over the years, various software development methodologies, from the structured phases of the Waterfall model to the iterative cycles of Agile, have attempted to tackle these issues. Despite this, maintaining and updating physical infrastructure, integrating tools, and enforcing consistent development standards often remains a time-consuming endeavour. Additionally, in contrast to PaaS platforms, traditional development environments frequently lack built-in support for rapid scalability, security features, and on-demand resource allocation. As a result, the development cycle can become too prolonged with teams devoting significant effort to non-development related tasks. This drastically delays the time-to-market, as well as places operational demands on software development teams that might otherwise be focused on delivery. This chapter answers **RQ 1**.

3.1 Principle Identification

Although established practices and theoretical frameworks exist for traditional development, scientific literature on software development or SDLC on PaaS platforms is scarce [27], even more so with regards to defining concrete principles and instructions on how the software development process must be conducted, or which methodologies should be used. Instead, the literature outlines the structure, capabilities, and limitations of PaaS platforms, as well as their comparison with each other for different types of software projects

Table 5: PaaS Development Principles [3, 5, 6, 9, 10, 12, 26, 27]

Principle	**How**	**Why**
Leverage built-in services	Use platform-provided services like databases, security features and integration connectors	By using pre-built services, developers can avoid the complexity of re-inventing the wheel or setting up and maintaining these components themselves, reducing error rates and reducing time-to-market
Adopt iterative and customer-centric development	Develop in short iterative cycles, which include building, testing, feedback and refinement	PaaS platforms support rapid deployment by providing the ability to quickly prototype ideas into real solutions
Encourage self-organisation	Foster a development culture where the choice of technology, tools, features and deployment is performed at a team level	The abstraction of infrastructure management promotes self-organisation by granting teams more creativity to focus on application logic and user needs

Utilise extensibility	Make full use of configurational and customizable capabilities	Provision of configurational tools and programming interfaces allow developers to configure and customise applications effectively
Prioritize security	Integrate security controls and compliance checks early in the development process	By leveraging the built-in security features of PaaS platforms, ensuring applications are secure by design
Build knowledge	Continuously update skills, knowledge, and utilise the platforms developer community	The ever-evolving nature of PaaS platforms and their technology requires ongoing learning to stay on par with best practices, ensuring optimal use of the platform and its features
Design for scalability	Define application architecture with scalability in mind	Maximize performance and reduce cost by utilising PaaS scalability features, such that applications can handle varying loads efficiently

To provide actionable development guidelines or principles for software development (Table 5), we focus on tangible benefits, such as fast time-to-market, customer satisfaction, high security, improved maintenance and adaptability.

3.2 Benefits

The largest advantage for most enterprises is the speed with which new applications can be launched to catch the next wave of business opportunities, as various cloud-integrated tools help developers focus on rapidly prototyping, building and deploying applications [5, 26]. In traditional software environments teams need to set up, maintain, and update their servers, operating systems, as well as middleware. In contrast, PaaS platforms abstract away most of the infrastructure layer, offering preconfigured environments. Additionally, considering the PaaS platform is a product in of itself, it is continuously updated and improved by its parent organisation. Furthermore, PaaS provides dynamic scalability, where on-demand resource allocation can be utilised to either scale up or down, whereas in traditional environments this process would be completed manually by either adding additional machines or upgrading existing resources, all of which require a degree of downtime and potential hardware procurement. By relying less on infrastructure and manual procedures, teams have the freedom to make decisions regarding deployment, coordination, and technology usage [6]. This independence creates an environment that is more adaptable and responsive to changes, allowing for efficient implementation. The ability to rapidly prototype, test, and iterate based on feedback from external stakeholders greatly enhances the learning process for all team members. It empowers teams to refine their understanding of requirements and address issues in time, fostering a cycle of improvement. Embracing PaaS streamlines the development process and empowers teams to swiftly adapt to evolving needs and innovate more effectively. The reusability of software components and various elements enhance efficiency, as well as reduces the time and effort required to develop new applications [5]. Additionally, developers appreciate the ability to implement custom code solutions through traditional programming capabilities whenever a particular use case is not supported by the capabilities of the platform [5]. The availability of comprehensive documentation and a supportive community that aids knowledge acquisition is also noted [5]. Such tools and technologies boast advantages that ensure a high level of quality and consistency, allowing developers to concentrate on core functionalities and features, thereby improving innovation opportunities and accelerating development speed [26].

3.3 Security Challenges

The level of technical expertise of developers in PaaS software development is typically lower than that which is required in traditional environments. Due to the extensive availability of tools and configurational interfaces, citizen developers are empowered to participate in the delivery of applications. This advancement further reduces time-to-market, as well as the reliance on traditional IT resources, however, it also introduces a plethora of potential security challenges, which must be addressed to mitigate risks [4]. Some of the risks include [4]:

- Poorly defined security requirements
- Insecure coding practises or component configuration
- Insufficient testing
- Exposing sensitive data
- Deployment of insecure components or services
- Vulnerabilities introduced through updates

While not enough on their own, PaaS platforms provide numerous security related features to help developers avoid risks. These include pre-built authentication, role-based access control, encryption, standardized or component-based code generation, compliancy checks, CI/CD pipeline, environment isolation, extensive dashboarding and reporting, service-level agreements, and modern UI frameworks. To achieve the best result, these features should be leveraged in combination with a security framework, such as RAAFT [4], to ensure all crucial risks are accounted for. All these features would require manual creation or integration with existing tools and technologies when manifested within a traditional software development environment, requiring high level of developer expertise.

3.4 Gradual Introduction

Lastly, before considering development, it is important to note that while implementing PaaS might seem like an easy solution to changing business realities, it also entails inevitable technical and sometimes regulatory challenges. Cohen [26] suggests that the best approach for established enterprises that might be considering PaaS is to add it initially as an alternative to its current array of technologies and gradually introduce PaaS to developers by replacing existing legacy tools and systems [26].

4 Compatibility with Software Development Methods

By now we have established that PaaS platforms drastically simplify the development pipeline and provide a vast set of capabilities for software developers through the provision of model-driven development and cloud-based services [5]. The unique characteristics of PaaS (Table 4) naturally enable rapid prototyping and iterative development cycles, allowing developers to produce applications faster, than in traditional environments. Built-in tools and shared user environments allow development teams to directly communicate and collaborate with project stakeholders. Additionally, we have analysed traditional, iterative, continuous, prototyping, and hybrid development methodologies for their strengths and weaknesses, as well as their applicability in different software development contexts. This chapter aims to bridge the gap between the characteristics and development principles of PaaS, as well as the different methodologies discussed in Chapter 2, to answer **RQ 2**.

The selection and usage of a methodology primarily depends on the project scope, requirements and the underlying set of tools or technologies available for development. Additionally, the choice is influenced by circumstances, such as team size and organizational revenue [23]. Agile or iterative methodologies are favoured by organizations with moderate revenues and smaller employee counts, prioritising flexibility and adaptability; however, pure Agile approaches may struggle to provide the structure needed for large projects [23], in which the complexity renders unclear requirements, unplanned outcomes, missed deadlines, or prolonged prototyping simply unaffordable. Traditional methodologies, such as waterfall, are more common in larger organizations with higher revenues; however, are too rigid to leverage the iterative and dynamic nature of PaaS [23]. *Hybrid methodologies, on the other hand, are applicable across various organizational sizes and project types* [23, 24]. Table 6 outlines the general applicability of software development methodologies with PaaS development principles (Table 5). Principles "Utilise extensibility" and "Build knowledge" are not included as the means to achieve them are subjective and highly dependent on parent organizations circumstances, such as the scope of the project, project complexity, budget, staff training, and technologies used.

Table 6: Methodology Compatibility with PaaS Development Principles

	Traditional	**Agile**	**CI/CD**
Leverage built-In services	Medium	Strong	Strong
Iterative and customer-centric development	Weak	Strong	Strong
Encourage self-organisation	Weak	Strong	Strong
Design for scalability	Medium	Medium	Strong
Prioritize security	Strong	Strong	Strong

4.1 Traditional

Traditional methodologies, such as Waterfall (Figure 2) or the V-Model (Figure 3), emphasize linear and sequential approaches. These measures excel at defining the requirements and scope of the project, as well as mitigating risks, manifesting themselves as valuable qualities during the initial stages of a PaaS software development project. However, traditional methods fail to promote a culture of self-organisation, as the choice of technology, tools, and features would have to be predefined before the stages of implementation, testing, or deployment are ever reached. The adoption of customer-centric and iterative development, in which teams develop, build, test, gather feedback, and refine user requirements in short iterative cycles, misaligns with traditional approaches, as one stage must be completed before the next is reached. In this case, the main advantage of PaaS, namely the ability to rapidly prototype, build, and produce software, is ignored. Furthermore, communication and collaboration between development teams and stakeholders is non-existent, as are the possibilities to implement any feedback once implementation has started.

4.2 Agile

Agile models prioritise adaptability, incremental delivery, and continuous stakeholder communication and collaboration. IaaS and PaaS layers of CC offer support through rapid provision of development environments and cloud interfaces to facilitate software deployment [21]. This results in significant benefits to Agile development teams – cloud computing reduces the time and effort required to test and deploy software, therefore the latency between completing development, and receiving feedback from product owners and users, is drastically reduced. Furthermore, frequent communication between team members, both in formal and non-formal environments, created a sense of unity and reduced the time needed to resolve issues [21]. The feedback cycle is reduced further through automation of CI/CD within CC platforms.

Process models, such as SCRUM, RAD, or Rugby, utilise incremental delivery with varying levels of formality. SCRUM defines artifacts such as stand-ups, backlogs, sprints, retrospectives, as well as user roles to complete and provide efficient project management. RAD is less structured, prioritising rapid prototyping, relying on heavy collaboration between developers and users [17]. Rugby proposes a dedicated eco-system split between five environments: development, integration, collaboration, delivery, and target. Depending on project responsibilities, development teams and other project stakeholders interact with these environments. All process models require self-organising and cross-functional teams to operate successfully within an Agile philosophy [13, 17, 18, 20]. Regardless of environment or artifact, an interface or a technology is required to accommodate development teams to complete their project responsibilities. One unique characteristic of PaaS is shared components, which can be defined as interfaces or modules that accommodate access and security controls, data management capabilities, and platform connectivity. Additionally, most PaaS platforms (e.g. ServiceNow, Mendix) provide built-in collaboration tools that facilitate communication and coordination between development teams and stakeholders. If not, then integrations with third party systems, such as Jira, are possible. To summarize, integration of Agile process models into PaaS is particularly powerful, as PaaS platforms are inherently designed to support iterative and incremental development, as well as team and stakeholder communication. Utilising Agile as a development methodology leverages built-in serves, operates in an iterative and customer-centric manner, designs for scalability, and encourages self-organisation, all in line with PaaS development principles defined in Table 5.

4.3 CI/CD

CI/CD is responsible in minimizing the delivery and feedback loop of software, handling testing, integration, and deployment autonomously. CI/CD processes are not only supported but are often integrated as a core feature in PaaS platforms. For example, ServiceNow software development projects typically run on three instances: development, test, and production. Development teams work on a product increment or iteration on the development instance, after which it is packaged, and either manually, or through automation, deployed to the test instance. The customer then tests the release and provides feedback. Separately, ServiceNow provides a native automated test framework (ATF) which allows for automated functional testing. If everything operates without issues, the release then is packaged for deployment into the final target environment – production instance. This whole pipeline can be automated using ServiceNow DevOps plugin which has Git, Azure DevOps, and Jenkins integrated for easier automation. CI/CD concepts are often used in combination with other methods, such as Agile and Traditional, manifesting hybrid approaches depending on project needs [16, 23]. CI/CD pipelines handle complex workflows, automating testing, building, and deployment of software, as well as facilitate rapid feedback via automated tests and shorter development cycles, hence aligning seamlessly the principles (Table 5) of utilising extensibility, design for scalability, leverage built-in services, and adopt iterative and customer-centric development.

5 Methodology as a Service

This chapter describes MaaS illustrating the practical application of a hybrid software development methodology tailored specifically for metadata-driven PaaS platform software development based on the results of **RQ 1** and **RQ 2.**

5.1 Development Process

Relying on the insights discussed so far, we define a development process consisting of three main phases: planning, development, and maintenance. Sequentially completed, the process combines elements from the traditional, Agile, prototyping, and continuous methodologies to address the unique characteristics and development principles of PaaS we selected as seen in Table 7.

