JETIR.ORG ISSN: 2349-5162 | ESTD Year : 2014 | Monthly Issue
JOURNAL OF EMERGING TECHNOLOGIES AND INNOVATIVE RESEARCH (JETIR)
An International Scholarly Open Access, Peer-reviewed, Refereed Journal

Supporting Large Scale Agile Development With Domain Driven Design

[1]Dhruv Seth, [2]Swamy Athamakuri, [3]Aneeshkumar Sundareswaran

[1]Solution Architect, [2]Senior Engineering Manager, [3]Solution Architect
[1]Walmart Global Tech, Sunnyvale, California, USA
[2]Walmart Global Tech, Sunnyvale, California, USA
[3]Walmart Global Tech, Sunnyvale, California, USA

Abstract : This paper discusses the role of DDD principles in the context of Agile methodologies for software development in complex projects. Agile methodology here prioritizes iterative development, flexibility, and customer collaboration, in contrast to the DDD problem-domain centric approach where technical solutions are aligned to business objectives. One of the benefits of Agile DDD combo is improved management of complexity, better communication and teamwork and meeting domain requirements more precisely. Organizations like Netflix and Spotify have real-world applications of DDD in Agile environments to provide imaginative software solutions that are scalable, reliable and maintain efficient performance. Nevertheless, the complexity management, the communication issues and technical debt have to be solved in order to define the project success. Software development teams, key takeaways are including building a common understanding of the domain, simplifying the domain modeling, and communication between domain experts and developers. Moving on, the future use of DDD and Agile methodologies looks promising with new tools, frameworks, and approaches being developed for large-scale Agile projects. Companies should harness the synergy between DDD and Agile frameworks in order to explore the new opportunities for innovation and successful development of their software.

IndexTerms **- Domain-Driven Design (DDD), Agile Methodologies, Large-Scale Software Development, Collaboration, Complexity Management, Innovation.**

I. INTRODUCTION

1.1 Context and Background

The ever-expanding field of software development agility has become a key pillar for those teams that are looking to produce quality products as fast and effectively as possible. The center of such movement is based on the Agile methodology, which is the set of principles and techniques devised to advance collaborative and responsive environment during the development course [1].

A. Agile Development Methodology

The Agile methodology, as outlined in the Agile Manifesto in 2001, revolves around iterative development and incremental approach. It puts the customer collaboration above the rigid planning and documentation, responding swiftly to the changes [2]. It emphasizes the notion of close work by the cross-functional teams that deliver working software in the form of the short, repeating development cycles known as sprints. Scrum, Kanban, and Extreme Programming (XP), are practices which have reformed the software development process itself, helping the team focus on difficulties of quickly adaptive of need for the market and requirements changes [1]. In the 15th Annual State of Agile Report by VersionOne indicates that 97% of the organizations surveyed are practicing Agile in one form or another [3]. The survey underlines the general view that Agile helps the teams to be more productive, to increase project visibility, and to bring a higher satisfaction to the clients.

B. Domain-Driven Design (DDD)

Although Agile methodologies are very good at solving interpersonal problems and creating iterative approach, they might experience difficulty with handling complexity of large projects, worst with intricate business logic and multiple business partners. Exactly in this situation DDD comes as an auxiliary one. DDD, pioneered by Eric Evans in his seminal book "Domain-Driven Design: "Tackling the Complexity," Author of the book argues for the design of complex software systems consistent with the real-world domain that they behold in their hearts [4]. Essentially, DDD suggests that the software designs are based on the domain concept of domain and its content of concept, relationship, and behavior. DDD has been holding on apparent favor as a helpful gateway for progressing compilation of the complexity of software development particularly in fields like e-commerce, healthcare, and finance. The business world understands that good software may arise only if they ensure the presence of masterminds, and there is a smooth communication between developers and domain specialists.

