Journal of Theoretical and Applied Information Technology
31st May 2024. Vol.102. No. 10

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

BANYAN AGILE: A NEW APPROACH FOR MONOLITH AND MICROSERVICE DEVELOPMENT

VICTOR[1] , MARIA SERAPHINA ASTRIANI[2]

[1]Computer Science Department, BINUS Graduate Program - Master of Computer Science, Bina Nusantara University, Jakarta, Indonesia 11480

[2]Computer Science Department, School of Computing and Creative Arts, Bina Nusantara University, Jakarta, Indonesia 11480

E-mail: [1]victor010@binus.ac.id, [2]seraphina@binus.ac.id

ABSTRACT

This research explores the challenges of implementing the Software Development Life Cycle (SDLC) in developing software with monolith and microservice architectures. The SDLC method serves as a framework guiding the stages of software creation, completion, and maintenance. Challenges arise when applying SDLC to software development teams transitioning from monolith to microservice architectures. Factors such as synchronization among teams, a lack of expertise in implementing specific SDLC methods, and escalating project costs become major problems. This research seeks solutions by proposing a new approach called Banyan Agile. Banyan Agile integrates SDLC principles with the flexibility of agile methods, creating a framework that can address these challenges and facilitate effective software development. The findings reveal that the implementation of Banyan Agile within PT. ASD has positively impacted collaboration among cross-functional teams and overall project productivity. Despite encountering challenges during the implementation phase, the identification of these hurdles highlights areas for improvement to optimize project outcomes.

Keywords: *Software Development, Agile, Monolith, Microservice, Strategic Planning*

1. INTRODUCTION

1.1. Background

In the dynamic landscape of software development, the choice of architecture and methodology plays a crucial role in the success of projects. The microservice architecture fosters flexibility and modularity, empowering teams to develop, test, and deploy each microservice autonomously. This approach streamlines rapid iteration and facilitates the adoption of diverse technologies and configurations tailored to specific requirements [1]. The transition from traditional monolithic structures, characterized by tightly integrated components, to the more modular and scalable microservice architectures has become a common pursuit for many forward-thinking organizations. This architectural shift is driven by the desire to enhance flexibility, scalability, and overall efficiency in meeting the ever-evolving demands of the modern digital landscape.

As organizations embark on this transformative journey towards microservices, they encounter many challenges ranging from complexity of the monolith architecture, complexity of the data repository, organizational culture of the company, experience of the software development team in creating microservices, and the ability to make a proper division of the monolith architecture to create microservices [2]. Microservices, with their decentralized nature, promise improved agility and easier maintenance, yet the journey towards their adoption requires careful consideration of various factors such as service communication, data consistency, and system observability. Conversely, the prevalence of monolithic structures, though deemed more straightforward in design, often poses challenges in scalability and adaptability, hindering the agility required to respond to rapidly changing business needs.

Amidst this architectural evolution, the implementation of effective Software Development Life Cycle (SDLC) methodologies becomes paramount. Each SDLC method has strength and weaknesses, and that they are appropriate for a variety of situations [3]. The waterfall methodology represents a sequential approach to development, necessitating the completion of each phase before the subsequent one can commence [4]. Agile methodologies involve iterative and incremental processes, where software is developed in brief cycles, each resulting in a functional product [5]. The traditional SDLC models, while successful, may fall short in accommodating the distinctive characteristics and requirements of monolithic and microservice development.

To summarize, the adoption of either monolith or microservices significantly impacts software development. However, employing an SDLC that inadequately supports the simultaneous development of both architectures can impede software progress and have repercussions on an organization's finances and growth trajectory. As organizations grapple with the need for an SDLC framework that can seamlessly cater to these diverse architectural paradigms, there arises a compelling opportunity to innovate and formulate a novel approach that transcends the limitations of existing models.