Table 7: MaaS Core Principles

Principle	**Definition**
Structured initial planning	Sequential approach to gather and define foundational requirements, scope, maintenance support, and other project-related expectations
Iterative and incremental development	Leverage methodologies based on Agile philosophy for rapid development, adaptability, and user involvement, depending on project scope and requirements
Prototyping for broad requirements	Use a prototyping methodology, such as RAD, to deal with projects with ambiguous requirements, or with a focus on user experience
Continuous integration and delivery	Shorten feedback cycles, increase developer-customer interaction, improve product integrity and scalability
Streamlined project management	Utilise environment split approach of Rugby to manage collaboration, development, integration, delivery, and feedback efficiency

MaaS leverages the structured and sequential approach of traditional models to proceed between planning, development, and maintenance, where each phase is to be completed before the next. During the planning phase of the project, requirements, scope, and system design are defined, ensuring a solid foundation, correct setting of expectations, and rigorous documentation. The development phase is supported by either SCRUM or RAD Agile process models, depending on the completeness of requirements and scope defined in the planning phase. If the requirements are vague and not fully developed, or if the project contains subjective outcomes, such as a heavy focus on UI/UX, the RAD Agile process model should be chosen as the main development paradigm. If the requirements, scope, and project outcomes are clearly defined, the SCRUM process model should be chosen as the main development paradigm. Both SCRUM and RAD models strongly adhere to PaaS software development principles outlined in Table 5, as well as make effective use of PaaS services and capabilities. Following Rugby, CI/CD is integrated in parallel with either of the Agile process models (SCRUM or RAD), ensuring speed and reliability of releases, automation, and scalability. This results in a significant increase in frequency and quality of interactions between the developer and the customer, leading to a higher accuracy in delivering desired requirements, as well as significantly improved collaborative development and development speed [18]. Finally, project management utilises the five environments defined in the Rugby process model: development, integration, collaborative, delivery, and target environments. MaaS offers flexibility to handle varying levels of requirement clarity, faster time-to-market via iterative development and the integration of CI/CD, improved project stakeholder satisfaction and collaboration through short feedback loops, and scalable project management with a defined environment-based process model. To summarise, the MaaS core principles are defined in Table 7, and the model in Figure 9.

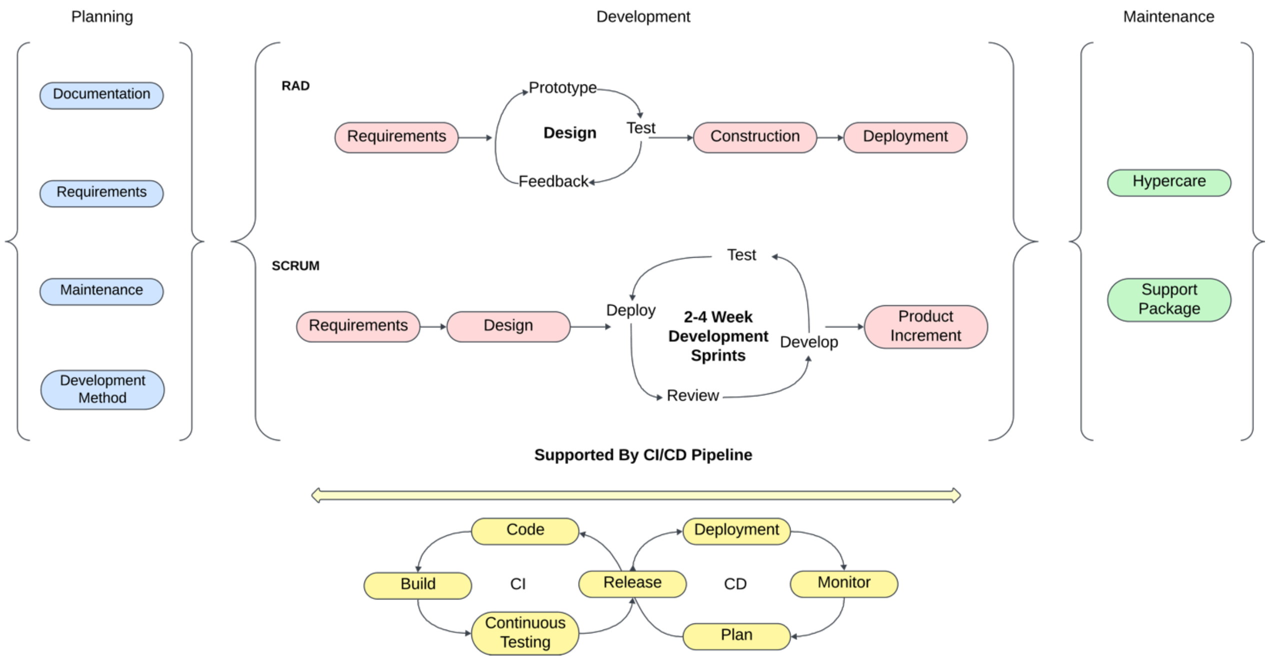


Figure 9: Methodology as a Service (MaaS)

5.2 Planning

The planning phase lays the foundation for the entire project by establishing a relationship with the customer, as well as scope, requirements, and overall application architecture. Two steps are defined in the planning phase: initiation and execution.

5.2.1 Initiation

Initiation is about understanding the needs of the customer. To achieve this, stakeholder workshops are conducted to identify the goal of the project, and its expected deliverables of the project: functional features, use-cases, integrations, and high-level system design choices. User roles, their responsibilities, and their interactions with the system are also defined. The completeness of these deliverables directly influences the results of the phase, as well as the choice of development process model in the development phase. If the completeness of requirements is low, customer-intended or otherwise, RAD is recommended as the development process model relying on rapid prototyping and customer collaboration. Likewise, if the completeness of requirements is high, customer-intended or otherwise, SCRUM is recommended as the development process model. Documentation-related deliverables, such as technical documentation, process guides, user manuals, testing depth, and training sessions are defined. Additionally, the governance of the project is defined via setting of roles and responsibilities of the customer, the end-user, and the development team. Lastly, PaaS-specific details are discussed by asking the following questions: does the customer already utilise the underlying PaaS for any other processes or use-cases? Are the and built-in services offered by PaaS platforms enough to accommodate the desired use-case, or configuration and customization is required? Which platform security features can be and should be utilised? This determines the level of extensibility (customization or configuration) of the resulting application.

5.2.2 Execution

Information resulting from the initiation step is compiled into a proposal or statement of work (SoW), which is a document outlining all deliverables that needs to be signed by the customer to trigger the official start of the project. The SoW includes the project description, goal, scope, governance, project methodology, assumptions, team structure, and a high-level timeline of the project. Financial analysis and cost estimates are also provided in the SoW. Once the document is signed, the project is officially underway.

5.3 Development

The development phase of MaaS entails iterative development cycles and continuous feedback loops, leveraging the rapid prototyping, building, and deployment capabilities offered by PaaS platforms [5, 26]. To maximise efficiency and adaptability, MaaS provides development teams the ability to choose between RAD and SCRUM methodologies based on project requirements and scope. A simple decision framework is provided in Table 8.

Table 8: Development Methodology Decision Framework

Criteria	**RAD**	**SCRUM**
Requirement clarity	Ambiguous or evolving requirements, UX heavy	Evolving requirements
Time-to-market urgency	Rapid prototype provision to meet urgent market needs, has initial product value	Regular delivery via time-boxed sprints, product value grows over time
Stakeholder involvement	Continuous involvement throughout development iterations	Involvement in SCRUM defined events (artifacts), such as a sprint review. Participate in testing and feedback
Planning and documentation	Lean planning and documentation for rapid development, evolving business realities	Detailed planning involving structured events and artifacts
Project complexity and scale	Best suited for small or medium sized projects, where room for rapid or unexpected change is possible	Scales well for complex projects with interdependent tasks

5.3.1 RAD

MaaS proposes the use of a prototyping Agile process model, namely RAD, if the requirements gathered from the planning phase are vague, the project entails a heavy focus on user experience, or other RAD favoured criteria are met (Table 8). The benefits of using this process model over SCRUM consist of faster prototyping and feedback cycles, reduced upfront planning and documentation, user-centric design focus, and increased flexibility in handling ambiguous or changing requirements. RAD consists of four steps, namely requirement definition, user design, construction, and deployment, which are described in detail in Section 2.2.5.

Focusing on most critical user needs, broad requirements are defined with some Definition of Done (DoD). Additionally, basic deadlines are set, together with DoD, to avoid infinite development or scope creep. These requirements, covered earlier by the planning phase of MaaS (Section 5.2), are classified as epics. Epics are agile components, typically representing large bodies of work (broad requirements), which can be further broken down into smaller units of work, namely stories. The user design step marks the beginning of iterative development. In this first iteration, developers and designers rapidly create the skeleton of the application based on requirements consisting of basic UI elements, functionality, and interactivity. Users provide immediate feedback, which is associated with epics defined earlier. This feedback is broken down into stories (small units of work), associated with their correlating epic, and added to a story backlog. This step is iterated continuously until a user-accepted design is reached. In each iteration, a new version of the product is built, reviewed by users, and improved continuously. In the next step, namely construction, developers work to complete the accepted design utilising the stories generated via user feedback, improving on their parent epic. Once the users are satisfied with the functionality of the product, the cutover step (deployment) begins. The application is finalised and deployed to a pilot group. Additionally, the development team remains available to handle minor adjustments or bug fixes based on the feedback of the pilot group.

RAD does not define concrete roles, therefore we introduce them as the development team, and the product owner (PO). The PO is the double-facing bridge between the development team and the customer. A PO represents the business needs of the customer, maintains the backlog, and manages prioritization of work. The development team is responsible for building, delivering, and maintaining the product.

5.3.2 SCRUM

MaaS proposes the use of an Agile development methodology, namely SCRUM, if the requirements gathered from the planning phase are well-defined, or the time-to-market urgency is low, or other SCRUM favoured criteria are met (Table 8). SCRUM process, roles, events, and artifacts are described in detail in Section 2.2.4.

Initial SCRUM backlog is derived from requirements gathered in the planning phase (Section 5.1). SCRUM proceeds in time-boxed iterations called sprints, which represent development iterations. During each sprint, new insights, changes, and modifications are also continuously added to the backlog.

At the beginning of a sprint, a sprint planning session is held to determine which backlog items are going to be delivered, further accompanied by daily stand-up meetings for progress alignment and impediment resolution. Once the sprint is over, a sprint review session is held to evaluate and demo the progress to stakeholders. The stakeholders (users) provide feedback which is then added to the backlog. Lastly, a sprint retrospective is held for the development team to reflect on their progress and operability in the last sprint.

For SCRUM, we define the following roles: PO who maintains the sprint backlog and, similarly to RAD, is a double-facing bridge between the development team and the customer; development team, who are responsible for building, delivering, and maintaining the product; SCRUM master who facilitates SCRUM related processes and artifacts, assists developers, and removes impediments.

5.4 Maintenance

After the development phase of the project is completed, a period of maintenance is critical to ensure the capture of unexpected bugs or defects. We propose to utilise the following two options in MaaS: hypercare and maintenance packages. Hypercare is a predefined short-term support period where the development team works on bugs or defects identified during or post product delivery. Additionally, we propose maintenance packages, where customers can opt in for a supporting team which will work to provide maintenance for a predefined set of hours every month. This can also entail work on improvements or feature requests. Both the hypercare and maintenance packages are discussed and agreed upon during the project planning phase.

5.5 CI/CD

MaaS incorporates CI/CD practices to enhance the integrity and efficiency of development, as well as to reduce user-developer feedback loops. It ensures that every development iteration results in a validated, tested, and deployable product via automating application build, publishing, installing, and testing processes.

As previously stated, MaaS utilises the five-environment solution of Rugby (Section 5.5) to streamline project management. Development teams work on stories in the development environment. At the end of either RAD or SCRUM development iteration, the completed units of work (code and configuration) are packaged into a release artifact via the integration environment. This release artifact is then deployed to a delivery environment for final testing, where, if successfully tested, is further deployed to a target (production) environment. CI/CD automates the tasks of building, unit-testing, integration-testing, packaging, and deployment of releases.

5.6 Project Management

Project management in MaaS is centred around the need for an efficient and CI/CD adaptable development process for PaaS platforms. Based on the findings reported in Chapters 2 and 4, a hybrid methodology consisting of elements from Agile, RAD, SCRUM, CI/CD, and Rugby to ensure effective development and project execution.

