AI & SOCIETY
https://doi.org/10.1007/s00146-021-01240-x

ORIGINAL ARTICLE

An architecture governance approach for Agile development by tailoring the Spotify model

Abdallah Salameh[1] · **Julian M. Bass**[1]

Received: 18 April 2021 / Accepted: 1 June 2021

Abstract
The role of software architecture in large-scale Agile development is important because several teams need to work together to release a single software product while helping to maximise teams' autonomy. Governing and aligning Agile architecture across autonomous squads (i.e., teams), when using the Spotify model, is a challenge because the Spotify model lacks practices for addressing Agile architecture governance. To explore *how software architecture can be governed and aligned by scaling the Spotify model*, we conducted a longitudinal embedded case study in a multinational FinTech organisation. Then, we developed and evaluated an approach for architectural governance by conducting an embedded case study. The collected data was analysed using Thematic Analysis and informed by selected Grounded Theory techniques such as memoing, open coding, constant comparison, and sorting. Our approach for architectural governance comprises an organisational structure change and an architecture change management process. The benefits reported by the practitioners include devolving architectural decision-making to the operational level (i.e., Architecture Owners), enhancing architectural knowledge sharing among squads, minimising wasted effort in architectural refactoring, and other benefits. The practitioners in our case study realised an improved squad autonomy by the ability to govern and align architectural decisions. We provide two key contributions in this paper. First, we present the characteristics of our proposed architectural governance approach, its evaluation, benefits, and challenges. Second, we present how the novel Heterogeneous Tailoring model was enhanced to accommodate our architectural governance approach.

Keywords Large-scale Agile developments · Agile architecture · Spotify Tailoring · FinTech · Thematic analysis · Grounded Theory

1 Introduction

The Spotify model, which was introduced by Kniberg and Ivarsson (Kniberg 2014; Kniberg and Ivarsson 2012), has become influential among Agile proponents and hence formed the basis of Agile methods used in several other organisations (Salameh and Bass 2018, 2019b, 2020). In the study of knowledge sharing using the Spotify model, previous research has identified patterns for knowledge sharing by cultivating Spotify Guilds (Šmite et al. 2019) and presented the importance of cultivating participation culture in general and establishing communities of practices (Šmite et al. 2020). Also, previous research on the Spotify model has revealed a new approach to Agile tailoring, called Heterogeneous Tailoring (Salameh and Bass 2020). Two key features characterise this approach. First, each squad (i.e., team) is empowered to select and tailor its development method. This key feature is supported by a set of identified guidelines that facilitate building squads' autonomy (Salameh and Bass 2020). Second, each squad is aligned with other squads and common product development goals. This key feature is supported by a set of identified influential factors on aligning autonomous squads (Salameh and Bass 2018, 2019b). Each identified factor is supported by a set of practices and processes that facilitate aligning autonomous squads. However, our empirical research identified a challenge in governing and aligning Agile architecture across the autonomous squads of the Spotify model.

✉ Abdallah Salameh
a.salameh@edu.salford.ac.uk

Julian M. Bass
j.bass@salford.ac.uk

[1] University of Salford, 43 Crescent, Salford M5 4WT, UK

Published online: 24 June 2021

Our research addresses the question: *How can software architecture be governed and aligned by scaling the Spotify model?* We conducted a longitudinal embedded case study in a multinational FinTech organisation to gain an understanding of how the Spotify model is being used. In this longitudinal embedded case study, we conducted a direct observation of Agile practices over 21 months and 14 semi-structured open-ended interviews. Our analysis identified a challenge in governing and aligning Agile architecture across autonomous squads. Therefore, we decided to intervene by developing and evaluating an approach for architectural governance. We conducted an intervention embedded case study, which lasted 3 months, during which 32 ceremonies were observed and 8 semi-structured open-ended interviews were conducted. The collected data was analysed using Thematic Analysis (Braun and Clarke 2006) and informed by selected Grounded Theory techniques (Glaser 1998).

Our approach for architectural governance considers aligning autonomous teams vertically on the product level and horizontally on the individuals' skillsets, which was not the case in the previous work (Bellomo et al. 2014; Martini and Bosch 2016; Nord et al. 2014). Our approach incorporates a structural change and an architecture change management process. According to the practitioners of our case study, the proposed approach for architectural governance has shifted the boundaries and facilitated the alignment and governance of Agile architecting by tailoring the Spotify model. This transformation has, in turn, improved squads' autonomy by aligning architectural decisions across autonomous squads.