The SDLC is applied in software project management to guarantee timely, budget-conscious completion of projects while meeting the necessary quality standards [6]. In this context, the research endeavors to propose and explore a new SDLC model tailored to address the intricacies of both monolithic and microservice development. By acknowledging the unique challenges posed by each architecture and synthesizing the strengths of various existing methodologies, this research aims to provide a comprehensive and adaptable framework. Such a framework should not only facilitate the successful development and deployment of software systems but also pave the way for a harmonious coexistence of monolithic and microservice components within the same organizational ecosystem.

Through a meticulous examination of the current state of software development practices, this research seeks to contribute valuable insights that resonate with organizations navigating the complexities of architectural choices. It aspires to be a guiding beacon for developers, architects, and project managers alike, offering pragmatic solutions and best practices that align with the nuances of both monolithic and microservices landscapes.

In essence, this research aims to bridge the gap between evolving architectural paradigms and effective SDLC methodologies, ushering in a new era where organizations can harness the benefits of both monolithic and microservice development seamlessly and with confidence.

1.2. Challenges in traditional SDLC for monoliths and microservices

This research delves into the multifaceted challenges encountered by software development teams tasked with navigating the intricacies of both monolith and microservice architectures, with a central focus on achieving an efficient Software Development Life Cycle (SDLC) implementation. The contemporary software development landscape witnesses a pivotal shift as organizations strive to harness the benefits of microservices' scalability and agility while grappling with the established structures of monolithic systems. From the author's observations and interviews with employee and managers working at PT. ASD in software development, challenges were identified in maintaining, developing, and migrating several software components within both monolithic and microservices architectures.

The crux of the challenge lies in striking a delicate balance between the rigidity required by a structured SDLC and the adaptive flexibility demanded by the dynamic nature of microservices. This intricate dance poses a unique set of hurdles that necessitate a nuanced and comprehensive exploration. Key among these challenges are the complexities of team synchronization, wherein the divergent approaches demanded by monoliths and microservices can lead to collaboration gaps and operational inefficiencies.

Moreover, the prevalence of skill gaps within development teams presents a significant obstacle. The diverse skill sets required to navigate traditional SDLC methodologies for monolithic architectures versus the newer paradigms demanded by microservices can lead to a fragmentation of expertise. This not only hinders the seamless integration of both architectures but also raises questions about the efficient utilization of resources and the potential need for additional training.

Financial implications further amplify the challenges, especially in the context of project expansion or transition. Organizations often find themselves at a crossroads, weighing the costs associated with scaling a monolithic system against the potential benefits of adopting a microservices architecture. This financial conundrum adds a layer of complexity to SDLC decision-making, requiring a careful examination of return on investment, long-term sustainability, and the adaptability of existing resources.

This research aims to dissect and understand the multifaceted challenges arising from the convergence of monolithic and microservice architectures within the realm of SDLC implementation. By delving into the intricacies of team dynamics, skill development, and financial considerations, the goal is to provide actionable insights that empower software development teams to navigate this complexity successfully and derive optimal value from their chosen architectural approaches.

2. LITERATURE REVIEW

In software development, the need for team management methods and software architecture tailored to the requirements and conditions of a team is essential. The requirements and conditions of one team may vary from those of another. Some teams may have the need to develop software within a short time frame, while others may require maintaining existing software to operate effectively. To meet these diverse needs and conditions, different planning methods are necessary.

2.1. Overview of Existing SDLC Models

In this study, several existing Software Development Life Cycle (SDLC) methods will be selected for comparison. The chosen methods for examination include prototype, agile, and scrum.

2.1.1. Prototype

In 2022, a study by Maulida (2022) asserts that the development method of the prototype system consist of several stages, incorporating system analysis, structured design, and black box testing methods [7].

According to a study by Bhatnagar (2015), a prototype is an evolving model. Its development process involves gathering requirements, followed by developers creating a brief design, and the results are then discussed again with the users [8].

2.1.2. Agile

In 2001, The Agile Alliance engaged in discussions and formulated a manifesto known as The Agile Manifesto. This manifesto serves as the foundation for the development of future agile methodologies (The Agile Manifesto, 2001) [9].