MaaS adopts the five-environment ecosystem approach of Rugby (Section 2.2.6), split into development, integration, collaboration, delivery, and target environments. Rugby emphasises CI/CD and rapid feedback loops [18], making it an ideal fit for MaaS. During each development iteration developers work within the development environment, after which their work is merged via the integration environment and deployed to a delivery environment for feedback. The customer then provides feedback via both the delivery and collaboration environments. If the delivered work is acceptable and tested it can be moved to the target environment. This environment structure can be mapped onto PaaS platforms based on their services and features.

We define a project manager (PM) role to oversee the entire five-environment process in collaboration with other actors of either the RAD or SCRUM development processes.

6 Validation

This chapter introduces the validation process for the application case study of the software development methodology MaaS defined in Chapter 5. The goal of this validation process is to assess whether MaaS can provide measurable benefits over alternative established development methodologies with regards to PaaS software development. These benefits are otherwise defined as improved development outcomes, such as faster development iterations, improved stakeholder collaboration, and fewer defects. Accomplishing this goal answers **RQ 3**. Additionally, it serves as the demonstration step within the Design Science paradigm, in preparation for the evaluation (Likert scale survey questions) and communication steps (survey result analysis, future work, and conclusion) [7, 8].

To demonstrate the feasibility of MaaS, we apply it to a mock project on ServiceNow, one of the leading metadata-driven PaaS platforms. Additionally, we provide an architectural example of how the methodology can be applied to a similar metadata-driven PaaS platform – Mendix.

6.1 Use Case

Due to the complexity of real-life projects and their potential obscurement of the methodology, we define a mock project, allowing us to observe how MaaS works when applied on ServiceNow. The customer is a security operations centre (SOC) for a large enterprise. This enterprise already has an established regular incident management process in ServiceNow; however, the number of incoming incidents has grown two-fold. It is becoming increasingly difficult to efficiently manage all incidents with varying degrees of impact and priority. For this reason, SOC requires a security incident management tool to allow a subset of users to create and track the progress of security incidents, thus detaching them from regular incidents. SOC decided to build a new custom application in ServiceNow to manage security incidents separately, as they use the platform for other services, including the basic incident management. Table 9 shows two possible example sets of requirements which would trigger either a RAD type or a SCRUM type development model implementation method.

Table 9: SOC General Project Requirements

	RAD	**SCRUM**
Project vision	Build an intuitive and visually appealing portal for the creation, tracking, and management of security incidents. Requires a fast time-to-market	Develop a comprehensive security incident management application with detailed workflows, access management, and advanced reporting
Visual interface	Focus on a minimalistic, yet highly responsive modern design	Minimalistic design, UI is structured to support role-based views, as well as self-servicing functionality, such as tooltips
Design	A simple and dynamic security incident form with basic fields (e.g. “Title”, “Description”). Use of visual cues such as icons to support clarity	Detailed security incident form featuring predefined fields (e.g. type, severity, affected systems, date and time) that can support a full security incident lifecycle, including escalations, approvals, service-level agreements, and auditing

Integration	Acts as standalone application with minimal integration with the existing incident management tool. Must have the ability to create a security incident from a normal incident and vice-versa. CI/CD used for rapid deployment of incremental improvements	Standalone application with seamless integration with existing incident management tool, as well as other third-party security tools. CI/CD pipeline is established to support regular builds, testing, and deployment
Reporting and analytics	Ability to track security incident, state-change notifications, simple dashboard to view the current states of security incidents (e.g. open, in progress, complete)	Detailed dashboards with real-time analytics, multi-level notification system, custom reporting, and KPI tracking

6.2 Applying MaaS to ServiceNow

MaaS was developed as a hybrid software development methodology specifically tailored for PaaS platforms, incorporating elements from traditional and Agile methodologies, as well as CI/CD practices, to create a structured yet flexible approach which aligns with PaaS development principles defined in Table 5. This section describes how MaaS can be utilised in a project setting, through the demonstration of a run of a development iteration for our mock use case (Section 6.1), following the phases laid out in Chapter 5.

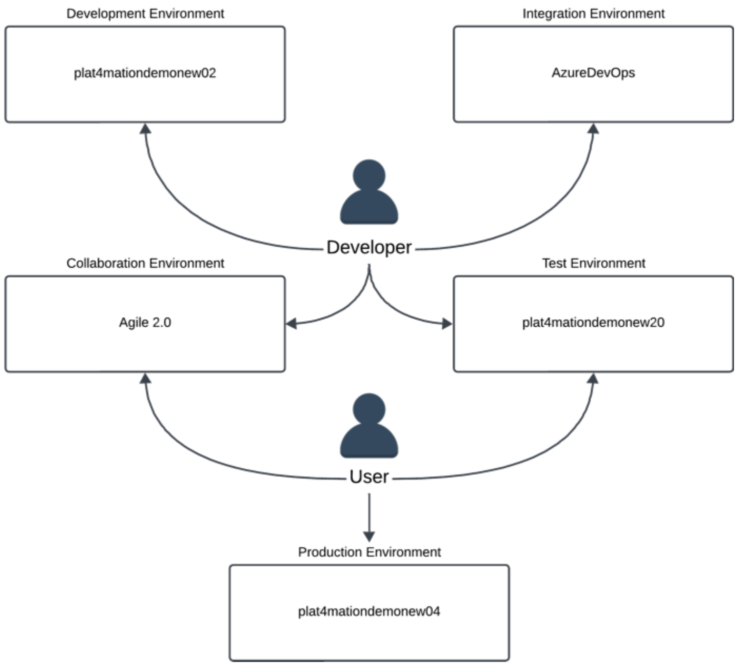


Figure 10: Adaptation of Rugby's Five-Environment Model for ServiceNow

At its core, ServiceNow is a cloud platform that streamlines and automates enterprise service management, providing a configurable data model, powerful workflow engine, and an array of built-in applications to manage IT, HR, security, among others. The primary reason for selecting ServiceNow is the availability of access to the product, but also due to it being recognised for its strong workflow automation, pre-built components and shared services, low-code and no-code capabilities, scalability, security, extensibility, and integrated CI/CD support. Lastly, these characteristics are shared among other metadata-driven PaaS platforms (Section 2.1.1), making ServiceNow a suitable testing ground for MaaS.

6.2.1 Project Management

For MaaS project management we follow the MaaS principles defined in Table 7 and adapt the five-environment model of Rugby (Figure 10). We use ServiceNow to facilitate the following structure: three ServiceNow instances for our development, test (clone of the production environment), and production environments; Azure DevOps is used as the integration environment, providing both CI/CD pipeline facilitation, and a Git code repository. Finally, ServiceNow's Agile 2.0 module is utilised as the collaboration environment, allowing for communication, backlog management, and progress tracking, between developers, end-users, and other involved actors. Similarly to the environment model, we draw inspiration form Rugby to define our process model (Figure 11). For this demonstration, we keep it simple. The following sections go in-depth into the steps of the process model.

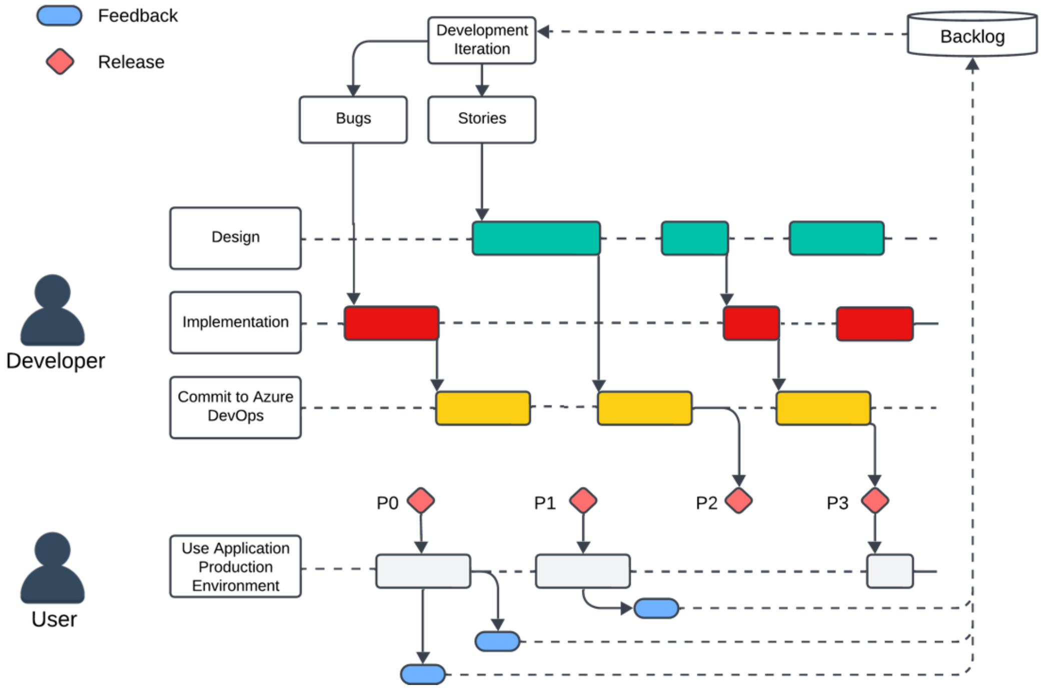


Figure 11: MaaS Process Model (adapted from [18])

6.2.2 CI/CD

For our CI/CD pipeline, Azure DevOps serves as the facilitator, hosting a Git repository for code management, and enabling automation via a pipeline. The pipeline uses a YAML definition (code snippet 1) to provide automation of the build, test, and deployment processes. We set our trigger to be a commit on the "master" branch. Changes from developers are committed to the Azure repository, triggering the build stage. For this case study, we include the step of building and publishing the application on the development instance (environment), as committing the changes does not inherently publish the application and is a separate action in ServiceNow. Typically, a custom application requires to be published to be available for manual retrieval or deployment from a different instance, i.e., test instance. For additional information, application artifacts, such as the application version number, are added in the relevant connection (to Azure DevOps) record of that instance. If the build stage ran successfully, test stage is triggered, which automatically deploys the application on our test environment. Once deployed, automated tests are executed via ServiceNow Automated Test Framework (ATF). We implement a rollback feature if testing is unsuccessful, safeguarding the integrity of the production environment. Lastly, if the test stage is completed successfully, we automatically deploy the application to the production instance.