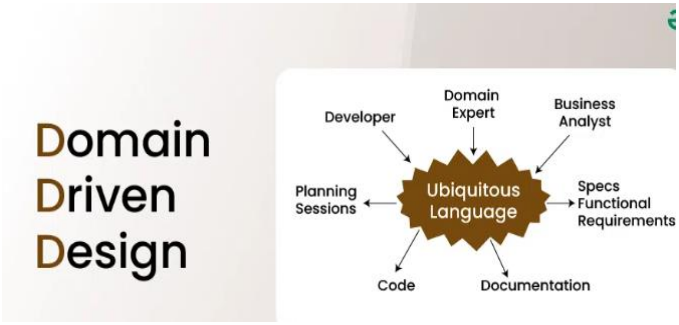

Figure. 1. Domain-Driven Design Interaction in the Context of Unbiquotous Language. The purpose for the application of Domain-Driven Design (DDD) software development methodology is to acquire and construct the problem area of domain where the system functions. This aspect illustrates the fact that one has to combine with those experienced in the field in order to gain a firm grasp of the particular and complex details of the field. Developers are more capable of not only understanding but also expressing in the software design of conceptual entities in the domain by using the DDD's principles, patterns, and methods which are available to them.

C. Importance of Effective Methodologies in Large-Scale Projects

Massive deployment of software development involves additional intricacy, like the gathering of multiple resources, the number of people involved and project scaling. In these kinds of situations, selecting the right methodology is essential to the accomplishment of the project. Besides helping collaboration and communication, successful methodologies facilitate providing management complexities, and aligning technical solutions into business objectives. According to a McKinsey & Company study, Agile has become a mission-critical approach for organizations that aspire to keep improving and retain their position in the current hypercompetitive and fast-passing market [5]. Though expanding Agile beyond teams of specialists is a major issue, it is provoked by issues such as maintaining uniformity, managing the dependencies between teams, and preserving among all the Agile teams. This paper highlights how DDD can be a useful tool for scaling Agile development and manage complexity through a structured approach domain-driven design synchronizes the technical solutions with domain requirements and directs a collaborative environment between cross-functional teams [5]. Through case studies, examples, and best practices, this article highlights how the Agile and DDD synergize and show how the software projects at scale can be powered by these methodologies and how the organizations can successfully use both of the Agile and DDD methodologies to deliver their projects.

I. UNDERSTANDING AGILE DEVELOPMENT

A. Core principles of Agile methodology

The Agile Methodology is based on a set of core principles that are all oriented towards meeting customer needs through iterative development and continuous feedback loops. Such a set of principles, discussed in Agile Manifesto, mean that people and interactions are more valued than processes and tools, that a working software is worth more than any bulky documentation, that client collaboration is better than any contract negotiation, and that the way to change is preferable to any rigid plan. Adaptability, flexibility, and a customer-focused approach are the key themes of the Agile Manifesto underlying software development [6]. Agile teams are shaped by continuous change and feedback during development which initiates utilizing of opportunities to adapt to changing customer needs and market forces as soon as possible. The recent wave of Agile software development is already much of focus on practices such as DevOps and CI/CD [7]. Team's ability to automate processes, build up continuous delivery pipelines, and hasten time-to-market is undoubtedly possible by availing these. As you can read in the State of DevOps Report in 2021 of Puppet, organizations that successfully manage DevOps practices deliver software faster and with higher productivity. Besides, employee satisfaction is higher because of such practices [8].

B. Benefits of Agile for Software Development Teams

By implementing Agile methodologies, the developers and their team are experiencing a lot of advantages which we are going to highlight here. Agile practice facilitates teams to fix the complex projects on lots of smaller understandable iterations that will later be integrated to one complete product. So, team will be able to deliver the working software more often, on the later stages shortening the time to market and increase the rate of customers' satisfaction. The Agile technique is a tool of communication and collaboration among team members and has the aim to encourage a culture of transparency, accountability, and continuous improvement [9]. The critical element of Agile teams in achieving this goal is that they constantly maintain a shared consciousness on the project vision and the goals progress through practices such as retrospective sessions, daily stand-up meetings and sprint planning. This makes the teams more flexible in that they can respond quickly to any obstacles they may face. Apart from that, the Agile techniques emphasize the customer-value delivery, where the work on the products is used in respond to the business values and user feedback. By ensuring that development efforts are in line with user and market objectives, this customer-centric strategy lowers the danger of creating features that are superfluous or irrelevant [9]. The Digital.ai research showed that companies using Agile approaches experience higher project success rates, higher productivity of the team, and the quality of their software products improvement compared to the traditional methodology. Consequently, this is evidence of the gradually growing perception of Agile as being a game-changer in relation to the software development field.