A study conducted by Megargel, Shankararaman, and Walker in 2020 concludes that companies should consider adopting Agile practices, utilizing the cloud to expedite software development. The study also suggests that transforming existing monolithic software into a microservices architecture is better achieved through a migration method, gradually transitioning from existing program segments until the entire monolithic program can be discontinued [10].

Another study by Dirk Beerbaum in 2023 concludes that diverse Agile methodologies share a commonality, emphasizing shared ownership and autonomous team collaboration. The study also notes that successful Agile implementation requires clear communication, training, and support from management, necessitating organizational changes to embrace Agile practices [11].

2.1.3. Scrum

In a study by Kadenic, Koumaditis, and Junker-Jensen (2023), the composition of team factors, such as allocation, member substitution, member capabilities, and self-regulation tendencies, significantly impacts the implementation of Scrum. All predefined roles within Scrum play a crucial role. The developer's ability to adapt to planning, the product owner's prioritization skills, and the Scrum master's ability to ensure adherence to plans all influence the success of Scrum implementation [12].

In another study (Verwijs, Russo, 2022), an examination of 13 case studies identifies five high factors, including responsiveness, stakeholder concern, ongoing development, the team's ability to achieve goals, and management support. Additionally, 13 low factors such as goal focus, sprint review quality, and others affect the

effectiveness of Scrum teams, indicated by their ability to satisfy stakeholders and maintain high team morale [13].

A study by Ozkan & Tarhan (2020) concludes with an assessment that while Scrum is intended to support scaling and agility as team size increases, it presents new challenges that question its effectiveness. The study suggests considering design changes or alterations to the Agile Manifesto and its scalability principles [14].

In another study (Asma Akhtar, Brra Bakhtawar, and Samia Akhtar, 2022), a comparison between Extreme Programming and Scrum reveals numerous similarities and differences. These distinctions, when combined, could potentially form an experiment leading to a more effective methodology [15].

2.2. Monolith Development

Expanding on the discussion regarding monolithic development, the year 2022 witnessed a study by Blinowski, Ojdowska, and Przybylek, emphasizing the monolith's characterization from the operating system standpoint. Described as an application operating within a single process on the application server, the monolith is praised for its inherent simplicity. Deployment becomes more straightforward, with the database conveniently consolidated in a single location, among other benefits [16].

However, as applications evolve and expand, a notable shift occurs. The process of transforming a monolithic application becomes intricate and challenging. The initial advantages in simplicity face the complexities associated with scalability and adaptability. This transformational phase marks a critical juncture where the once-straightforward system architecture encounters hurdles, necessitating careful consideration and strategic planning.

2.3. Microservices Development

A study conducted by Blinowski, Ojdowska, and Przybylek in 2022 delves into the comparison between monolith and microservice architectures, leveraging various cloud providers. The research concludes that a monolithic architecture is more suitable for smaller systems that do not require extensive support for a large user base [17].

On a related note, a study by Abgaz, McCarren, Elger, et al. in 2023 explores the intricate task of decomposing a monolith into microservices. The study highlights the complexity of this process and notes the absence of a well-organized guideline that spans from the initial steps to the final stages [18].

2.4. Comparative Analysis

Table 1: Comparison Prototype, Agile, Scrum

Aspect	Prototype	Agile	Scrum
Flexibility	Less flexible in change	Highly adaptable to change	Flexible to change
Process	Linear, step by step	Iterative and incremental	Iterative and incremental with specific roles
Output	Program prototype and initial features	Ready-to-use program iterations	Sprints with program enhancements
Communication	Development team with stakeholders	Intensive communication between team and stakeholders	Open and transparent communication between team and stakeholders
Stakeholder	Dependent on prototype interpretation	Prioritizing based on business value	Clear roles and responsibilities for each team member and stakeholder
Development Speed	Tends to be slower	Faster and adaptive	Fast and structured with scheduled sprints
Uncertainty and risk	Reducing uncertainty and risk in early development stages	Addressing uncertainty with quick responses and feedback	Reducing uncertainty and risk through iteration and adaptation

Based on Table 1, those studies indicate that software development methods have their own strengths and weaknesses. The Prototype approach provides stakeholders with clear visualizations but consumes significant time and proves difficult to customize post-approval. On the other hand, Agile and Scrum methodologies prioritize customization and swift processes, though necessitating teams to swiftly adapt and grasp all facets. This contrasts

with microservices, which call for specialized teams to manage each service separately. Further research can be conducted by combining the differences and requirements of existing methods, tailored to specific conditions and needs. This approach can result in an effective and efficient method for the given circumstances.