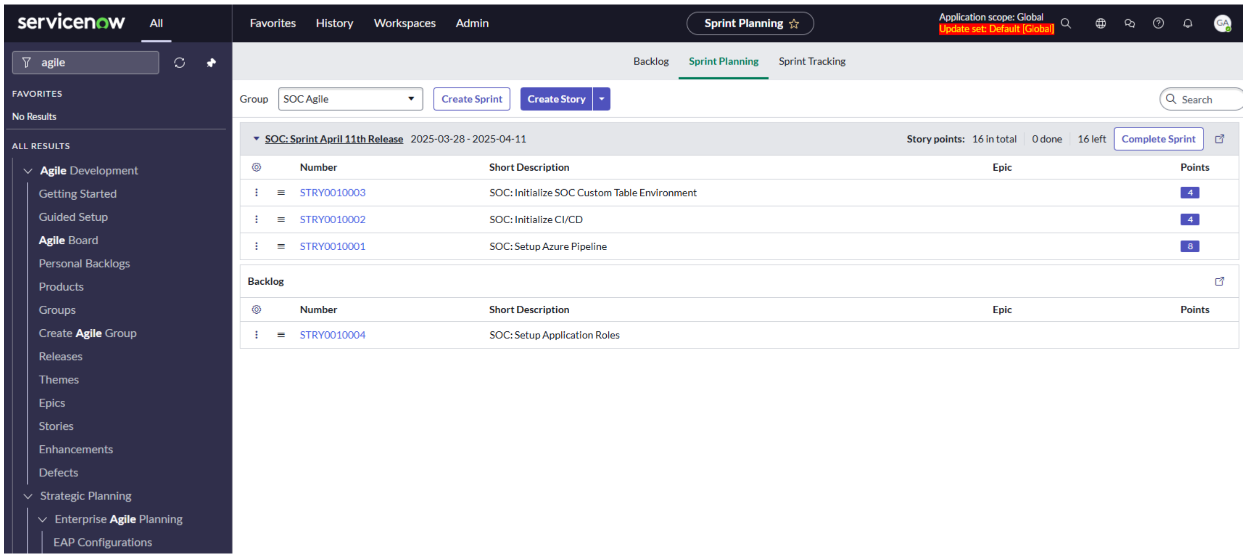
```yaml
trigger:
  branches:
    include:
    - master
variables:
- name: APPSYSID
  value: 4951ec8fc39866102d4bf50f05013175
- name: TESTSUITEID
  value: cd4d28cbc3d866102d4bf50f05013144
- name: BRANCH
  value: $(Build.SourceBranchName)
- name: JUNIT_FILE_DEV
  value: '$(System.DefaultWorkingDirectory)/DevTest.xml'
- name: JUNIT_FILE_TEST
  value: '$(System.DefaultWorkingDirectory)/ACCTest.xml'
- name: BRANCH
stages:
- stage: Build
  condition: eq(variables['Build.SourceBranch'], 'refs/heads/master')
  jobs:
  - job: ApplyChange_Publish
    steps:
    - task: ServiceNow-CICD-SC-Apply@2
      inputs:
        connectedServiceName: 'SOC Dev Instance'
        appSysId: '$(APPSYSID)'
        branchName: '$(BRANCH)'
    - task: ServiceNow-CICD-App-Publish@2
      condition: succeeded()
      inputs:
        connectedServiceName: 'SOC Dev Instance'
        sysId: '$(APPSYSID)'
        versionFormat: 'detect'
    - task: ServiceNow-DevOps-Agent-Artifact-Registration@1
      inputs:
        connectedServiceName: 'plat4mationdemonew02-ServiceNow CICD-ServiceNow DevOps Service Connection'
        artifactsPayload: "{\n  \"artifacts\": [\n      {\n        \"name\": \"ServiceNow Azure CICD\",\n        \"version\": \"1.$(Build.BuildId)\",\n        \"semanticVersion\": \"1.$(Build.BuildId).0\",\n        \"repositoryName\": \"ServiceNow Azure CICD\"\n      }\n  ]\n}\n"
    - task: ServiceNow-DevOps-Agent-Package-Registration@1
      inputs:
        connectedServiceName: 'plat4mationdemonew02-ServiceNow CICD-ServiceNow DevOps Service Connection'
        packageName: 'ServiceNow Azure CICD'
        artifactsPayload: |
          {
              "artifacts": [
              {
                  "name": "ServiceNow Azure CICD",
                  "repositoryName": "ServiceNow Azure CICD",
                  "version": "1.$(build.buildId)",
                  "pipelineName":"$(system.teamProject)/$(build.definitionName)",
                  "taskExecutionNumber":"$(build.buildId)",
                  "stageName":"$(system.jobDisplayName)",
                  "branchName":"$(build.sourceBranchName)"
              }],
              "pipelineName":"$(system.teamProject)/$(build.definitionName)",
              "taskExecutionNumber":"$(build.buildId)",
              "stageName":"$(system.jobDisplayName)",
              "branchName":"$(build.sourceBranchName)"
          }
- stage: Test
  condition: and(succeeded(), eq(variables['Build.SourceBranch'], 'refs/heads/master'))
  jobs:
  - job: InstallToTest_RunTest
    steps:
    - task: ServiceNow-CICD-App-Install@2
      inputs:
        connectedServiceName: 'SOC Test Instance'
        sysId: '$(APPSYSID)'
    - task: PowerShell@2
      name: RunTestATF
      inputs:
        targetType: inline
        script: "$user = \"test.azure\"\n$pass = \"012s+@&TgV.1@L5a^\"\n$base64AuthInfo = [Convert]::ToBase64String([Text.Encoding]::ASCII.GetBytes((\"{0}:{1}\" -f $user, $pass)))\n$headers = New-Object \"System.Collections.Generic.Dictionary[[String],[String]]\"\n$headers.Add('Authorization',('Basic {0}' -f $base64AuthInfo))\n$headers.Add('Accept','application/json')\n$headers.Add('Content-Type','application/json')\n\n$uri = \"https://plat4mationdemonew20.service-now.com/api/sn_cicd/testsuite/run?test_suite_sys_id=cd4d28cbc3d866102d4bf50f05013144\"\n$method = \"post\"\n$atfProgress = Invoke-RestMethod -Headers $headers -Method $method -Uri $uri\n$progressID =  $atfProgress.result.links.progress.id | ConvertTo-Json | ConvertFrom-Json\necho $progressID\nStart-Sleep -s 30\n\n$uri = \"https://plat4mationdemonew20.service-now.com/api/sn_cicd/progress/$progressID\"\n$method = \"get\"\n$atfResult = Invoke-RestMethod -Headers $headers -Method $method -Uri $uri \n$resultID = $atfResult.result.links.results.id | ConvertTo-Json | ConvertFrom-Json\necho $progressID\n\n\n$uri = \"https://plat4mationdemonew20.service-now.com/api/pl4/cicd_junit_file?result_id=$resultID\"\n$method = \"get\"   \n$response = Invoke-RestMethod -Headers $headers -Method $method -Uri $uri\necho $response.result.file | ConvertTo-Json | ConvertFrom-Json > '$(JUNIT_FILE_TEST)'\n"
    - task: PublishTestResults@2
      inputs:
        testResultsFormat: 'JUnit'
        testResultsFiles: '**/ACCTest*.xml'
      condition: succeededOrFailed()
    - task: CmdLine@2
      inputs:
        script: 'cat $(JUNIT_FILE_TEST)'
  - job: Rollback
    dependsOn:
    - InstallToTest_RunTest
    condition: failed()
    steps:
    - task: ServiceNow-CICD-App-Rollback@2
      inputs:
        connectedServiceName: 'SOC Test Instance'
        sysId: '$(APPSYSID)'
        autodetectVersion: 'yes'
- stage: DeployToProd
  condition: and(succeeded(), eq(variables['Build.SourceBranch'], 'refs/heads/master'))
  jobs:
  - job: InstallToProd
    steps:
    - task: ServiceNow-CICD-App-Install@2
```

```
          inputs:
            connectedServiceName: 'SOC Prod Instance'
            sysId: '$(APPSYSID)'

```

Code Snippet 1: Azure DevOps Pipeline YAML

6.2.3 Planning



SOC: Sprint April 11th Release 2025-03-28 - 2025-04-11 Story points: 16 in total | 0 done | 16 left

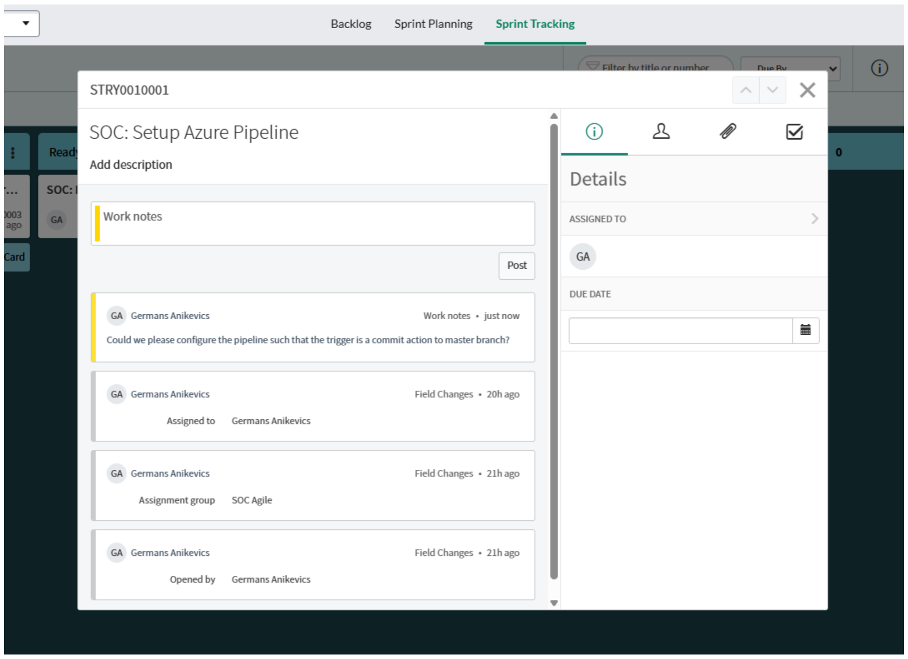
Number	Short Description	Epic	Points
STRY0010003	SOC: Initialize SOC Custom Table Environment		4
STRY0010002	SOC: Initialize CI/CD		4
STRY0010001	SOC: Setup Azure Pipeline		8

Backlog

Number	Short Description	Epic	Points
STRY0010004	SOC: Setup Application Roles		

Figure 12: SOC Sprint Planning

During the planning phase workshops are conducted to identify scope, project requirements, selection of the agile development process model (SCRUM or RAD), and define documentation structure, testing approach, deadlines, project roles, and the maintenance support package. MaaS proposes the use of two alternative agile process models for the development phase, SCRUM and RAD. However, the choice should not be limited solely to these two models, if the requirements of the project demand a different and more suitable iterative process. We provide an example of requirements (Table 9), which can trigger either one of the models, via our project use case (Section 6.1). For demonstration purposes, the choice of model does not matter, as both are iterative development processes.



STRY0010001

SOC: Setup Azure Pipeline

Add description

Work notes

Post

GA Germans Anikevics — Work notes • just now
Could we please configure the pipeline such that the trigger is a commit action to master branch?

GA Germans Anikevics — Field Changes • 20h ago
Assigned to: Germans Anikevics

GA Germans Anikevics — Field Changes • 21h ago
Assignment group: SOC Agile

GA Germans Anikevics — Field Changes • 21h ago
Opened by: Germans Anikevics

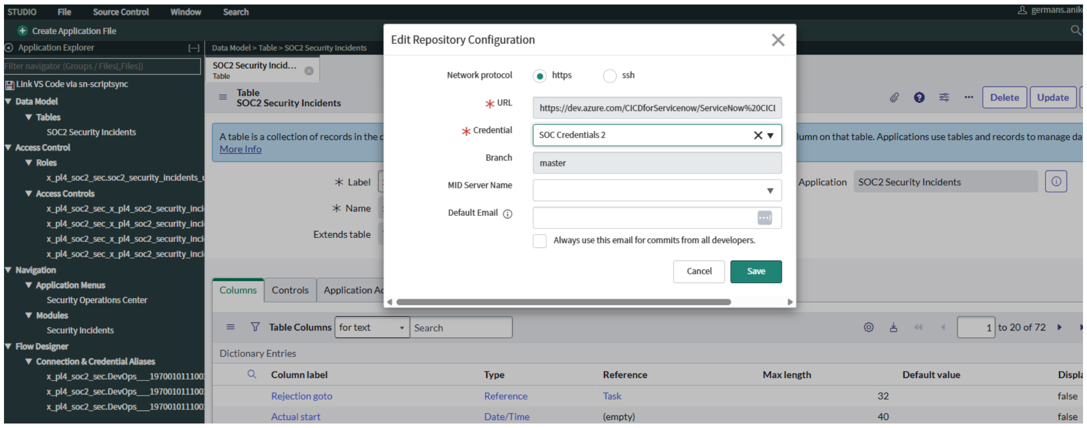
ASSIGNED TO: GA

DUE DATE:

Figure 13: SOC Collaboration Communication Example

6.2.4 Development

Requirements captured during the planning phase are converted into actionable units of work called user stories, utilising Agile 2.0 of ServiceNow, which is setup on the production instance (Figures 12, 13). These stories initially populate a general backlog, which are chosen into iterative sprints or rapid prototyping cycles, depending on their priority. Developers complete the story tasks in the development environment utilising the built-in features, updating their progress directly in ServiceNow. This allows the customers to track the progress of work, communicate, and provide feedback or adaptation suggestions if needed. Once the stories are completed within an iteration, developers commit their changes to Azure DevOps (Figure 14), which triggers deployment pipeline initially to test, and subsequently to production environments.


STUDIO
File
Source Control
Window
Search
Create Application File
Application Explorer
Link VS Code via sn-scriptsync
Data Model
Tables
SOC2 Security Incidents
Access Control
Roles
x_pl4_soc2_sec.soc2_security_incidents_
Access Controls
x_pl4_soc2_sec_x_pl4_soc2_security_inci
x_pl4_soc2_sec_x_pl4_soc2_security_inci
x_pl4_soc2_sec_x_pl4_soc2_security_inci
x_pl4_soc2_sec_x_pl4_soc2_security_inci
Navigation
Application Menus
Security Operations Center
Modules
Security Incidents
Flow Designer
Connection & Credential Aliases
x_pl4_soc2_sec.DevOps___19700101110
x_pl4_soc2_sec.DevOps___19700101110
x_pl4_soc2_sec.DevOps___19700101110
Delete
Update
Columns
Controls
Application A

Edit Repository Configuration

Network protocol: [✓ https] ssh

URL: https://dev.azure.com/CICDforServicenow/ServiceNow%20CICI

Credential: SOC Credentials 2

Branch: master

MID Server Name:

Default Email:

Always use this email for commits from all developers.