C. Challenges of Scaling Agile for Large Projects

Agile approaches are great for small to medium-sized teams, but there are special difficulties when trying to scale them for larger projects. Coordination overhead, communication breakdowns, and alignment problems worsen as projects get bigger and more complicated. Agile transformations of the scale presupposes shift in structures, processes and culture so that the teams and departments can work effectively and be aligned. Disparity in the standard practices and non-consistent implementation among

various teams might bring about problems like complex interfaces and failures in related tasks. As a result, expanding Agile encounters challenges of dealing with rising number of stakeholders, prioritizing between stakeholders, and preserving a shared vision of the organization. In the absence of appropriate governance mechanisms and feedback from leaders of change, the transformation to Agile may run the risk of facing resistance and inertia from processes that have been already consolidated and structures that are reputedly siloed.

D. Need for Complementary Methodologies to Address Scaling Challenges

This challenge of scaling Agile for large organizations is being answered with more new methods and frameworks which complement the main method. The combined use of these parallel approaches brings new direction, structure and consequently provides additional tools for the operation of cross-team coordination, integration, and alignment across multiple Agile teams. For example, there are Agile frameworks like Scaled Agile Framework (SAFe), Large-Scale Scrum (LeSS) and Disciplined Agile Delivery (DAD) whose flexibility allows them to be adopted to the needs of huge organizations [10]. These structures offer pointers on how to expand the use of Agile methods, synch up the dependencies, and make sure that strategies are in agreement with performance. Ultimately supplementing Lean philosophy, Systems Thinking, and Design Thinking by offering principles and tools for dealing with complexity, streamlining processes, and driving innovation. Through the integration of complementary approaches into their Agile transformations, organizations can improve their large-scale project agility, resilience, and flexibility. Agile approaches are very beneficial for software development teams; nevertheless, in order to scale Agile for large projects, careful planning, coordinating, and integrating is needed. In order to successfully implement Agile transformations at scale and manage scaling problems, complementary approaches and frameworks are essential.

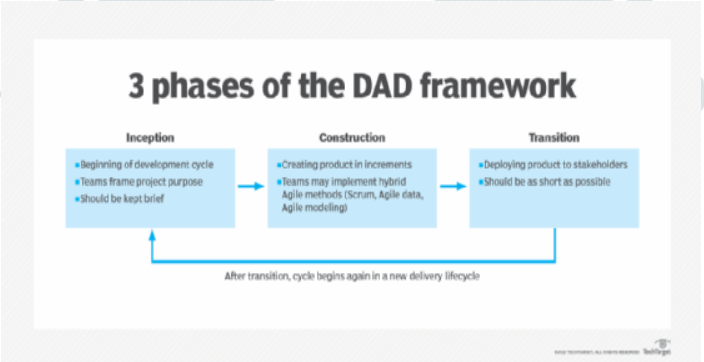

Figure. 2. Phases of Disciplined Agile Delivery (DAD). The three main phases of the DAD framework that mark stages in product development are inception, construction, and transition.

Figure.3. Scrum Structure

From figure 3, coordination of several Scrum teams working on the same project is done using the Scrums approach. It is usually used in scenarios when there are multiple Agile teams, each of which adheres to Scrum procedures on its own.

II. INTRODUCTION TO DOMAIN-DRIVEN DESIGN

Domain-Driven Design, is an approach to software development that has the execution of the understanding and modeling of the specific problem domain as the key objective of the development. Coined by Eric Evans in his influential book "Domain-Driven Design: While the title may seem to pose great complexity itself, DDD is actually a set of design principles, patterns, and practices that help developers working on building software systems to achieve a closer match to the real world domain that the software operates in. At the core of DDD is a principle of the domain which is the subject area or a problem space that the application system deals with. The primary purpose of DDD is to determine the direction for modeling the domain in a manner that captures the core concepts, relationships, and behaviors and enforces the creation of software solutions that are congruent to the business requirements.