The author hypothesizes that the adoption and execution of the devised novel method named Banyan Agile methodology may lead to a noticeable improvement in the overall performance of employees within the organizational framework.

3. BANYAN AGILE

The need for introducing a new model stem from the evolving landscape of software development, marked by the simultaneous coexistence of monolithic and microservices architectures. Existing Software Development Life Cycle (SDLC) models often grapple with the challenges posed by these diverse architectural paradigms. While traditional models have proven effective, the complexities arising from the integration of monoliths and microservices within the same development lifecycle necessitate a more adaptive and comprehensive approach.

The author intends to devise a novel method named Banyan Agile. This method, formulated by the author, aims to address control management issues and supporting monolith and microservices architectures.

3.1. Key Principles of Banyan Agile

Banyan Agile is formulated with key roles, including the Product Owner, Banyan Leader, Team Leader, Quality Assurance, and Developer. The Product Owner is responsible for defining features or changes to be implemented. The Banyan Leader ensures that tasks progress according to the set targets. The Team Leader is obligated to ensure and support the developer team members in completing assigned tasks, ensuring they align with the established targets.

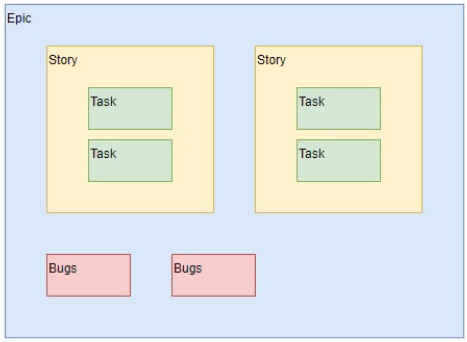

Figure 1: Banyan Agile Task Illustration Organization

As seen in Figure 1, Banyan Agile also formulates that tasks will be categorized into four types: Epic, Story, Task, and Bug. Epic provides a broad overview of a target. Story represents a feature that needs to be developed or modified to fulfill the Epic target. Task is a process that developers need to undertake to fulfill the Story. Bug refers to issues that arise while maintaining the running software.

The tasks assigned to developers will be tailored to their respective expertise. If collaboration with other teams is required, separate tasks will be created to serve as connectors between tasks. It is expected that with this method, developers with more specialized and expert skills will be cultivated.

The process of implementing Banyan Agile is as follows:

1. The Product Owner engages in discussions with Quality Assurance (QA) to formulate the necessary changes or features.
2. The Product Owner, Banyan Leader, and relevant Team Leader hold discussions, if needed, regarding the changes or features to be worked on.
3. Each task (Epic, Story, Task, and Bug) will have an Assignee and Quality Assurance. The Assignee is responsible for ensuring that the task is completed according to the target, while Quality Assurance ensures that the development results align with the target.
4. Quality Assurance assigns a weight to each Story as a variable indicating the complexity of the Story.
5. Every Story and Task is assigned priority levels (High, Medium, Low) to determine prioritized tasks. Bugs are given a high priority (High).

6. Stories, Tasks, and Bugs have logging time to track the time spent by developers.
7. The Banyan Board does not have a fixed set of steps.
8. Weekly meetings are conducted to review the sprint and plan for a new sprint.
9. A sprint can last 1-2 weeks, adapted to the conditions.