Cancel | Save

Data Model > Table > SOC2 Security Incidents

Table SOC2 Security Incidents

A table is a collection of records in the d… lumn on that table. Applications use tables and records to manage da…

Application: SOC2 Security Incidents

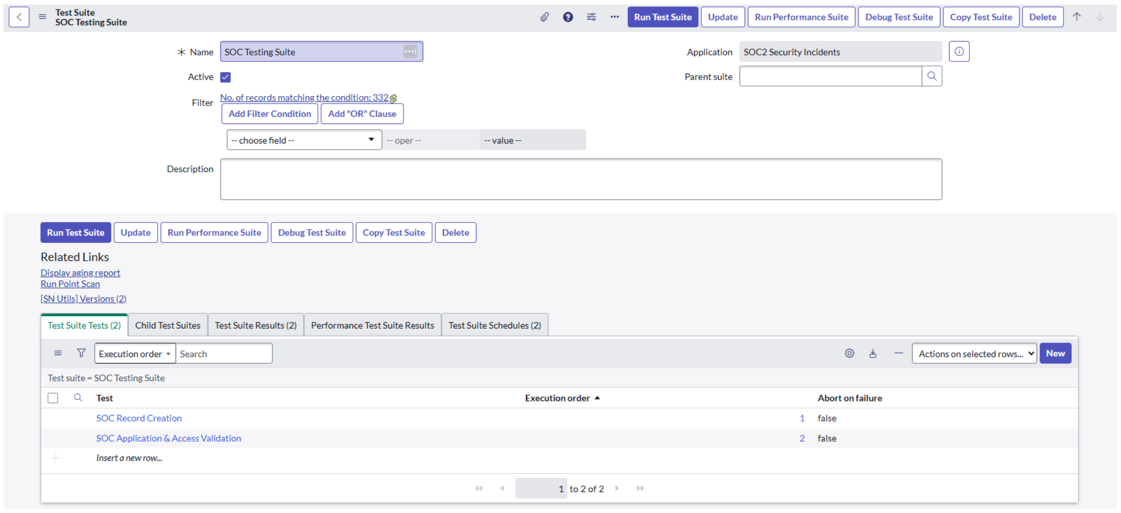
Table Columns for text — 1 to 20 of 72

Dictionary Entries

Column label	Type	Reference	Max length	Default value	Displ
Rejection goto	Reference	Task	32		false
Actual start	Date/Time	(empty)	40		false

Figure 14: SOC Commit to Azure DevOps

How the user stories are tested depends on the agreement made during the planning phase. For this iteration demonstration, testing is first manually conducted by developers on the development instance, after a user story is complete. At the end of an iteration, when the work is committed to Azure DevOps (Section 6.2.2), our pipeline is triggered, which runs the ServiceNow ATF test suite, upon deploying the application to test environment. If the test suite tests are completed successfully, the application is deployed to the production environment.



Name	SOC Testing Suite
Active	✓
Filter	No. of records matching the condition: 332
Description	
Application	SOC2 Security Incidents
Parent suite	

Test suite = SOC Testing Suite

Test	Execution order	Abort on failure
SOC Record Creation	1	false
SOC Application & Access Validation	2	false

Figure 15: Test Environment Test Suite

6.2.5 Maintenance

The maintenance phase is dependent on the type of agreement established during the planning phase. MaaS proposes the use of two alternatives: hypercare and managed services. For this demonstration, we use the latter. A managed services team provides ongoing support through a predefined support package. This can be done in a form of a contractual agreement which dictates the number of hours the team is expected to spend every week on the maintenance of the application. Maintenance activities include bug fixes and enhancements which may arise after the completion of the development phase. Considering the CI/CD pipeline is already setup, managed services team can continue to utilise it to deliver their improvements iteratively and efficiently. The feedback loop remains consistently short, ensuring continuous delivery quality, and responsiveness towards evolving user needs.

6.3 Applying MaaS to Mendix

Next to ServiceNow, we provide an example of a potential application of MaaS on another metadata-driven PaaS platform, namely Mendix, which is a platform that facilitates development primarily through a visual development environment called Mendix Studio Pro. Unlike ServiceNow, Mendix Studio Pro provides developers the ability to build and test their applications locally, rather than through cloud. It encourages rapid prototyping and experimentation due to development being local, avoiding additional costs, as well as affecting other environments. For the collaboration environment, Mendix can utilize Sprintr, which is a project management tool built into Mendix. Packages from local Mendix environments can be uploaded directly into Sprintr, and subsequently to integration, acceptance (test) or production environments. For testing, Mendix does not offer automated testing services, however, third party vendors exist which do offer such capabilities. Lastly, collaboration and communication are offered via Sprintr built-in user story, backlog, and sprint planning capabilities, however, third-party systems such as JIRA or Slack are usually preferred. The planning, development, and maintenance phases follow the same process model (Figure 11) and steps defined for ServiceNow in Chapters 5 and 6, while utilising the environment setup of Rugby onto Mendix as displayed in Figure 16.

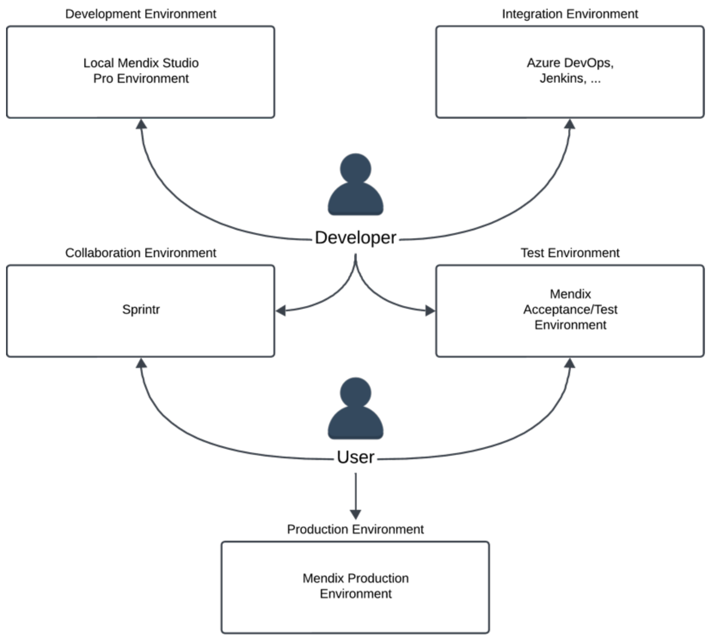

Development Environment
Local Mendix Studio Pro Environment
Integration Environment
Azure DevOps, Jenkins, ...
Developer
Collaboration Environment
Sprintr
Test Environment
Mendix Acceptance/Test Environment
User
Production Environment
Mendix Production Environment

Figure 16: Adaptation of Rugby's Five-Environment Model for Mendix

7 Results

In this chapter we present the results obtained through answering the research questions defined in Section 1.3. The thesis explored the foundational differences between the processes of traditional software development and software development conducted using PaaS environments to answer **RQ 1**. We examined the alignment and applicability of established and novel software development methods, as well as their components, with software development principles specific to PaaS to answer **RQ 2**. Lastly, based on these insights, we answer **RQ 3**, namely whether the benefits of a PaaS-oriented software development methodology led to better outcomes compared to non-PaaS oriented methodologies.

7.1 Survey

Validation of MaaS has been performed through the evaluation of benefits associated with improved development outcomes, such as faster development iterations, improved stakeholder collaboration, and fewer defects. From these benefits, we derive the following evaluation metrics: collaboration, performance, resource efficiency, end-user involvement, and delivery effectiveness. Considering the use of experts, the questions used for evaluation aim to quantify subjective human perceptions, such as opinions, attitudes, and experiences, with regards to the demonstration of the MaaS application on a mock project. To transform perceptions into measurable data, we use the Likert scale, which was devised specifically for measuring perception in an accepted and validated manner [28]. In other words, we aimed to capture qualitative insights (participants subjective experiences) quantitatively.

$$\alpha = \frac{k}{k-1}\left(1 - \frac{\sum_{i=1}^{k}\sigma_i^2}{\sigma_T^2}\right)$$

Equation 1: Cronbach's Alpha

$$W = \sum_{i=1}^{N_r}[\operatorname{sgn}(x_{2,i} - x_{1,i}) \cdot R_i]$$

Equation 2: Wilcoxon Signed-Rank Test

The Likert scale values represent a particular sentiment, ranging from either strongly agreeing or strongly disagreeing with the posed question or statement, while also maintaining a neutral middle point. Likert scale point ranges typically vary between 1-3, 1-5, 1-7, and 1-10 points, sometimes beyond that, depending on the underlying goal of the research, and its human perception evaluation. For this survey, a Likert scale of 1-7 was used, as it reduces ambiguity via a more nuanced spectrum of choices. In lower point ranges, participants might be forced to choose between two equally undesirable points, whereas a range of 1-7 points provides more choices, increasing the probability of meeting the objective reality of people [28]. Higher point ranges can have an opposite effect, introducing confusion, as it becomes increasingly more difficult to reliably distinguish between a larger number of options.

$$U_1 = n_1 n_2 + \frac{n_1(n_1+1)}{2} - R_1, U_2 = n_1 n_2 + \frac{n_2(n_2+1)}{2} - R_2$$

Equation 3: Mann-Whitney U Test

There are two schools of thought with regards to the treatment of Likert scales for analysis: ordinal and interval scales. Ordinal scales consider choices as arranged in a particular ranking order, without considering the relative distance between two responses quantitatively, whereas interval scales aim to combine a set of responses to produce a general composite score for a particular question [28]. Our goal is to evaluate five distinct metrics, meaning that for each metric a subset of questions is defined to deduce a more general composite score as the outcome. In the case of this thesis, we have a small sample size of ten experts. Because of this, normality assumption cannot be guaranteed even if it were true. Therefore, we use non-parametric tests and consider our scale as ordinal; namely Wilcoxon signed-rank test (Equation 2) to determine if the experts view any given evaluation metric of MaaS as above average, Mann-Whitney U test (Equation 3) to compare the composite scores *technical* or *functional* subgroups to uncover any potential role perception differences, and the Spearman correlation test (Equation 4), to explore if experts who rank one metric positively also rank the other metrics similarly. For an additional reliability check on our Likert scale survey questions, we use Cronbach's alpha test (Equation 1) to make sure the questions for each evaluation metric measure the same thing. Finally, as we treat our scale as ordinal, medians and interquartile ranges (IQR) are used for descriptive statistical analysis after Cronbach's alpha test.

$$\rho = 1 - \frac{6 \sum d_i^2}{n(n^2 - 1)}$$

Equation 4: Spearman's Rank Correlation

7.2 PaaS and Traditional Software Development Differences

The key differences between PaaS and traditional software development were identified through a literature review, explored and analysed in Section 2.1 and Chapter 3, identifying PaaS characteristics (Table 4) and PaaS development principles (Table 5). To answer **RQ 1,** we compile the key findings in Table 10 below.

Table 10: Development Differences Between Traditional and PaaS Environments

Infrastructure abstraction	Traditional software development environments require the manual management of their infrastructure, namely hardware, software, middleware, and databases. In contrast, PaaS platforms facilitate and therefore abstract this infrastructure management, allowing developers to focus primarily on delivery of applications, user needs and functionalities, and other project tasks
Built-in tools and services	PaaS platforms inherently offer pre-built services through shared components, such as access and security controls, data management, functioning software templates and building blocks, security frameworks, development tools, integration connectors, and more. Traditional environments in contrast require manual configuration and management or custom solutions to facilitate the same functions
Scalability and flexibility	Traditional environments typically scale through manual hardware upgrades or additional server procurement, whereas PaaS platforms support automatic on demand scaling, reducing downtime, maintenance, effort, and allowing resource usage to match the demand or load on the platform

Development speed	Rapid prototyping capabilities of PaaS platforms through extensive component reusability, templates, and other built-in tools and services, as well as the general infrastructure abstraction significantly shorten developer loops, allowing for a fast time-to-market delivery of software applications
Knowledge	Traditional environments require developers to have a deep understanding for tools, programming languages, and the surrounding infrastructure, which can differ from project to project, or organisation to organisation. PaaS platforms significantly lower the entry barrier for developers due to their extensive configurational and other built-in services capabilities, documentation, training and learning materials, enabling both experienced software developers and citizen developers to produce quality software.