A. Core Principles of DDD and Its Focus on Domain Modeling

The main ideas from the DDD point that domain modeling, the process of creating the shared understanding of the domain among the stakeholders and the transformation of that understanding into a conceptual model that is further used for the software system design and implementation, are at the heart of this approach.

Key concepts in DDD include:

1. Ubiquitous Language: DDD promotes the use of a common language by domain experts and developers. This widely used language ensures that all parties involved in the domain discussion are speaking the same language and helps close the communication gap between technical and non-technical stakeholders.

2. Bounded Contexts: DDD notices that there might exist different implementations model for the same domain in the system. Bounded contexts determine the context of operation of models, which one can use to manage complexity and resolving of conflicts which are caused by different models' interaction.
3. Aggregates and Entities: DDD is all about entity and collective use and defines the business logic, mostly using aggregates that represent critical information domain. Aggregates serve as the means for defining consistency boundaries in such a way that these boundaries settle only after all changes that were done as part of the transaction are consistent [11]. On the other hand, entities represent the objects that have their own identities and lifecycles.

B. Benefits of DDD for Software Development, Especially in Complex Domains.

DDD offers several benefits within the complex domains characterized by intricate business rules, diverse stakeholders, and evolving requirements:

1. Increased Alignment with Business Objectives: Through the manner of representing the domain precisely and model it, DDD ensures that the solutions of software systems are quite close to the intended business objectives and will meet the needs of the users. Accuracy and agreement would mean that software development would no longer be at risk of creating products that solve perceived problems but not the ones that matter.
2. Improved Communication and Collaboration: In particular, DDD includes collaboration between the field experts and software developers conjointly developing the problem area knowledge unleashing better understanding of the problem area and communication competency. The collaboration, in this case, would help the custom software development company make sure that their products provide firstly the needs of business stakeholders and, secondly, that of the end-users.
3. Reduced Complexity and Technical Debt: Domain modeling is achieved by designing DDD in a way that takes away the presentation layer and keeps the business logic within aggregates and entities. This structure levels down complexities and decreases the build-up of technical debt [11]. A well-formed domain model is a basis which one can apply the principles of software development, thus facilitating maintenance and improvement of the system. Agile principles emphasize on the active customer involvement, good communication and team work. DDD accent on the being model, its massive design and specification as well as ecosystem.

C. How DDD Complements Agile Principles

Domain-Driven Design is considered the extension of Agile principles as it offers the structured method to tackle complexity, makes the designed solutions to be aligned with the domain needs and a communication channel for various operational teams. On the other hand, Agile methodologies highlight an important aspect: delivering working software iteratively and responding to change [11]. DDD considers the problem domain in the first place and making a complete model of it so it can be understood. Teams can thus leverage the most relevant features of the DDD approach and embed them into iterative Agile development cycles to make sure that their implementations have not only a high technical quality but also fit perfectly with business goals and users [11]. The ubiquitous language, context directionalities, and the domain-driven design patterns are the tools that Agile teams use to develop software which explains the domain exceptionally well and gives the relevant stakeholders the maximum value from it. The Domain-Driven Design brings principles, patterns, and practices in the space of software systems' domains for building good models that correspond to complex problems. It is through domain modeling, communication, and collaboration, which gives Agile the necessary adornment that ensures that the teams finish with a successful software solution even in the most difficult cases.

Figure. 4. Various importance Importance of Agile scaling.