3.1.1. Flexibility for both monoliths and microservices

With the principles outlined above, the organization aims to establish management control by leveraging the capabilities of each team leader. To address communication challenges in microservices, teams are encouraged to communicate internally, aligning with their respective responsibilities. Additionally, a dedicated team is assigned to handle monolith projects, facilitating seamless communication within the team.

The incorporation of Quality Assurance (QA) in each task is expected to ensure that the development outcomes adhere to standards that meet the requirements defined in the initial planning stages. This strategic approach not only enhances management control through team leaders but also fosters effective communication within and between teams, addressing both microservices and monolith development needs.

4. STAGES OF BANYAN AGILE

In developing software, Banyan Agile follows stages as shown in Figure 2.

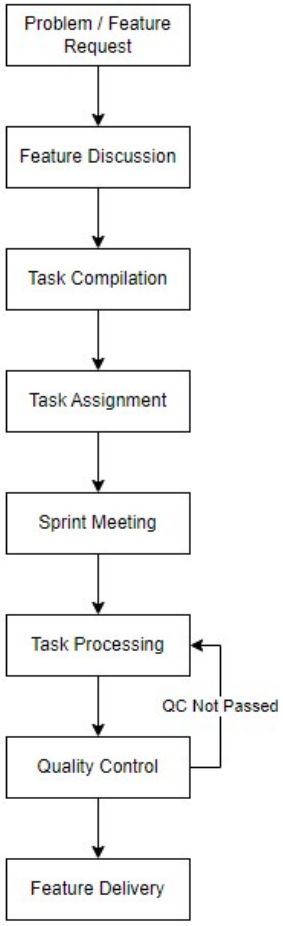

Figure 2: Stages of Banyan Agile

4.1. Problem or Feature Request

The Product Owner extends their role by welcoming change requests from users across various units within the organization, fostering collaboration and addressing needs that may arise from different segments of the user community.

4.2. Feature Discussion

The Product Owner collaborates with users to thoroughly understand the nuances of the requested features and scope. Following this, they coordinate with the Banyan Leader, Team Leader, and QA to ensure that the proposed request aligns with the overall development strategy. After reaching a consensus, the request is formally scheduled for implementation, marking a seamless integration of user input into the development workflow.

4.3. Task Compilation

The Team Leader meticulously outlines the specific tasks required to meet the criteria of the user's request. They consider the skills and expertise of the team members, ensuring an efficient allocation of responsibilities. This detailed task planning lays the foundation for a systematic and organized approach to implementing the requested features.

4.4. Task Assignment

The Team Leader further delegates the tasks to the developers, distributing responsibilities based on each team member's skills and expertise. These tasks are then scheduled during sprint meetings, which occur either once or twice a week. This regular cadence ensures a consistent and transparent workflow, allowing for ongoing progress tracking and adjustments as needed.

4.5. Sprint Meeting

The Team Leader conducts meetings to discuss the upcoming tasks as per the schedule and addresses any issues encountered during the development of features. This proactive approach fosters effective communication within the team, allowing for collaborative problem-solving and ensuring a smooth development process.

4.6. Task Processing

The developers proceed to complete the tasks assigned to them, leveraging their skills and expertise to deliver the required features or changes. This phase marks the hands-on implementation of the planned development tasks by the team.

4.7. Quality Control

The Quality Assurance (QA) team engages in testing to assess whether the features meet the specified criteria. Through meticulous testing processes, QA ensures that the implemented features align with the defined standards and fulfill the user requirements.

4.8. Feature Delivery

Tasks that meet the criteria are then updated, and User Acceptance Testing (UAT) is conducted to validate the functionality with the users. This step involves seeking user feedback and ensuring that the implemented features align seamlessly with user expectations and requirements.

5. RESULTS AND IMPLEMENTATION

The Banyan Agile methodology has undergone testing and integration at PT ASD. PT. ASD, established in 2012, is a company operating in the healthcare sector. With its headquarters located in the city of Medan, Indonesia. PT. ASD has expanded its presence with several hospital branches in cities such as Jakarta, Bandung, and Bali. To facilitate the development and maintenance of the software utilized in hospitals, PT. ASD maintains an internal IT development team dedicated to supporting operational needs. The IT development team is subdivided into software development, software implementation, and hardware teams.