7.3 PaaS Methodology Compatibility

We have established that PaaS platforms streamline software development through the provision of model-driven development, cloud services and built-in tools, collaborative environments, integration capabilities, and infrastructure abstraction to allow developers to produce and prototype applications with a faster time-to-market than with traditional environments. We analysed established and novel methodologies against PaaS development principles (Tables 4, 5, and 10) to answer **RQ 2** in Chapter 4.

The findings identified Agile process models and CI/CD support as highly compatible with PaaS software development principles. Agile methodologies emphasize iterative and incremental delivery, as well as user collaboration, which align with the inherent rapid prototyping, short development loops, collaboration environments, and fast-time-to market capabilities of PaaS platforms. CI/CD capabilities align with PaaS via automating testing, integration, and deployment processes, further shortening development loops, enhancing development speed, and improving software sanitation. Additionally, many PaaS platforms, e.g. ServiceNow and Mendix, provide built-in support of CI/CD pipelines. Hybrid methodologies which blend Agile and traditional elements provide flexibility to leverage the benefits of structured initial planning with iterative development and prototyping, utilising the full range of PaaS automation and scalability capabilities. Lastly, the Rugby Agile process model emphasizes CI/CD and its frequent releases with immediate user feedback loops, aligning with PaaS capabilities and development principles. Its five-environment model, consisting of development, integration, collaboration, delivery, and target environments, complements the multi-instance setup typically used by PaaS platforms and can therefore be used for facilitating structured project management.

7.4 Benefits of a PaaS-oriented Software Development Methodology

Finally, based on the answers of **RQ 1** and **RQ 2**, we proposed an experimental hybrid methodology MaaS (Chapter 5) to conduct an application case study using ServiceNow and answer **RQ 3**, namely, to explore if benefits of a PaaS-oriented software development methodology led to better outcomes in collaboration, performance, resource efficiency, end-user involvement, and delivery. To achieve this, a validation process was performed (Chapter 6) in the form of a mock project on ServiceNow to assess whether MaaS can provide measurable benefits over established traditional, iterative, or hybrid methodologies. This process was then presented to ten industry experts with experience in both traditional and PaaS software development environments. The feedback was recorded in the form of

Likert-scale survey with blocks of questions (Appendix A) pertaining to each improved outcome metric (Section 7.1).

Table 11: Initial Median Scores

Collaboration	Performance	Resource Efficiency	End-User Involvement	Delivery
5.0	3.5	5.0	6.0	5.5
4.5	4.5	5.0	5.5	5.5
4.5	5.0	4.5	6.0	4.0
4.5	5.0	4.5	5.5	4.0
6.5	6.5	6.0	6.0	6.5
5.0	5.0	4.5	5.0	4.0
4.0	5.0	5.0	4.0	4.5
4.5	5.0	5.0	5.0	5.0
5.5	4.0	5.0	4.5	6.0
5.0	5.0	5.0	5.5	5.0

Table 12: Initial Cronbach's Alphas

	Collaboration	Performance	Resource Efficiency	End-User Involvement	Delivery
α	0.723	0.656	0.420	0.862	0.833

First, we calculate initial composite scores as median scores (Table 11) across all five metrics for all respondents and their subsequent Cronbach's alphas (Table 12) to determine if the questions for each metric measure the same thing. We use a score of $\alpha < 0.70$ to determine if the questions in each block do not pertain to the same topic. If $\alpha < 0.70$ for a metric, then we remove certain questions from each metric question block to improve the α. After further analysis, to achieve $\alpha > 0.70$ for each metric, we removed questions (see Appendix A) "To what extent do you agree that MaaS accelerates and shortens development iteration cycles?" and "Compared to traditional software development methods, does MaaS improve performance outcomes with regards to PaaS-specific software development projects?" for the performance question block, and questions "To what extent do you agree that MaaS optimises the use of resources in PaaS-specific software development projects?" and "Compared to traditional software development methods, does MaaS demonstrate higher resource efficiency with regards to PaaS-specific software development?" for the resource efficiency question block. Once removed, we update our initial median scores (Table 13) based on the positive $\alpha > 0.70$ score (Table 14).

Table 13: Updated Median Scores

Collaboration	Performance	Resource Efficiency	End-User Involvement	Delivery
5.0	3.0	4.0	6.0	5.5
4.5	4.0	4.5	5.5	5.5
4.5	5.0	5.0	6.0	4.0
4.5	4.5	4.0	5.5	4.0
6.5	6.5	6.5	6.0	6.5
5.0	4.5	4.0	5.0	4.0

4.0	5.5	5.0	4.0	4.5
4.5	5.0	5.0	5.0	5.0
5.5	5.0	5.0	4.5	6.0
5.0	4.5	6.0	5.5	5.0

Table 14: Updated Cronbach's Alphas

	Collaboration	**Performance**	**Resource Efficiency**	**End-User Involvement**	**Delivery**
α	0.723	0.732	0.728	0.862	0.833

Table 15 presents medians, means, and dispersion indices for all completed responses. In general, for all five of the composite metric scores (medians), the neutral point of 4.0 is exceeded, indicating an overall favourable perception of MaaS by experts. End-user involvement shows the strongest agreement with median = 5.5 and mean = 5.30 ± 0.67 suggesting that MaaS engages end-users effectively, followed by resource efficiency and delivery – both with medians = 5.0. Collaboration and performance share the lowest, however, still positive agreement with medians = 4.75 each.

Table 15: MaaS Likert Survey Descriptive Statistics

	Count	**Median**	**Mean**	**Std**	**Min**	**Max**	**Sem**
Collaboration	10.0	4.75	4.90	0.699	4.0	6.5	0.221
Performance	10.0	4.75	4.75	0.920	3.0	6.5	0.291
Resource Efficiency	10.0	5.00	4.90	0.843	4.0	6.5	0.267
End-user Involvement	10.0	5.50	5.30	0.675	4.0	6.0	0.213
Delivery	10.0	5.00	5.00	0.882	4.0	6.5	0.279

To determine if the experts rated any given metric above average, or the neutral midpoint of four (1-7 Likert scale), Wilcoxon signed-rank test was performed on every composite metric with results recorded in Table 16. All five medians differ significantly from the neutral midpoint with $p < 0.05$ in every case, signifying that experts perceive MaaS as potentially beneficial across all evaluation metrics. The highest W values of W = 45.0 can be found for collaboration and end-user involvement, indicating that almost every expert ranked these metrics above the neutral midpoint of four. Even the lowest W values of W = 28.0 for resource efficiency and delivery also report a positive outlook with $p < 0.05$. Given the small sample, the tests should be viewed as exploratory, however, the consistent pattern of positive perception of MaaS on all its evaluation metrics further supports our descriptive results in Table 15, as well as the proposition that MaaS can deliver advantages in collaboration, performance, resource efficiency, end-user involvement, and delivery.

Table 16: Wilcoxon Signed-rank Test Results

	Collaboration	**Performance**	**Resource Efficiency**	**End-User Involvement**	**Delivery**
W	45.0	39.5	28.0	45.0	28.0
p	0.002	0.021	0.008	0.002	0.008

In Section 1.5.1 we presented the experts partaking in the survey and their respective roles. To examine whether *technical* experts (n = 6) or *functional* experts (n = 4) perceived MaaS differently, we conduct an exploratory Mann-Whitney U test on each composite metric score. We also include the rank-biserial correlation to show by how much does the technical or functional expert group rank one metric higher than the other. The results are shown in Table 17. In general, no statistically significant group differences were found for any metric with p > 0.05 in all cases. With r of +0.63 and +0.54 for resource efficiency and delivery respectively, and p values of 0.12 and 0.19 respectively, only moderate indication of difference is suggested in favour of the *technical* experts, yet with p > 0.05 remains not significant enough. The results suggest that both *technical* and *functional* expert groups ranked MaaS evaluation metrics similarly, with small indications in favour of the *technical* group for resource efficiency and delivery. The positive indications remain statistically insignificant, and the results remain exploratory due to the small sample size.

Table 17: Mann-Whitney U Test Results

	U	**p**	**Rank biserial r**
Collaboration	10.0	0.74	+0.17
Performance	11.0	0.91	+0.08
Resource Efficiency	4.5	0.12	+0.63
End-user Involvement	9.5	0.66	+0.21
Delivery	5.5	0.19	+0.54

Lastly, to explore if experts who rank one metric positively also rank the other metrics similarly, we performed Spearman rank correlation test due to having a small survey sample size and the data as ordinal medians. The results are displayed in the form of a heat map in Figure 16. Considering this is a two-tailed test, p < 0.05 is required for a positive correlation. With an α of 0.05 for two-tailed test, and n = 10, we get the critical value of ±0.648, or 0.65 if rounded. This means that only values above 0.65 can be considered statistically significant. In our results, only performance and resource efficiency share a correlation p value higher than 0.65 with p ≈ 0.71. Experts who see MaaS improve performance also tend to see MaaS improve resource efficiency. A moderate collaboration and delivery link can also be observed with correlation p ≈ 0.61, however fails to cross the significancy threshold of 0.65. A small link indication can be observed between resource efficiency and delivery with correlation p ≈ 0.43, but also remains below the threshold of 0.65.

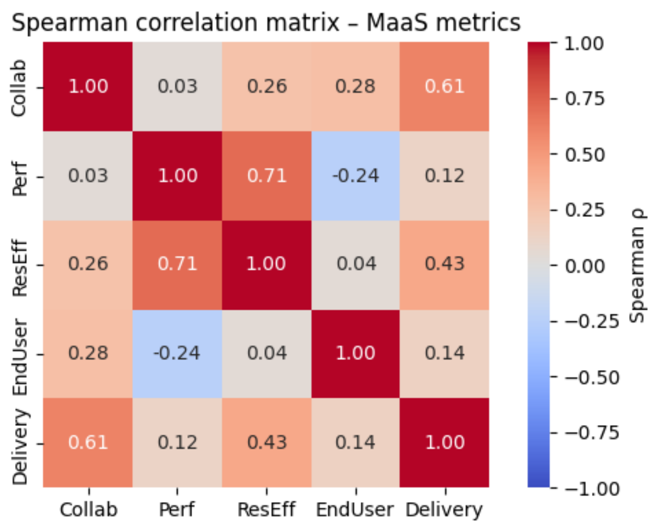


Figure 17: Spearman Correlation Matrix Heat Map

Answers to open questions (Appendix A) by experts following each Likert-scale block largely reinforce the statistical results, supplying concrete examples of both positive and negative remarks on MaaS (Table 18). In summary, experts praised MaaS for facilitating transparent communication, integrated team collaboration, accelerated development cycles through CI/CD pipelines and continuous feedback loops, reuse of platform components, user involvement, and automated testing. While negative remarks were scarce, they help identify critical areas for improvement. Some of these areas include over-reliance on automation, end-user fatigue due to constant methodology-related activities, licensing and resource allocation, and end-user platform adoption and usage.