III. WAYS OF SUPPORTING LARGE-SCALE AGILE DEVELOPMENT WITH DOMAIN-DRIVEN DESIGN

Business success in delivering large-scale Agile software using Domain-Driven Design (DDD) is only possible through the application of sound principles and practices, recognizing the fact that both collaboration and alignment with strategic objectives are two major factors in successful development of any software. The process begins with a careful drawing of domain boundaries, which establishes the foundation for an organized method. Clarity in duties is ensured and subsequent architectural decisions are set in motion by identifying the system's primary domains. Methods such as Event Storming and Domain Mapping are helpful in illustrating these boundaries and deciphering complex domain interconnections, which helps teams develop a comprehensive understanding of each other [13]. Business success in delivering large-scale Agile software using Domain-Driven Design (DDD) is only possible through the application of sound principles and practices, recognizing the fact that both collaboration and alignment with strategic objectives are two major factors in successful development of any software. It will be considered first to describe the framework of the scope. This will create a general outline for the entire course. Recognizing what domain the system is constructed on and juxtaposing it with other domain function is a great way to start the architectural decision and improve the intersections of areas which have interrelated functions. These techniques especially Event Storming and Domain Mapping show up-front where we need to build solid boundaries, and how the system contributes there in together with hidden partners in the areas determining the dominant domain, therefore putting wholesome information to teams.

As an important measure, the architecture modularization has become a dominant element in large-scale Agile projects with DDD as its leading force [14]. Unbundling the system into the loosely connected modules or microservices simplifies the development efforts as everyone can independently work on their discipline-specific modules. These bounded contexts glue these modular structures together, providing system modules with distinct boundaries and independence that are essential for successful integration of diverse teams. Accordingly, teams can fast tracks within their areas, and as a result adjusting the organizations' ability to respond, even to the constantly changing requirements. CI/CD are vital components without which the development process cannot be converted fully into an automated and Agile system. A CI/CD pipeline works like a channel for effortless and continuous software deployment. A team can make software deployments faster and involve in a process while the software is functioning. Incorporating a quality culture of a continuous improvement where issues are audited and addressed promptly, would enhance the existing culture of hardy and responsive. Collaborative modeling is seen as a crucial piece in the puzzle which is the gap-bridging between the business domain experts and the developing teams, increasing the level of the problem domain understanding [12]. Techniques such as Event Storming, Example Mapping and Domain Workshops enable cross-functional collaboration, where stake holders can work together to create a domain model which does not only embody their insights and perspectives.

Through an iterative process of collaborative modeling, the domain model is kept up to date with changing business demands and accurately captures the nuances of the issue domain. One of the main advantages of the evolutionary approach in architecture is that it emphasizes the need for flexibility and adaptability in order to overcome change. Through the adoption of a model that lets teams revise the design incrementally, responding to the changes of the business environment and technology advancements, changes in the architecture can be made. The mindset of continuous evolution empowers the team to experiment, to look for new ways and to improve the architecture from time to time to be able to cope with new eligible threats. Agile cross-functional teams are key elements of Agile projects at a large scale, ensuring compatibility in the composition and approaches involving different skill sets and views [15]. Through building up teams with domain knowledge, technical proficiency, and testing competence, organizations will inculcate cultures of ownership, accountability, and empowerment. Decentralizing the decision-making power to this cross-functional team allow them to decide with data and autonomy, it opens up the possibility to innovate based on their deep understanding of the problem domain and thus able to obtain the greatest benefit. The core aspect in the efficient adoption of large-scale Agile development with DDD is building a domain-specific communication language between persons with direct domain knowledge and programmers. Making everyone speak the same language ensures that the documents, codes, and conversations is clear and consistent throughout the project, helping effective communication and collaboration between the team members. With shared language, the teams will be able to create a unified understanding of domain concepts that will be used to ensure the alignment and coherence of the teams by preventing misunderstanding and ambiguity. In the domain of testing, Domain-Driven Testing is a guiding principle steering testing strategies towards domain concepts and business objectives. Through Behavior-Driven Development (BDD) and Acceptance Test-Driven Development (ATDD), together with example situation modeling, teams can build tests that are based on what the system from a domain perspective is expected to do [16]. This process of system testing alone from the holistic approach alleviates the worry of the user about the system's functionality and resilience, and allows the team to deliver quality software at scale. It is crucial to note that the foundation of large-scale Agile development using DDD is investing in strong tools and infrastructure, which makes team integration and cooperation easy. Tools for domain modeling, version control, automated testing frameworks, and collaboration platforms operate as enablers to improve transparency, visibility, and traceability in development processes. Organizations may fully utilize DDD concepts within an Agile framework, fostering innovation and providing value to stakeholders, by utilizing the appropriate tools and technology.