PT. ASD has previously utilized SDLC methodologies such as Prototype, Waterfall, and Scrum. During the implementation phase, PT. ASD had a team of 22 employees dedicated to software development.

5.1. Employee Demographics

The total number of employees in the software development team at PT. ASD is 22, with the following details:

1. The demographic details of employees based on age are provided in Table 2.

Table 2: Employee Demographics by Age

Age	Count
20-25 years	9
26-30 years	7
> 30 years	6

2. The demographic details of employees based on gender are provided in Table 3.

Table 3: Employee Demographics by Gender

Gender	Count
Man	14
Woman	8

3. The demographic details of employees based on education and majors are provided in Table 4.

Table 4: Employee Demographics by Education and Majors

Education and Majors	Count
Bachelor's degree in information technology	7
Bachelor's degree in information systems	14
Master's degree in computer science	1

4. The demographic details of employees based on the duration of work experience at PT. ASD are provided in Table 5.

Table 5: Employee Demographics by Duration of work experience at PT. ASD

Duration of work experience	Count
1-3 years	9
3-5 years	5
>5 years	8

5.2. Challenges in Implementation

From the observation and document analysis, issues identified in the implementation of Banyan Agile at PT. ASD include:

1. There is documentation of change requests that have been left unprocessed for an extended period due to various reasons, such as low priority, insufficiently detailed requirements from the user, and the user's lack of experience as an information source.
2. Task documentation remains incomplete in some cases due to urgent requests and communication through chat media, leading to oversight in documentation.
3. Tasks have been divided with a scope of 35 hours per week within a 40-hour work week, yet there are still instances where tasks exceed the allocated time due to additional time spent on activities such as meetings, discussions, and underestimation of the expected time spent.
4. Many additional tasks have been identified that are not documented, causing disruptions to the overall planning.
5. There are tasks that have been technically completed but have not been tested by QA because the user is still seeking QA support for software implementation.

6. COMPARISON

6.1. Comparison before and after implementation of Banyan Agile

We collected the pooling data from employees of PT. ASD before and after the implementation of Banyan Agile, generating noticeable values as depicted in Table 6.

Table 6: Employee Pooling before and after implementation of Banyan Agile (0 means worse, 10 means best)

No.	Subject	Average before	Average after	Changes
1	Communication between teams	5.27	7.32	+2.05
2	Task conflict	5.18	7.23	+2.05
3	Collaboration	4.95	6.68	+1.73
4	Productivity	5.36	7.09	+1.73
5	Workload	5.27	6.73	+1.46
6	Bugs that arise	5.4	6.32	+0.92
7	Overtime	5.05	5.36	+0.31

The pooling results indicate improvement across all measured dimensions. Communication between teams experienced a rising average score of 5.27 points to 7.32 points ($\Delta = +2.05$). Similarly, task conflict showed increase ($\Delta = +2.05$), with average scores climbing from 5.18 points to 7.23 points. Collaboration and productivity also increase ($\Delta = +1.73$) each (Collaboration: 4.95 points to 6.68 points, Productivity: 5.36 points to 7.09 points). Workload perception among team members decreased, showing an improvement of ($\Delta = +1.46$) from 5.27 points to 6.73 points). The occurrence of bugs decreased marginally, with a change of ($\Delta = +0.92$) from 5.4 points to 6.32 points. Overtime hours demonstrated a modest decreased, with increase ($\Delta = +0.31$) from 5.05 points to 5.36 points. When correlated with observational findings and document reviews, further enhancement is achievable by addressing issues that occurred during implementation.

6.2. Feedback from Employees

Based on the interview results with the employee in the software development team at PT. ASD after implementing Banyan Agile, key feedback points obtained include:

1. The need for improved documentation for each task.
2. A stronger commitment to implementing Banyan Agile.
3. Continuous training to better understand the concepts of Banyan Agile.
4. Selecting tools that support the implementation of Banyan Agile.