Table 18: Compiled Expert Insights

	Answer^1	Answer^2	Answer^3	Answer^n
Collaboration	"Having all stories structurally managed in the Agile Development on the platform, helps tremendously. All project members always have access to these and the status is visible to everybody. This is crucial for coordination."	"Traditional methods (waterfall) isolate planning, development, and delivery. MaaS improves collaboration by integrating those aspects and more agile approaches and CI/CD from the start, which I think should reduce misalignment and improve responsiveness to feedback"	"Agile methods are important component of MaaS. The agile methods alone (without the MaaS frame around it) aren't as effective"	"I doubt MaaS would improve cross-functional team collaboration. From my experience teams from non-software development domains have hard time understanding agile and other methodologies that are native to our field. And in my opinion, it's hard to understand those unless you followed them at least once"

Performance	“MaaS is a comprehensive method for a development. From my perspective it not a specific component which makes the difference but the combination of all aspects”	“The use of CI/CD pipelines and collaborative tools, such as Git and ServiceNow's Agile 2.0, improves the output of the software development teams specifically”	“Traditional methods often delay feedback and lack automation, leading to slower delivery and higher risk of rework”	“Simultaneous use of CI/CD and SCRUM sprints means cycles can be kept short as improvements on features are put on the backlog and picked up again for the next sprint”
Resource Efficiency	“Utilization is higher per resource compared to other traditional and agile methods”	“Development members are efficiently optimised as they are put to work for CI/CD and Sprints and they are also needed for design, which now happens continuously”	“- Time of developers - Time of project managers (less thinking about methodology etc. more about following the workflow)”	“The beginning phase seems to be a bit stricter, as it follows a more traditional method. This may create idle time for developers, which isn't as productive as full Agile”
End-user Involvement	“Traditional methods generally consider feedback only during the last stages of the project. MaaS seems to promote end-user involvement during the development stage, which improves flexibility”	“End-users can spot illogical functionality and weird behaviour much quicker, making decisions for rework or revisioning easier thanks to continuous testing and feedback”	“For example, a previously determined user trend may not be accurate due to lack of user feedback during the planning phase. New analysis, using short feedback loops during the development phase, can lead to the change of functional requirements”	“With the involvement of end users throughout the whole development process, bottlenecks and issues can be identified early and it can be avoided that late rework needs to be done”
Delivery	“CI/CD combination with development sprints contributes most towards quality and timeline management”	“Different methodologies for different stages of the project, each one that is most suitable for the stage”	“Holistic view - people, process, technology”	“Consideration of end-user and other stakeholders during development, continuous delivery, short feedback loops, and backlog visibility”

In conclusion, to answer **RQ 3**, we used five types of analysis – descriptive statistics, qualitative expert insights, one-sample Wilcoxon test, Mann-Whitney U test for group comparison, and the Spearman rank correlations. In general, all medians are above the neutral point of 4.0, indicating a positive outlook on all MaaS evaluation metrics from our experts, supported further by Wilcoxon test. Mann-Whitney U test showed that both *technical* and *functional* experts do not differ in their ratings with the lowest p value being p = 0.12, or in other words no statistical significance was achieved to show otherwise. The Spearman rank correlation test found a strong positive relationship between resource efficiency and performance, a relatively positive link between collaboration and delivery, and a weak positive link between resource efficiency and delivery. Convergingly, the results from all statistical tests indicate empirical support for MaaS as a hybrid methodology which improves development outcomes for PaaS software development within the dimensions of collaboration, performance, resource efficiency, end-user involvement, and delivery. Expert insights achieved via open questions further support quantitative findings, as well as provide avenues for improvement. These results, however, are to be considered exploratory and non-decisive due to the small sample size of experts, and the lack of a real-world project scenario.

8 Final Remarks

8.1 Discussion

In this thesis we set out to analyse and subsequently evaluate software development methods tailored for PaaS platforms, culminating in the proposal and subsequent application study of MaaS – a hybrid software development methodology adherent to PaaS software development principles (Chapter 3). It combines the structured planning approach of traditional methodologies, the iterative nature and flexibility of Agile methodologies, and integrates it with CI/CD delivery pipelines. To achieve this goal, we followed Design Science structures outlined by Peffers et al. and Offermann et al. [7, 8], where we identified the problem – fundamentally different software development processes and requirements between traditional and PaaS environments; presented a solution based on available scientific literature in the form of MaaS, demonstrated it through a mock project using ServiceNow as our PaaS example, and lastly evaluated it via expert surveys and subsequent result analysis (Chapter 7). While the findings suggest that MaaS improves software development on PaaS platforms in areas of collaboration, end-user involvement, performance, resource efficiency and delivery effectiveness, the results must be considered as solely exploratory due to limitations with regards to applying MaaS on a real-world project, and a small sample size of experts for evaluation. Additionally, we were not able to find sufficient scientific literature on this topic, therefore the theoretical background of this thesis might be relatively weak.

While MaaS inherently does not propose any novel methods, it presents itself as a construction of methods best suited for PaaS-specific software development, taking structure from traditional methods, using Agile process models to facilitate development due to their flexible, iterative, and collaborative nature, supporting the development process model with an equally iterative and user-centric feedback CI/CD pipeline, and applying a relatively novel project methodology and five environment model of Rugby. Across all evaluation metrics of MaaS, namely collaboration, end-user involvement, performance, resource efficiency and delivery effectiveness, expert Likert-scale survey results (Section 7.4) show a unanimous positive outcome (Table 15). With the neutral midpoint score being 4.0, the lowest median scores were recorded for collaboration and performance at 4.75, and the highest at 5.50 for end-user involvement.

Based on the open survey questions (Appendix A) (Table 18) particularly positive expert sentiment identified integration of CI/CD with Agile process models, structured planning and responsibilities, iterative development, and continuous feedback loops as critical success factors of MaaS. End-user involvement was significantly praised over all other metrics, as MaaS involves users in planning and development activities through workshops and short feedback loops. Having all development activities situated in a single platform, which provides a plethora of built-in services for development and collaboration, and is accessible to both development teams and users, allows for rapid feedback provision and improved resource utilization. Subsequently, similar sentiment was directed towards resource efficiency, as it is improved due to pre-built components and services offered by PaaS platforms, meaning that the need for third party software, tools, or other supporting infrastructure is drastically reduced. Delivery and performance were strongly praised for CI/CD automation, fast build-test-deploy cycles, and automated testing promoting frequent and high-quality application deployments.

8.2 Limitations

While the results indicate on overall positive support for the benefits of MaaS, several areas of improvement are also discovered. End-user involvement and collaboration are enabled through tools offered by PaaS platforms, and structure offered by MaaS, however, are not effective if the relevant project stakeholders and users do not actively use the platform. Without constant participation, the benefits offered by PaaS platforms and MaaS are underutilised. Stronger governance structures and user adoption trainings in the planning phase, or during an initial preliminary sprint, through workshops or change management, are required to ensure all stakeholders fully engage with the platform and its services. Similarly, some experts doubt that MaaS would improve cross-functional team collaboration, as non-software development teams of either the client or the software producer may not be familiar with PaaS platforms or not using them at all. The same could be said with regards to the usage and understanding Agile or other types of methodologies. If the project is long and complex, users also may experience fatigue in constant feedback provision. Therefore, depending on project requirements and timeline, suitable pacing should be introduced and agreed upon during the planning phase. CI/CD automation was particularly praised for automated testing, however, experts cautioned that it also may disguise poor testing practices. In our demonstration in Chapter 6, we presented a simple process, where automated test suites would run on the test environment, and upon successful completion, the piece of software would be deployed to the production environment. Upon testing failure, the deployment would be cancelled, and the software rolled back. Ideally, automated testing should run side by side with manual testing, as, to the understandable disappointment of developers, it remains the best practice. Once a story is completed, the responsible developer should also conduct testing such that the story acceptance criteria is met. User acceptance testing (UAT) is also highly recommended, where test cases and test steps are made for any piece of completed software and completed by the users on the testing environment during the final stages of a project lifecycle. Considering the use of PaaS platforms, scope creep can produce unexpected negative outcomes in terms of licensing costs, particularly when working on rapid prototyping type projects. To avoid this, strict scope planning, and resources which affect licensing costs (number of tables, users, etc.), should be established early on and adhered to in the planning phase.

8.3 Benefits

Compared to solely Agile models, MaaS is more versatile, suitable both for complex projects requiring rigid planning, as well as small-scale projects requiring a rapid time-to-market. It provides flexibility in the choice the project relevant Agile process model based on requirements of the project and incorporates a structured and clearly defined planning phase to tackle topics of governance, scope, requirements, and other project-relevant documentation. Lastly, it leverages the five-environment model of Rugby, built specifically for Agile process models, providing a structured yet flexible framework which traditional Agile neglects. Compared to traditional methods, MaaS improves delivery by integrating flexibility via iterative development, CI/CD pipelines, and user feedback while still retaining the structured phase approach between planning, development, and maintenance. The development phase is changed to employ an Agile process model, enabling rapid prototyping, frequent feedback, and flexible adaptation to changing requirements. MaaS incorporates the strengths of PaaS platforms, such as ServiceNow or Mendix, namely low-code or no-code functionalities, workflow automation, templates, integrated Agile project management, and CI/CD support, all of which neither Agile nor traditional methodologies account for. In essence, MaaS is designed to work on all possible projects of all scale and scope intended to be developed on PaaS platforms.

8.4 Future Work

The main future research recommendation would be to polish the proposed hybrid MaaS methodology to a more complete form and conduct validation on a real-life project with actual development teams, project, and stakeholders, as well as evaluation with a much broader number of experts to achieve meaningful insights. A real example would allow researchers to gather data on stakeholder behaviour and adaptability, defect or bug rates, development speed, user satisfaction, and possibly return on investment (if comparing the use of PaaS platforms with MaaS or traditional environments with non-MaaS methodologies). This would provide means to more accurately evaluate MaaS against traditional, Agile, or other hybrid methodologies. Additionally, researchers should explore the organisational change management imposed by MaaS on topics such as adoption barriers, organisational culture alignment, and learning curves. As it stands currently, MaaS is nothing more than a proposal, demonstrated on a mock project, and evaluated by a limited number of expert respondents. While achieving a generally positive receptance of MaaS and its methods, the results can only be considered as exploratory and not indicative of any real impact.

With the flexible nature of MaaS, and its vision of supporting PaaS-specific software development projects of all scopes and sizes, a promising and novel direction of future research lies in the development of a modular methodology framework. An adaptive model where software development methodology components can be selectively assembled and interchanged depending on specific project variables. MaaS partially implements this on a high level with the ability to choose the appropriate Agile process model for the development phase depending on project requirements. Considering the existence of tens, if not hundreds of different methodologies, consistently choosing the most appropriate one inherently would introduce limitations, which are otherwise likely supported by a different methodology. A large decision matrix could be created to support the choice of particular methodology components, regardless of project phase, based on such factors as project requirements, team size, available tools and technologies, and other relevant characteristics.

8.5 Conclusion

The primary goal of this thesis was to conduct an application case study to evaluate a software development methodology which can serve the software development principles of PaaS platforms more efficiently and effectively, than their established counterparts across five key dimensions: collaboration, end-user involvement, performance, resource efficiency, and delivery effectiveness.

The results (Chapter 7) produced a positive sentiment towards MaaS across all five evaluation dimensions, with the highest median score attributed to end-user involvement of 5.50, and lowest median score to collaboration and performance of 4.75, both above the neutral midpoint of 4.00. These results, while answering **RQ 3**, are to be considered exploratory and non-decisive due to the small sample size of experts (N = 10), and the lack of a real-world project scenario. Expert sentiment gathered through open questions highlight critical success factors of MaaS, namely integration of CI/CD with Agile methods, automated testing, structured planning and responsibilities, iterative development, and continuous feedback loops. MaaS provides the structured discipline, sequential phases, and rigid documentation characteristic of traditional methodologies, while enhancing the flexibility, iterative feedback, collaboration, and speed characteristic of Agile methodologies. The capability to effectively integrate CI/CD pipelines, leverage the strengths of services offered by PaaS platforms, while maintaining a high level of stakeholder collaboration and adaptability makes MaaS particularly advantageous for software projects typical to PaaS platforms.

In summary, this thesis does not provide any objective evidence with regards to the effectiveness of a software development methodology which can serve the software development principles of PaaS platforms more efficiently and effectively than their established counterparts. However, it does provide substantial exploratory evidence to support a further refined study on the effectiveness of MaaS by applying it in a real project setting and evaluating it with a substantially larger group of experts.

"*MaaS brings the process, people, and technology together in a way which hybrid methods fail to do*".

Acknowledgements

I would like to express my sincere gratitude to Dr. L. Ferreira and Dr. G. Sedrakyan for their guidance, support, and an infinite pool of patience towards myself and my work process!

References

[1] – Ņikiforova, O., Babris, K., Madelāne, L. Expert Survey on Current Trends in Agile, Disciplined and Hybrid Practices for Software Development. Applied Computer Systems, 2021, Vol. 26, No. 1, pp. 38-43. ISSN 2255-8683. e-ISSN 2255-8691. Available from: doi:10.2478/acss-2021-0005

[2] – Chnar Mustafa Mohammed & Subhi R.M Zeebaree, 2021. "Sufficient Comparison Among Cloud Computing Services: IaaS, PaaS, and SaaS: A Review," International Journal of Science and Business, IJSAB International, vol. 5(2), pages 17-30.

[3] – Wulf, Frederik & Lindner, Tobias & Strahringer, Susanne & Westner, Markus. (2021). IaaS, PaaS, or SaaS? The Why of Cloud Computing Delivery Model Selection - Vignettes on the Post-Adoption of Cloud Computing. 10.24251/HICSS.2021.758.