IV. CASE STUDIES AND EXAMPLES

A. Case Study 1: Successful Implementation of DDD in a Large-Scale Agile Project

Large-scale Agile initiatives have been implemented recently by financial institutions, and Domain-Driven Design (DDD) is essential to the project's success. The goal of these projects is to update the antiquated financial systems of these businesses in order to enhance customer satisfaction and simplify internal processes. A team of developers' uses bounded contexts to clearly define boundaries between various system components and identify important business concepts like accounts, transactions, and customer profiles. Iterative development cycles supported by Agile methodologies empowers the team to gradually implement domains design models such as aggregates, entities and value object. This has made it possible to build the system in phases, involving the stakeholders in the process and improving the system based on the feedback and early bugs. Financial services organizations successfully implemented DDD in conjunction with Agile, resulting in the delivery of a modern banking system that satisfied the requirements of both internal and external users. The enhanced functionality, expandability, and performance of the system raised client satisfaction and enhanced operational effectiveness.

B. Case Study 2: Challenges Faced and Lessons Learned in Adopting DDD Alongside Agile

A software development company should be excited to implement Domain-Driven Design (DDD) concepts if it is starting an Agile transformation with the aim of providing a new e-commerce platform, however, there may be obstacles. Matching the development team's in-depth subject knowledge with that of domain experts—who lack technical expertise—represents one of the primary hurdles. Delays in the development process might arise from miscommunication and divergent viewpoints on domain concepts. The correct balance between applying DDD patterns and producing functional software iteratively can also be difficult for teams to achieve.

C. Real-World Examples of Organizations Leveraging DDD Principles in Agile Environments

Companies have used various approaches based on the Domain-Driven Design (DDD) concepts to effectively do projects in an Agile environment. Netflix, the major global content streaming service, run their business on DDD because of the highly-complicated nature of the network. Through DDD, Netflix bridges the gap of content delivery between users globally, and this achieves an optimum of streaming experience. The organization uses DDD to visualize different aspects of its content dispatch network, such as the content caching, server routing, and bandwidth optimization [17]. With its domain split into small pieces and creating clear boundaries between components, Netflix helps in smooth content delivery to the users despite their geographic location widely.

In addition, Spotify, a leading music streaming site from millions of users worldwide, also embodies DDD especially in terms of storing their abundant songs and playlist compilations. Spotify blends of domain-driven design with Agile uncovers the key to

bringing an excellent user experience and keeps the scalability and reliability at the same time. The DDD approach is used by Spotify to model its large music catalog, including artist profiles, album metadata, as well as user-generated playlists among many others [18]. The system has clear domain boundaries and within aggregates and entities is logically bound of the business logic, so the consistency and integrity in Spotify music database are guaranteed. It is with the application of the DDD principles that Netflix and Spotify achieve the objective of scalability, reliability, and speed in their platforms. With the DDD, there is more development optimization in the backend infrastructure for the scalability and performance needed to ensure smooth music playback and reduced latency for the end-users worldwide. Thus, with DDD Spotify can craft an architecture that may resolve the situation of millions of users that connect at one time, even with the requirements of high availability and reliability. Also Netflix by using DDD which helps suggests content based on the user history and preference. With DDD as its starting point, Netflix can create personalized content recommendations which will be appealing to individual users so that they can increase the engagement and keep the users longer.