6.3. Comparison with existing methodology

Table 7: Comparison Banyan Agile, Prototype, Agile, Scrum

Aspect	Banyan Agile	Prototype	Agile	Scrum
Flexibility	Flexible to change	Less flexible in change	Highly adaptable to change	Flexible to change
Process	Iteratif and incremental with sepecific roles	Linear, step by step	Iterative and incremental	Iterative and incremental with specific roles
Output	Sprints with program enhancements	Program prototype and initial features	Ready-to-use program iterations	Sprints with program enhancements
Communication	Communication by Product Owner and stakeholders	Development team with stakeholders	Intensive communication between team and stakeholders	Open and transparent communication between team and stakeholders
Stakeholder	Clear roles and responsibilities for each team member and stakeholder	Dependent on prototype interpretation	Prioritizing based on business value	Clear roles and responsibilities for each team member and stakeholder
Development Speed	Fast and structured with schedule sprints	Tends to be slower	Faster and adaptive	Fast and structured with schedule sprints
Uncertainty and risk	Reducing uncertainty and risk through Quality Assurance, iteration and adaptation	Reducing uncertainty and risk in early development stages	Addressing uncertainty with quick responses and feedback	Reducing uncertainty and risk through iteration and adaptation
Team Structured	Team divide based on specialty and/or organization structured	Not specifically specified	Not specifically specified and requires role rotation over time	Not specifically specified

Based on Table 7, The Banyan Agile framework is tailored to suit organizations in their software development endeavors, whether employing monolithic or microservices architectures. It must exhibit flexibility and agility to effectively respond to the organization's evolving dynamics and the adoption of new technologies, all while preserving the organizational structure and ensuring the high quality of the software output.

7. CONCLUSION

7.1. Summary of Findings

This research delves into the introduction of Banyan Agile and its practical implementation on a PT. ASD. The study aimed to assess the effectiveness and adaptability of Banyan Agile approach in enhancing project management processes in software development using monolith and microservice architectures. The findings reveal some positive impact on improving collaboration among cross functional teams and productivity on overall project achievement.

Some challenges encountered during the implementation phase were also identified, shedding light on areas that required consideration to further improve the project outcomes like improving documentation, implementation

collaboration and training and tools in improving Banyan Agile.

7.2. Implications for The Industry

The integration of Banyan Agile implies that embracing this methodology holds the potential to enhance collaboration and boost productivity within the industry. This suggests that by incorporating Banyan Agile practices, the industry stands to reap significant benefits in terms of improved teamwork and increased efficiency, thereby contributing to overall growth and success.

7.3. Recommendation

As this study contributes to the discourse on the introduction and implementation of a Banyan Agile, several recommendations for future exploration and development emerge.

1. Deepening Understanding through Longitudinal Studies

Conduct longitudinal studies to capture the long-term impact and sustainability of Banyan Agile. This approach would provide insights into how Banyan Agile practices evolve over time and their enduring effects on organizational dynamics.

2. Exploring Industry-Specific Applications

Tailor Banyan Agile methodologies to specific industries or sectors. Future studies could delve into customizing Banyan Agile practices to address the unique challenges and requirements of diverse industries, fostering a more nuanced understanding of its applicability.

3. Cultural and Organizational Adoption Studies

Investigate cultural and organizational factors that influence the successful implementation of methodologies. Future research could delve deeper into the cultural nuances and organizational structures that facilitate or hinder the adoption of Banyan Agile practices, providing actionable insights for diverse contexts.