The key contributions of this paper are: (1) developing and evaluating an approach for architectural governance when using the Spotify model; and (2) adapting the Heterogeneous Tailoring approach to include aligning and governing architectural decision across autonomous squads by introducing our approach for architectural governance to the Spotify model. In this paper, we present the characteristics, benefits, and challenges of our proposed approach for architectural governance. Also, we present the impact of the evaluated approach for architectural governance on the Heterogeneous Tailoring approach.

The remainder of this article is outlined as follows. Background and related work are presented in Sect. 2. In Sect. 3, we describe the employed research methodology. The findings are presented in Sect. 4 and concluded with a discussion in Sect. 5, which discusses our contribution, limitations and threats to validity, and related work. In Sect. 6, we conclude the article.

2 Background

2.1 Agile architecture

The role of software architecture in Agile software development has been controversial in previous research (Abrahamsson et al. 2010; Kilu et al. 2019). Many advocates give software architecture a vital role in Agile development as in other development approaches, other opponents against this. This clash between the two different cultures is because of different believes on each side. For example, some advocates argue about the importance of scaling any development approach that does not pay sufficient attention to software architecture in a large-scale (Abrahamsson et al. 2010; Kilu et al. 2019). One of the principles behind the Agile Manifesto is "*continuous attention to technical excellence and good design enhances agility*". Hence, Agile software development should pay attention to software architecture.

Agile architecting is an iterative and incremental way of architecture evolution, which is recognised by previous research as an approach that replaces Big-Design-Up-Front and keeps a project synchronised with the latest changing conditions (Booch 2009; Dingsøyr et al. 2010). The notion of "Agile architecture" evokes two concepts (Bellomo et al. 2014): (1) software architecture that is easy to evolve and modify and (2) an Agile way that defines an architecture using an iterative lifecycle. While the first is resilient enough not to degrade after a few changes, the second does evolve over time, as the problem and the constraints are better understood. These two concepts are not the same; non-Agile software development process can lead to a flexible and adaptable architecture. Also, an Agile process can lead to rigid software architecture. However, what teams build is influenced and constrained by how they build it. Also, how teams build a product is influenced and constrained by the followed design and architecture (Buschmann and Henney 2013). Thus, Agile practitioners should focus on architectural issues that block teams' agility to achieve technical excellence, good design, and improve the agility of software development (Buschmann and Henney 2013). Neglecting specific architectural considerations even early in the development process can make architectural refactoring costly (Erdogmus 2009).

Agile Architecture can be understood as patterns and tactics that enable a simultaneous focus on architecture and Agile development (Bellomo et al. 2014). Bellomo et al. (2014) identified Agile architecture patterns and tactics that influence the time and cost to implement, test, and deploy requested changes. These patterns and tactics include layer architecture, separate interfaces, restrict dependencies and separate concerns. Also, they identified

some patterns and tactics that improve product scalability, which includes clustered architecture with load balancing and replicated copies, encapsulation of algorithms, and data caching. Moreover, Bellomo et al. identified some patterns and tactics that focus on flexibility in deployment and controlling the cost and time for testing. These patterns and tactics include virtualisation (layering both the infrastructure and the application), standardised and configurable architecture (parameterisation and static and dynamic binding), and executable (interface-driven code structure).

Nord et al. (2014) provide an example of using architectural tactics and aligning architecture, Agile development teams, and production infrastructure. They explored architectural tactics that support scaled Agile development and improve the alignment of the architecture and the development of the organisation. The proposed alignment by Nord et al. includes (1) a vertical and horizontal decomposition of the software architecture to enable alignment of the teams accordingly, (2) a matrix augmented-role team structures by using Scrum, and (3) a catalogue of tactics mapped to Agile development. This catalogue of tactics can be collected from successful organisations and literature.