V. CHALLENGES IN LARGE-SCALE AGILE DEVELOPMENT

Large-scale Agile development projects pose complex issues that the companies should tackle in order to enhance the success of the project. One of the many complexities that arise is the issue of complexity management, especially in larger projects with complex business logic, but a high number of requirements. Complexity management involves proper scope and source design, as well as collaboration between multi-faceted teams' members to guarantee that the system is continuously coordinated and manageable [19]. Also, coordination and communication issues among multiple Agile teams are problematically frequent among the complications that attend large-scale Agile development. With the evolution of projects, it tends to become more complicated and important; hence the communication and alignment among teams to ensure effectiveness becomes a serious concern [20]. The communication breakdown, conflicting objectives and lack of cooperation might result in the prolonged timelines, the duplication of activities, and finally in the subparagraph answers. Keeping homogeneity in domain between different teams and systems is essential for managing consistency and composure in sizeable scale Agile projects. It becomes a necessity to maintain a unified domain understanding among team members since they all work as a collective striving to build a solution to the same problem. Additionally, the need for consistency in domain models and interfaces becomes evident. Loss of domain independence induces mismatching if there are interconnections between different parts of the system and the whole system becomes less reliable. Apart from these, large-scale Agile projects must tackle the issue of removing tech debt and scalability. Technical debt, this accumulated shortcut or compromise or maintenance deferment would eventually grow bigger and cause system's agility and maintainability issues later on. Furthermore, the issue of scalability needs to be addressed as the projects grow and have to accept more and more user, more data and more changes in services. Among other things, Agile teams need to make time-to-time addressing of technical debt and extending of solutions that are scalable in order to guarantee the long-term viability and stability of the project.

VI. CONCLUSION

The inclusion of Domain-Driven Design (DDD) with Agile methodologies in large software projects brings in a wide array of benefits and advantage. Incorporating DDD into Agile is the way to cope with complexity. The team is able to ensure alignment of technical solutions with appropriate domain needs and to form much more effective cross-functional cooperation among the team members. DDD, which is based on the academic thoughts formulated by Eric Evans, Martin Fowler, and other renowned domain modeling pioneers, offers a systematic method for keeping the work of the software system following the objectives of business and the demands of users for maximum project productivity. Two main messages should be acknowledged here by software development teams that contemplate DDD application in Agile environment: the necessity for describing and adhering to the domain included among the stakeholders, simplicity and pragmatism as priorities in domain modeling, and collaboration between domain experts and the development team. Through DDD applied with Agile approaches, the capability of the team to produce software which is adjusted constantly to the customers and it could be extended and perform well can be achieved. Speaking of the future, it is reasonable to think that the combination of DDD and Agile approaches in software development will be very successful. As and as enterprises discover new ways to implement Agile transformations as well as search for the best practices in software development, DDD and Agile synergy is expected to grow in the coming years. One of the possibilities lying in the future of these practices may be the launching of new tools, frameworks, and best works of art that could be used in large-scale Agile initiatives. While the application of DDD and Agile approaches calls for organizations to be the pioneers in its exploration, they must discover the advantages that come from the congruence of these methods. Investing in the training and mentoring of DDD approach assistance as well as creation of programs that support organizational procedures and processes can help organizations to unblock the obstacles to meet the enterprise goals and gain new opportunities for innovation, collaboration, and success in the software development process.

VII. REFERENCE

Öznur Uludağ, M. Hauder, M. Kleehaus, C. Schimpfle, and F. Matthes, "Supporting Large-Scale Agile Development with Domain-Driven Design," pp. 232–247, May 2018, doi: https://doi.org/10.1007/978-3-319-91602-6_16.

J. Highsmith, "History: The Agile Manifesto," Agilemanifesto.org, 2019. http://Agilemanifesto.org/history.html

W. A. Cram, "Agile Development in Practice: Lessons from the Trenches," Information Systems Management, vol. 36, no. 1, pp. 2–14, Jan. 2019, doi: https://doi.org/10.1080/10580530.2018.1553645.

E. Evans, "Domain-Driven Design Tackling Complexity in the Heart of Software," 2003. Accessed: Apr. 21, 2024. [Online]. Available: https://fabiofumarola.github.io/nosql/readingMaterial/Evans03.pdf

W. Aghina, C. Handscomb, O. Salo, and S. Thaker, "The impact of agility: How to shape your organization to compete | McKinsey," www.mckinsey.com, May 25, 2021. https://www.mckinsey.com/capabilities/people-and-organizational-performance/our-insights/the-impact-of-agility-how-to-shape-your-organization-to-compete

K. Brush and V. Silverthorne, "What is Agile Software Development (Agile Methodologies)?," TechTarget, Nov. 2022. https://www.techtarget.com/searchsoftwarequality/definition/Agile-software-development

"AGILE, DEVOPS AND BEYOND: Challenges in Software Development," www.linkedin.com, 2023. https://www.linkedin.com/pulse/Agile-devops-beyond-challenges-software-development-nioyatech-inc-z1whf?trk=article-ssr-frontend-pulse_more-articles_related-content-card#:~:text=Embracing%20Agile%20and%20DevOps (accessed Apr. 21, 2024).