REFERENCES:

[1] Dragoni, N., Giallorenzo, S., Lluch-Lafuente, A., Mazzara, M., Lafuente, A. L., Montesi, F., Mustafin, R., & Safina, L. (2017). Microservices: yesterday, today, and tomorrow Data Mining and Machine Learning for Knowledge Discovery and Predictive Analysis View project Combining Two Modelling Approaches: GQM and KAOS in an Open Source Project View project Microservices: yesterday, today, and tomorrow. https://www.researchgate.net/publication/315664446_Microservices_yesterday_today_and_tomorrow

[2] Velepucha, V., & Flores, P. (2023). A Survey on Microservices Architecture: Principles, Patterns and Migration Challenges. IEEE Access, 11, 88339–88358. https://doi.org/10.1109/ACCESS.2023.3305687

[3] Diansyah, A. F., Rahman, M. R., Handayani, R., Nur Cahyo, D. D., & Utami, E. (2023). Comparative Analysis of Software Development Lifecycle Methods in Software Development: A Systematic Literature Review. International Journal of Advances in Data and Information Systems, 4(2), 97–106. https://doi.org/10.25008/IJADIS.V4I2.1295

[4] Hurst, J. (2014). Comparing Software Development Life Cycles. SANNS Software Security

[5] Taya, S., & Gupta, S. (2011). Comparative Analysis of Software Development Life Cycle Models.IJCST, 2(4), Oct.-Dec.

[6] Hossain, M. I.. (2023). Software Development Life Cycle (SDLC) Methodologies for Information Systems Project Management. International Journal For Multidisciplinary Research, 5(5). https://doi.org/10.36948/IJFMR.2023.V05I05.6223

[7] Maulida, N, "Studi Literatur Penerapan Metode Prototype dan Waterfall dalam Pembuatan Sebuah Aplikasi atau Website", https://www.researchgate.net/publication/359814579, April 2022.

[8] Bhatnagar, V, "A Comparative Study of Software Development Life Cycle Models", International Journal of Application or Innovation in Engineering & Management, Vol. 4, Issue 10, 2015, pp. 23-29.

[9] Principles behind the Agile Manifesto. 2001. https://agilemanifesto.org/principles.html

[10] Megargel, A., Shankararaman, V., & Walker, D. K. (2020). Migrating from Monoliths to Cloud-Based Microservices: A Banking Industry Example. 85–108. https://doi.org/10.1007/978-3-030-33624-0_4

[11] Beerbaum, D. (2023, March). Agile Strategy - Achieving Sustainable Advantage. https://www.researchgate.net/publication/369440853

[12] Kadenic, M. D., Koumaditis, K., & Junker-Jensen, L. (2023). Mastering scrum with a focus on team maturity and key components of scrum. Information and Software Technology, 153. https://doi.org/10.1016/J.INFSOF.2022.107079

[13] Verwijs, C., & Russo, D. (2022). A Theory of Scrum Team Effectiveness. ACM Transactions on Software Engineering and Methodology. https://doi.org/10.1145/3571849

[14] Ozkan, N., & Tarhan, A. K. (2020). Evaluation of Scrum-based agile scaling models for causes of scalability challenges. ENASE 2020 - Proceedings of the 15th International Conference on Evaluation of Novel Approaches to Software Engineering, 365–373. https://doi.org/10.5220/0009390403650373

[15] Akhtar, A., Bakhtawar, B., & Akhtar, S. (2022, November). Extreme Programming VS Scrum: A Comparison of Agile Models. https://www.researchgate.net/publication/365118765

[16] Blinowski, G., Ojdowska, A., & Przybylek, A. (2022). Monolithic vs. Microservice Architecture: A Performance and Scalability Evaluation. IEEE Access, 10, 20357–20374. https://doi.org/10.1109/ACCESS.2022.3152803

[17] Blinowski, G., Ojdowska, A., & Przybylek, A. (2022). Monolithic vs. Microservice Architecture: A Performance and Scalability Evaluation. IEEE Access, 10, 20357–20374. https://doi.org/10.1109/ACCESS.2022.3152803

[18] Abgaz, Y., McCarren, A., Elger, P., Solan, D., Lapuz, N., Bivol, M., Jackson, G., Yilmaz, M., Buckley, J., & Clarke, P. (2023). Decomposition of Monolith Applications Into Microservices Architectures: A Systematic Review. IEEE Transactions on Software Engineering, 1–32. https://doi.org/10.1109/TSE.2023.3287297