[4] – G. Sedrakyan, M.E. Iacob, J. Hillegersberg, "Towards LowDevSecOps Franework for Low-Code Development", integrating Process-Oriented Recommendations for Security Risk Management

[5] – O. Gass, H. Meth and A. Maedche, "PaaS Characteristics for Productive Software Development: An Evaluation Framework," in IEEE Internet Computing, vol. 18, no. 1, pp. 56-64, Jan.-Feb. 2014, doi: 10.1109/MIC.2014.12.

[6] – Krancher, O., Luther, P., & Jost, M. (2018). Key Affordances of Platform-as-a-Service: Self-Organization and Continuous Feedback. In Journal of Management Information Systems (Vol. 35, Issue 3, pp. 776–812). Informa UK Limited. https://doi.org/10.1080/07421222.2018.1481636

[7] – Offermann, P., Levina, O., Schönherr, M., & Bub, U. (2009). Outline of a design science research process. Proceedings of the 4th International Conference on Design Science Research in Information Systems and Technology - DESRIST '09. doi:10.1145/1555619.1555629

[8] – Peffers, Ken & Tuunanen, Tuure & Rothenberger, Marcus & Chatterjee, S.. (2007). A design science research methodology for information systems research. Journal of Management Information Systems. 24. 45-77.

[9] – M. N. Aydin, Z. N. Perdahci, I. Safak and J. (Jos) van Hillegersberg, "Metadata Action Network Model for Cloud Based Development Environment", Advances in Intelligent Systems and Computing, vol. 1161, pp. 531-543, 2020.

[10] – Shu-Qing, Z., & Jie-Bin, X. (2010). The Improvement of PaaS Platform. 2010 First International Conference on Networking and Distributed Computing. doi:10.1109/icndc.2010.40

[11] – Singh, A., Sharma, S., Kumar, S. R., & Yadav, S. A. (2016). Overview of PaaS and SaaS and its application in cloud computing. 2016 International Conference on Innovation and Challenges in Cyber Security (ICICCS-INBUSH). doi:10.1109/iciccs.2016.75423

[12] – Walraven, S., Truyen, E., & Joosen, W. (2013). Comparing PaaS offerings in light of SaaS development. Computing, 96(8), 669–724. doi:10.1007/s00607-013-0346-9

[13] – Rohil, Harish & Syan, Manisha. (2012). Analysis of Agile and Traditional Approach for Software Development. International Journal of Latest Trends in Engineering and Technology. 1. 1- 10.

[14] – Fylaktopoulos, G., Goumas, G., Skolarikis, M. et al. An overview of platforms for cloud based development. SpringerPlus **5**, 38 (2016). https://doi.org/10.1186/s40064-016-1688-5

[15] – Bulajic, Aleksandar & Sambasivam, Samuel & Stojic, Radoslav. (2013). An Effective Development Environment Setup for System and Application Software. Issues in Informing Science and Information Technology. 10. 037-066. 10.28945/1795.

[16] – Singhto, W., & Phakdee, N. (2016). Adopting a combination of SCRUM and Waterfall methodologies in developing Tailor-made SaaS products for Thai Service and manufacturing SMEs. 2016 International Computer Science and Engineering Conference (ICSEC). doi:10.1109/icsec.2016.7859882

[17] – Ruparelia, N. B. (2010). Software development lifecycle models. ACM SIGSOFT Software Engineering Notes, 35(3), 8. doi:10.1145/1764810.1764814

[18] - Krusche, S., Alperowitz, L., Bruegge, B., & Wagner, M. O. (2014). Rugby: an Agile process model based on continuous delivery. In Proceedings of the 1st International Workshop on Rapid Continuous Software Engineering. ICSE '14: 36th International Conference on Software Engineering. ACM. https://doi.org/10.1145/2593812.2593818

[19] – Cardozo, Elisa & Neto, Benito & Barza, Alexandre & França, César & Silva, Fabio. (2010). SCRUM and Productivity in Software Projects : A Systematic Literature Review. 10.14236/ewic/EASE2010.16.

[20] – Srivastava, A., Bhardwaj, S., & Saraswat, S. (2017). SCRUM model for agile methodology. 2017 International Conference on Computing, Communication and Automation (ICCCA). doi:10.1109/ccaa.2017.8229928

[21] – Haig-Smith, T., & Tanner, M. (2016). Cloud Computing as an Enabler of Agile Global Software Development. In InSITE Conference. InSITE 2016: Informing Science + IT Education Conferences: Lithuania. Informing Science Institute. https://doi.org/10.28945/3477

[22] – Nath, Mahendra & Muralikrishnan, Jayashree & Sundarrajan, Kuzhanthaiyan & Varadarajanna, Madhu. (2018). Continuous Integration, Delivery, and Deployment: A Revolutionary Approach in Software Development. 5. 185-190 https://www.researchgate.net/publication/343760242_Continuous_Integration_Delivery_and_Deployment_A_Revolutionary_Approach_in_Software_Development

[23] – Vijayasarathy, L. R., & Butler, C. W. (2016). Choice of Software Development Methodologies: Do Organizational, Project, and Team Characteristics Matter? IEEE Software, 33(5), 86–94. doi:10.1109/ms.2015.26

[24] – Kuhrmann, M., Diebold, P., Munch, J., Tell, P., Trektere, K., Mc Caffery, F., … Prause, C. (2018). Hybrid Software Development Approaches in Practice: A European Perspective. IEEE Software, 1–1. doi:10.1109/ms.2018.110161245

[25] Mushtaq, Zaigham & Rizwan, M. & Qureshi, M. Rizwan. (2012). Novel Hybrid Model: Integrating Scrum and XP. International Journal of Information Technology and Computer Science. 4. 10.5815/ijitcs.2012.06.06.

[26] - Cohen, B. (2013). PaaS: New Opportunities for Cloud Application Development. Computer, 46(9), 97–100. doi:10.1109/mc.2013.323

[27] – Hacaloglu, T., Eren, P. E., Mishra, D., & Mishra, A. (2015). A Software Development Process Model for Cloud by Combining Traditional Approaches. Lecture Notes in Computer Science, 421–430. doi:10.1007/978-3-319-26138-6_45

[28] – Joshi, Ankur & Kale, Saket & Chandel, Satish & Pal, Dinesh. (2015). Likert Scale: Explored and Explained. British Journal of Applied Science & Technology. 7. 396-403. 10.9734/BJAST/2015/14975.

Appendix A: Survey Questions

Collaboration

- To what extent do you agree that MaaS improves collaboration among cross-functional teams on PaaS projects?
    - Can you describe a particular example, where MaaS improved cross-functional team collaboration? (Optional: Free-text answer)
- To what extent do you agree that is MaaS is effective in facilitating coordination among different cross-functional teams?
    - Which aspects or components of MaaS contribute most to improved coordination? (Optional: Free-text answer)
- To what extent do you agree that MaaS fosters more effective communication between developers and stakeholders?
    - Can you explain any changes you observed in the communication process? (Optional: Free-text answer)
- Compared to traditional software development methods, does MaaS improve collaborative processes with regards to PaaS-specific software development projects?
    - Which differences have you noticed in collaborative interactions between MaaS and traditional methods? (Optional: Free-text answer)
- Compared to Agile software development methods, does MaaS improve collaborative processes with regards to PaaS-specific software development projects?
    - Which differences have you noticed in collaborative interactions between MaaS and Agile methods? (Optional: Free-text answer)
- Compared to other hybrid software development methods you have experienced, does MaaS improve collaborative processes with regards to PaaS-specific software development projects?
    - Which differences have you noticed in collaborative interactions between MaaS and other hybrid methods? (Optional: Free-text answer)

Performance

- To what extent do you agree that MaaS improves development speed and performance?
    - Can you describe which MaaS steps or components facilitate improved development performance? (Optional: Free-text answer)
- To what extent do you agree that MaaS accelerates and shortens development iteration cycles?
    - Can you describe which aspect of MaaS contributes to shorter development iteration cycles? (Optional: Free-text answer)
- To what extent do you agree that MaaS leads to higher productivity in PaaS-specific software development projects?
    - Which specific process improvements of MaaS have you noticed that contribute to increased productivity? (Optional: Free-text answer)
- Compared to traditional software development methods, does MaaS improve performance outcomes with regards to PaaS-specific software development projects?
    - Which key differences have you noticed in performance outcomes between MaaS and traditional methods? (Optional: Free-text answer)

- Compared to Agile software development methods, does MaaS improve performance outcomes with regards to PaaS-specific software development projects?
    - Which key differences have you noticed in performance outcomes between MaaS and Agile methods?
- Compared to other hybrid software development methods you have experienced, does MaaS improve performance outcomes with regards to PaaS-specific software development projects?
    - Which key differences have you noticed in performance outcomes between MaaS and other hybrid methods?

Resource Efficiency

- To what extent do you agree that MaaS optimises the use of resources in PaaS-specific software development projects?
    - Can you describe which recourses is MaaS most efficient in optimising? (Optional: Free-text answer)
- To what extent do you agree that MaaS reduces redundant or ineffective software development processes?
    - Can you provide a particular example, where MaaS eliminates a redundant process?
- To what extent do you agree that MaaS leverages PaaS-specific built-in services to improve resource efficiency? (Optional: Free-text answer)
    - Which built-in services do you think MaaS utilises most effectively? (Optional: Free-text answer)
- Compared to traditional software development methods, does MaaS demonstrate higher resource efficiency with regards to PaaS-specific software development?
    - Which differences have you observed in resource use between MaaS and traditional methods? (Optional: Free-text answer)
- Compared to Agile software development methods, does MaaS demonstrate higher resource efficiency with regards to PaaS-specific software development?
    - Which differences have you observed in resource use between MaaS and Agile methods? (Optional: Free-text answer)
- Compared to other hybrid methods you have experienced, does MaaS improve on resource utilisation and cost savings?
    - Which differences have you observed in resource use between MaaS and other hybrid methods? (Optional: Free-text answer)

End-User Involvement

- To what extent do you agree that MaaS facilitates active end-user involvement throughout the development cycle?
    - Can you describe which MaaS components or processes have allowed for greater end-user involvement? (Optional: Free-text answer)
- To what extent do you agree that MaaS effectively utilises end-user feedback into iterative development cycles?
    - Can you describe in which part of the development iteration does the end-user feedback produce the greatest effect? (Optional: Free-text answer)

- To what extent do you agree that MaaS is effective in enabling end-users to influence project outcomes?
    - Can you provide an example on how end-user involvement in MaaS can influence project or development decisions? (Optional: Free-text answer)
- Compared to traditional software development methods, does MaaS encourage greater end-user engagement and feedback?
    - Which improvements have you observed in engagement and feedback between MaaS and traditional methods? (Optional: Free-text answer)
- Compared to Agile software development methods, does MaaS encourage greater end-user engagement and feedback?
    - Which improvements have you observed in engagement and feedback between MaaS and Agile methods? (Optional: Free-text answer)
- Compared to other hybrid methods you have experienced, does MaaS encourage greater end-user engagement and feedback?
    - Which improvements have you observed in engagement and feedback between MaaS and other hybrid methods? (Optional: Free-text answer)

Delivery Effectiveness

- To what extent do you agree that MaaS improves the quality and timelines of software releases in PaaS-specific software development?
    - Which aspects of MaaS do you believe are most significant in contributing towards release quality and timelines? (Optional: Free-text answer)
- To what extent do you agree that MaaS improves delivery sanitation and reduces the number of post-release defects?
    - Can you describe which MaaS components facilitate the reduction in post-delivery defects? (Optional: Free-text answer)
- To what extent do you agree that MaaS is effective in aligning delivered features with customer expectations?
    - Which MaaS features do you think contribute most to meeting customer expectations? (Optional: Free-text answer)
- Compared to traditional software development methods, does MaaS deliver superior outcomes in terms of delivery effectiveness?
    - Which key factors have you observed that contribute towards superior delivery effectiveness between MaaS and traditional methods? (Optional: Free-text answer)
- Compared to Agile software development methods, does MaaS deliver superior outcomes in terms of delivery effectiveness?
    - Which key factors have you observed that contribute towards superior delivery effectiveness between MaaS and Agile methods? (Optional: Free-text answer)
- Compared to other hybrid software development methods you have experienced, does MaaS deliver superior outcomes in terms of delivery effectiveness?
    - Which key factors have you observed that contribute towards superior delivery effectiveness between MaaS and other hybrid methods? (Optional: Free-text answer)