Puppet, "2023 State of DevOps Report: Success | Puppet by Perforce," Puppet.com, 2023. https://www.puppet.com/success/resources/state-of-devops-report (accessed Apr. 21, 2024).

T. Tran, "Benefits of Agile Methodology in Software Development," www.orientsoftware.com, 2022. https://www.orientsoftware.com/blog/benefits-of-Agile-methodology/

Project Management Institute, "Disciplined Agile Delivery | Disciplined Agile," www.pmi.org, 2024. https://www.pmi.org/disciplined-Agile/process/introduction-to-dad

S. Paradkar, "Navigating Software Complexity: Patterns, Principles, and Practices in Domain Driven Design, Agile…," Oolooroo, Jan. 17, 2024. https://medium.com/oolooroo/domain-driven-design-Agile-and-software-architecture-partners-in-design-a2e89057f64 (accessed Apr. 21, 2024).

K. McDonald, "Collaborative Modeling," Inside Product, Apr. 19, 2017. https://insideproduct.co/collaborative-modeling/ (accessed Apr. 21, 2024).

"Event Storming and Context Mapping: Powerful Tools for Software Development," www.linkedin.com. https://www.linkedin.com/pulse/event-storming-context-mapping-powerful-tools-alok-kulkarni (accessed Apr. 21, 2024).

J. Flasks, R. Harding, and M. Hewitt, "Embrace Modular Technology and Agile Process to Deliver Business Impact," EQengineered, Jun. 07, 2021. https://www.eqengineered.com/insights/embrace-modular-technology-and-Agile-process-to-deliver-business-impact

A. Olawale, "Agile Software Development Handbook – Scrum, Kanban, and Other Methodologies Explained," freeCodeCamp.org, Aug. 30, 2023. https://www.freecodecamp.org/news/Agile-software-development-handbook/#:~:text=Agile%20emphasizes%20the%20importance%20of%20cross%2Dfunctional%20teams%2C%20where%20members (accessed Apr. 21, 2024).

P. Thrimavithana, "A guide to Test-Driven Development, Acceptance Test-Driven Development and Behavior-Driven…," Medium, Oct. 02, 2023. https://medium.com/@yasarathrima/a-guide-to-test-driven-development-acceptance-test-driven-development-and-behavior-driven-15187b7097e2#:~:text=ATDD%2C%20also%20known%20as%20Behavior (accessed Apr. 21, 2024).

"Netflix: What Happens When You Press Play? - High Scalability -," High Scalability, Dec. 11, 2017. https://highscalability.com/netflix-what-happens-when-you-press-play/

N. Torabi, "The Inner Workings of Spotify's AI-Powered Music Recommendations: How Spotify Shapes Your Playlist," Medium, Aug. 28, 2023. https://neemz.medium.com/the-inner-workings-of-spotifys-ai-powered-music-recommendations-how-spotify-shapes-your-playlist-a10a9148ee8d

L. Bolzan de Rezende, J. Denicol, P. Blackwell, and H. Kimura, "The main project complexity factors and their interdependencies in defence projects," Project Leadership and Society, vol. 3, no. 1, p. 100050, Dec. 2022, doi: https://doi.org/10.1016/j.plas.2022.100050.

H. Saeeda, M. O. Ahmad, and T. Gustavsson, "Challenges in Large-Scale Agile Software Development Projects," Proceedings of the 38th ACM/SIGAPP Symposium on Applied Computing, Mar. 2023, doi: https://doi.org/10.1145/3555776.3577662.