Yang et al. (2016) identified several architecting activities and Agile architecting practices. They found that the architecting process, which covers the entire architectural lifecycle, is comprised of 11 architecting activities. These architecting activities received varying degrees of attention in Agile software development. These activities are Architectural Description, Architectural Evaluation, Architectural Understanding, Architectural Maintenance and Evolution, Architectural Analysis, Architectural Refactoring, Architectural Impact Analysis, Architectural Implementation, Architectural Synthesis, Architectural Reuse, and Architectural Recovery. Most of the previous research effort has been put on Architectural Description based on the desires of Agile practitioners (Yang et al. 2016). However, architectural understanding, analysis, and refactoring were identified as the most beneficial activities that can be used with Agile development (Yang et al. 2016). Besides, Yang et al. identified 41 Agile practices used with architecture. However, only a few of these practices have been widely employed in practice and discussed in the literature—such as Backlog, Sprint, Iterative and Incremental Development, Just Enough Architectural Work, and Continuous Integration. There is a lack of guidance on how and when to use such Agile architecting practices (Yang et al. 2016). Therefore, Agile practitioners are using such practices based on their experience and knowledge.

Martini and Bosch (2016) identified some architect roles and architecture practices, which are then used to develop and evaluate a framework for Agile architecting in large-scale organisations with embedded software projects. The identified Architect roles include Chief Architect, Governance Architect, and Team Architect. By analysing the relationships among the architects, Martini and Bosch found that most practices need the roles to coordinate and cooperate and thus mitigate the challenges. Besides the architect roles, Martini and Bosch identified different sorts of teams: Feature Team, Runway Team, Architecture Team, and Governance Team. Feature Teams are steered by Product Managers and consist of cross-functional teams. Each Feature Team has a Team Architect who is responsible for the reference architecture and leads architecture activities. Runway Teams are dedicated teams for one or more sprints to focus on the "architecture feature" rather than on customer-related features to overcome architectural debt, which might lead to crisis. Architecture Team, which comprises the Architect roles above, work together to coordinate and collaborate since no single architect can have all the information needed to support numerous teams within large-scale organisations. Governance Team involves Governance Architects and Product Owners with the responsibility of strategically making a risk assessment of architecture changes and prioritising the backlogs of the teams between features and architecture improvements to balance the short-term with the long-term objectives.

2.2 Autonomous teams

Large-scale projects are challenging because several teams need to work together to release a single software product (Conboy and Carroll 2019; Moe et al. 2016). Some examples of the identified challenges in large-scale Agile development are building and maintaining teams' autonomy (Conboy and Carroll 2019; Stray et al. 2018) and aligning self-organising teams (Moe et al. 2016; Stray et al. 2018).

The "Autonomous Teams" concept has different origins and definitions in the literature (Stray et al. 2018). This concept was studied and described from various perspectives in the past; socio-technical, organisational theory, and complex adaptive systems. Stray et al. found that the closest definition of autonomous teams as applied from outside software engineering into Agile development comes from the knowledge-management perspective. Stray et al. state that the first introduction of autonomous teams into software engineering was made by way of the Agile manifesto, which cited self-organising teams as the source of *"the best architectures, requirements, and designs*". Other researchers define autonomous teams in terms of informal self-organising roles (Hoda et al. 2013). Self-organisation is recognised as one of the Agile principles since the introduction of Agile Manifesto. Agile teams are self-organising teams that can manage the workload and embrace team-based decision-making while having mutual trust and respect (Cockburn and Highsmith 2001). Autonomy has a direct influence on

team effectiveness since the authority of decision-making is moved into the operational level (Moe and Dingsøyr 2008). This decentralised decision-making speeds up the development process and increases the accuracy of problem-solving. Such teams have a sense of accountability for the committed work (Cockburn and Highsmith 2001; Hoda et al. 2013). Also, they work toward a compatible goal while merging and resolving conflicting priorities (Cockburn and Highsmith 2001; Salameh and Bass 2018). However, self-organising autonomous teams are not uncontrolled (leaderless) teams (Cockburn and Highsmith 2001). The leadership in self-organising teams is considered light and adaptive. Leaders are responsible for building strategy, setting directions, aligning people, motivating teams, and providing feedback (Anderson and McMillan 2003). These leaders have different job titles such as Scrum Master in Scrum, Coach in XP, and Squad Leader or Agile Coach in Spotify.

Previous research (Conboy and Carroll 2019; Moe et al. 2008; Salameh and Bass 2018) has identified barriers to autonomous teams in large-scale Agile. For example, there is a lack of guidelines for how teams should be organised, which is the case in Scrum (Moe et al. 2008). Also, high individual autonomy can increase the individuals' preference for their own goals over the team's goals (Moe et al. 2008). Moreover, aligning autonomous teams is challenging because of the varying degrees in stakeholders' expectations for the alignment of teams (Conboy and Carroll 2019). Besides, there is a tension between teams' autonomy and alignment (Salameh and Bass 2018, 2019b). Too much alignment might hinder the autonomy of the teams, but without alignment, the teams are autonomous but are ineffective.

2.3 Tailoring the Spotify model

Spotify is a music streaming service, which had a total of 248 million monthly active users worldwide in October 2019. The Spotify organisation has benefited from substantial growth in the last decade because of its innovation (Kniberg and Ivarsson 2012; Salameh and Bass 2020). The Spotify organisation has developed its own Agile culture and tailored Agile practices to fit a very-large-scale software programme (more than 300 people) distributed across four cities (Kniberg and Ivarsson 2012; Salameh and Bass 2018).

The Spotify model is driven by creating autonomous, yet aligned squads (Kniberg 2014; Kniberg and Ivarsson 2012). To initiate the creation of autonomous yet aligned squads, Spotify employs an adaptive structure and creates communities around this structure (Kniberg 2014). This structure is based on a matrix of two dimensions (vertical and horizontal), which inspires innovation (Kniberg 2014). The communities of Squads and Tribes represent the vertical structure. The communities of Chapters and Guilds represent the horizontal structure.

A Tribe consists of a collection of co-located squads and is designed to be of less than 100 people (Kniberg 2014; Kniberg and Ivarsson 2012). A Tribe aims to promote collaboration and to mitigate dependencies between squads. Within each Tribe, there are small groups of people sharing a similar skillset and working within the same competency area, called Chapters. Members of each Chapter meet regularly to solve problems within their expertise. These Chapters are considered the glue that sticks the whole organisation together without sacrificing too much autonomy (Kniberg 2014; Kniberg and Ivarsson 2012). While Chapters are located within the same Tribe, there are wide-reaching groups of people across the whole organisation, called Guilds, with a desire to share knowledge and practice over the whole organisation (Kniberg and Ivarsson 2012).

Spotify squads are encouraged to use Lean Startup to promote innovation (Kniberg 2014; Linders 2016). However, squads are allowed to tailor their practices while having the support of Agile coaches (Salameh and Bass 2020). The Spotify model has become influential among Agile proponents and hence formed the basis of Agile methods used in several other organisations (Salameh and Bass 2018, 2019b, 2020). Previous research on Spotify Tailoring has identified (1) tailored practices that promote effectiveness in autonomous squads (Salameh and Bass 2019b), (2) influential factors on aligning autonomous squads (Salameh and Bass 2018), (3) influential factors on Spotify Tailoring for B2B product development (Salameh and Bass 2019a), (4) patterns for knowledge sharing by cultivating participation culture and establishing Spotify Guilds as communities of practices (Šmite et al. 2019, 2020), and (5) an approach to Agile tailoring when using the Spotify model (i.e., Heterogeneous Tailoring) (Salameh and Bass 2020).

Figure 1 illustrates our perception of the Heterogeneous Tailoring approach (Salameh and Bass 2020), which is characterised by two key features. First, each squad is empowered to select and tailor its development method, which is depicted on the bottom side of Fig. 1. For example, one squad uses Scrum whereas another uses a different method such as Kanban or Lean. Thus, each squad has the power to select and tailor its Agile development practices based on its mission.

Second, each squad is aligned with other squads and common product development goals, which is depicted in the upper side of Fig. 1. Previous research identified factors that influence the alignment of autonomous squads (Salameh and Bass 2018, 2019b). These influential factors are (1) adaptive structure, (2) collective code ownership, (3) collective decision-making, (4) knowledge sharing, (5) inter-team coordination, (6) mission-based planning, and (7) delivery strategy. Each identified factor is supported by a set of practices and attributes that strengthens the alignment of autonomous squads. The identified influential factors and

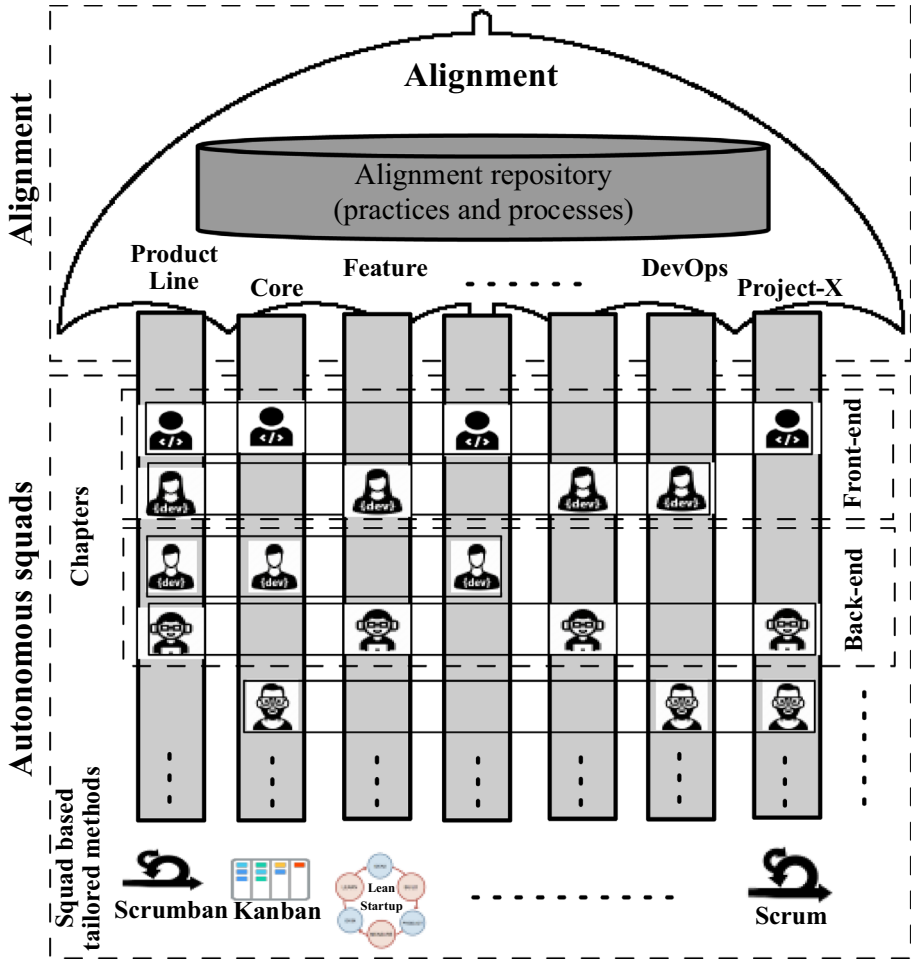

Fig. 1 Heterogeneous Tailoring approach

their related practices can aid Agile practitioners in aligning squads.

2.4 FinTech

FinTech is the abbreviation of "Financial Technology", which is a blend of "financial services" and "information technology". FinTech is defined as the technology used to provide markets with a financial software product or financial software-as-a-service (SaaS), which has sophisticated technology (Gimpel et al. 2018; Knewtson and Rosenbaum 2020). SaaS allows companies to use a cloud-based software application over the internet, instead of buying or building their own product.

FinTech organisations harness benefits such as data security, end-to-end cost savings, scalability and agility. Such organisations are, usually, characterised by having financial services, technological innovation and agility (Knewtson and Rosenbaum 2020). They have different business models, which operate by displacing or complementing current financial services in the industry or by creating new financial services. Also, financial institutions recognize the importance of employing a proven record of technology to embrace technological innovation. Furthermore, FinTech organisations must be Agile to adapt quickly to new market opportunities.

The infrastructure is comprised of financial APIs, which work as an interface between financial services firms and external parties (Gimpel et al. 2018; Knewtson and Rosenbaum 2020). For example, a financial API provides the infrastructure between a bank and an investment application to provide customers with a convenient banking experience.

The FinTech services are managed with national and international regulations related to finance to control data collection, storage and reporting (Knewtson and Rosenbaum 2020). Protocols such as software-as-a-service (i.e., SaaS), Payment Card Industry (i.e., PCI) data security standard, Anti-Money Laundering (i.e., AML) and Know Your Customer (i.e., KYC) are provided for which FinTech organisations should automate regulatory tracking and compliance to.

3 Research methodology

Initially, we conducted a longitudinal embedded case study to have a deep understanding of how the Spotify model is being used in the FinTech industry. The data were collected through observing 225 ceremonies over 21 months, conducting 14 semi-structured open-ended interviews, and accessing different sorts of artefacts. The collected data were analysed using Grounded Theory (Glaser 1998), and thereby the findings were published. In particular, we identified a challenge in governing and aligning Agile architecture across the autonomous squads when using the Spotify model. The Spotify model lacks practices addressing Agile architecture governance. Therefore, we decided to conduct an intervention embedded case study. We developed an approach for architectural governance by tailoring the Spotify model aiming to evaluate it in the same case study organisation.

3.1 Research setting

A multinational FinTech organisation with a large-scale project using the Spotify model was selected for this study. An overview of FinTech and its characteristics is provided in Sect. 2.4. Our case study organisation does process payment transactions in 65 markets around the world, employs approximately 700 staff, and processes around 65 billion EUR per year.

This study focuses on a SaaS project that manages autonomous financial services. These autonomous software services operate under the control of a single administrative project that presents a commonly defined management policy to the service. Thousands of customers (i.e., organisations) utilise this product to manage the payment transactions of their end-users. These payment transactions are managed by the case study project and go through many payment providers around the world. The project allows the customers to configure payment providers easily and quickly since it has a unique rules engine and intelligent routing capabilities that increase the payment acceptance rates. Also, the project helps the customers to detect fraud, analyse all payment transactions, and focus on growing their business.

The software development programme is co-located in the head-quarters—Stockholm—and consists of 37 members. The developers are distributed among six squads (i.e., teams) with around five members in each squad. The developers are also distributed into seven Chapters. In addition, there are one architect, five Product Owners (PO), three Key Account Managers (KAM), two Agile coaches, one support lead, and one test lead.

According to the confidentiality agreement by the organisation, we are not allowed to reveal a detailed description of the explored product and its teams.

3.2 Data collection

After introducing our developed approach to the case study organisation, the case study organisation agreed to try it. The first author, who works in the case study organisation as a senior Software Engineer, conducted a direct observation of the full development lifecycle for 3 months. During the observation period, we observed minor adaptations to our introduced approach, which were introduced by the development programme to respond to the organisation's needs. The first author used the memoing technique to record reflective notes about what the researcher was learning from the observed ceremonies. These recorded notes accumulate as written ideas in a notebook about identified concepts and their relationships. The observed ceremonies, which are 32, include daily stand-ups, backlog grooming, planning sessions, Chapter based meetings, product owners' meetings, and on-demand architectural-based discussions. The employment of direct observation provided the researchers with a deep understanding of the studied phenomenon and mitigated the possibility of deviation between "interviews" view of matters and the "real" case, which is in line with Robinson et al. (2007) findings.

After 3 months, an iterative way of data collection—through semi-structured interviews—and analysis was adopted to perform a constant comparison of data. Performing a constant comparison of collected data facilitated the guidance of future interviews and the analysis, while observations fed the emerging results (Glaser 1998). Since our collected data was analysed continuously, the subsequent interview questions had minor updates to focus on the emerging codes.

The semi-structured open-ended interviews targeted participants from different areas of software development. Thus, practitioners in several different organisational roles were approached. In a result, 8 interviews were conducted, and the participants were one Agile coach, three senior developers, one Product Owner, two Chapter leaders (i.e., Architecture Owners) and one Architect (i.e., Enterprise Architect). An open-ended guide was used to provide the interviewees with the opportunity to raise other issues. The interview guide was revised after the second interview to adapt the questions to focus on emerging results and to choose participants that can provide information on the emerging concerns. The interview guide is published online at (Bass and Salameh 2020). Each interview was recorded (approximately 50 min) and then transcribed verbatim for detailed analysis continuously.

3.3 Data analysis

The collected data was analysed using Thematic Analysis (Braun and Clarke 2006) and informed by some Grounded Theory techniques such as memoing, open coding, constant comparison, and sorting (Glaser 1998). Our analysis was carried out by following the six steps proposed by Braun and Clarke (2006): (1) familiarising with the data, (2) generating initial codes, (3) searching for themes, (4) themes review and refinement, (5) defining and naming themes, and (6) writing the final report. During these steps, we utilised some Grounded Theory techniques such as continuous memoing, open coding, constant comparison, and sorting (Glaser 1998). Utilising these Grounded Theory techniques empowered us with a rigorous method for systematically analysing the collected data.

When an interview was conducted, we transcribed it and started familiarising ourselves with the data by reading it. Then, we employed an *open coding* mechanism to break down the collected data analytically in detail (Glaser 1998). This mechanism begins by collating key points from each interview transcript. Then, a code, which represents a phrase that summaries the key point in 2 or 3 words, was assigned to each key point. A *constant comparison* method, which rigours the generated theory, was employed after conducting each interview (Glaser 1998). This method involves a constant comparison of emerging codes from each interview against other codes from the same interview as well as those from other interviews. Using this constant comparison method facilitated the process of grouping emerged codes into a higher level of abstraction, called themes.

An ongoing process of writing memos was employed throughout the analysis, called *Memoing* (Glaser 1998). The memos represent ideas about emerging codes and their relationships. Memoing is considered a powerful way to pour out the emerging variables (codes, sub-themes, or themes). Also, Memoing facilitates the emergence of relationships (similarities or differences) between different variables. The continuous data collection and analysis reflected on memos' ideas and caused some modification. *Sorting* the theoretical memos was initiated when data collection was almost finished, and coding was almost saturated. Sorting collected memos produced a theoretical outline, which put the scattered data back together (Glaser 1998). As the last step of our analysis, the observation data (i.e., memos) were analysed and compared to the derived themes from the analysed interviews. In result, minor contradictions were identified, which were explored and accommodated in the results.

As a result, two main themes were emerged from our analysis "benefits" and "challenges", which are depicted in Fig. 2. In this figure, the *themes* are marked with bold text and located within the rectangle. Each theme is associated with multiple *categories or concepts*, which are marked with italic text. Each *category* is supported by multiple Codes, which are marked with plain text. The emerged themes did not create a model that consists of higher-order themes relationships between them but instead did present the strengths and weaknesses of our approach by relying on the participants' perspectives. The next section presents these two themes as well as our approach for architectural governance, which comprises an organisational structural change and an architecture change management process, by tailoring the Spotify model.

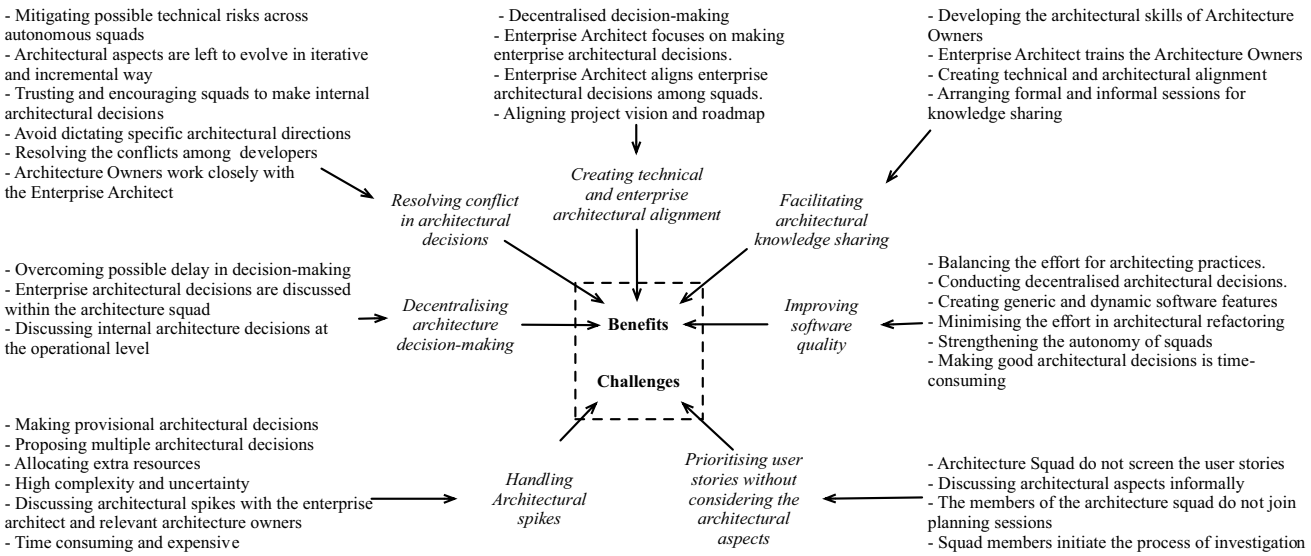

Fig. 2 The emerged themes, categories, and codes

4 Findings

This section presents the findings of our intervention embedded case study. It presents the current state of the organisation before and after conducting the intervention. Thus, this section describes the characteristics of our proposed approach for architectural governance, in its final state. This approach incorporates an organisational structural change and an architecture change management process. Also, this section describes the reported benefits and challenges of the approach. Finally, this section describes how our Heterogeneous Tailoring approach was adapted to accommodate our approach for architectural governance.

4.1 Before conducting the intervention—baseline

The case-study organisation employed Spotify's organisational structure, as illustrated in Fig. 1, while exercising a centralised based architectural decision-making. The squads were empowered to select and tailoring their Agile development practices based on their missions. Those people sharing similar skillset and working within the same competency area were distributed among Chapters while existing in different Squads. Hence, the organisation was utilising a two-dimensional structure. Despite aligning the squads on the product level, the autonomous squads had the freedom to do the required development on different associated parts of the software product because of the realisation of collective code ownership. Consequently, all squads were referring to an Architect because of the complexity of the FinTech project.

The Architect was responsible for designing the software product to address encountered business issues by developing architectural blueprints to produce a cutting-edge software solution with high quality. Practitioners say:

> *"Despite having chapters communities, I was the main reference for all squads when it comes to any architectural based change because of the complexity of the project"*–P2, Architect.
> *"We (developers) were always turning to our architect when it comes to architectural based decisions to figure out the best way to perform an architectural change"*–P4, Senior Developer and Chapter Leader.

However, Chapter communities should be empowered from an architectural perspective to transform decisions into the operational level. In a Product Owner's view, "*there should be someone or a team in charge of the bigger picture… A bit more structure around the ownership based on the missions, verticals, architecture, and by considering a long-term road map*"–P3, Product Owner.

The case study organisation had challenges in aligning architectural decisions across autonomous squads because of lacking a defined process for Agile architecting and utilising a centralised decision-making process. According to the Architect, "*The size of the development programme is now much larger than what it was 3 years ago… I'm overloaded with many responsibilities, which in turn causes a delay in taking architectural decisions and impacts squad's autonomy*"–P2, Architect. From another perspective, a senior developer highlights the importance of having a proper process for architecting to enable squads' autonomy. In making this comment, this developer reports that "*we lacked sufficient Agile architecting process that would improve the autonomy of our squads*" –P5, Senior Developer and Chapter Leader.

4.2 After conducting the intervention—the evaluated approach

To overcome the challenge of aligning and governing architectural-based decisions among autonomous squads, we conducted an intervention embedded case study. In this intervention, an approach for architectural governance, which incorporates a structural change and an architecture change management process, was developed and evaluated. This section describes the characteristics of the evaluated approach for architectural governance using the Spotify model.

4.2.1 Organisational structural change

Our approach includes changes to the organisational structure. These changes aim to facilitate the governance and alignment of architectural decisions across autonomous squads and ultimately to strengthen the squads' autonomy. The structural changes are (1) empowering Chapter Leaders and experienced developers with the role of Architecture Owners, (2) changing the responsibilities of the Architect to be of Enterprise Architectural focus, and (3) locating the Architecture Owners in a virtual squad that is led by an Enterprise Architect. Figure 3 illustrates the introduced structural changes with brown colour.

Architecture Owner role and responsibilities:

The role of Architecture Owner is assigned to Chapter Leaders and other experienced developers. Since Chapters are formed based on competency areas and Squads are aligned on the product-level, we have aligned the Architecture Owners accordingly. Practitioners say:

> *"Giving me the role of Architecture Owner facilitates taking architectural decisions within my Chapter… However, this role increases the overhead on me since*

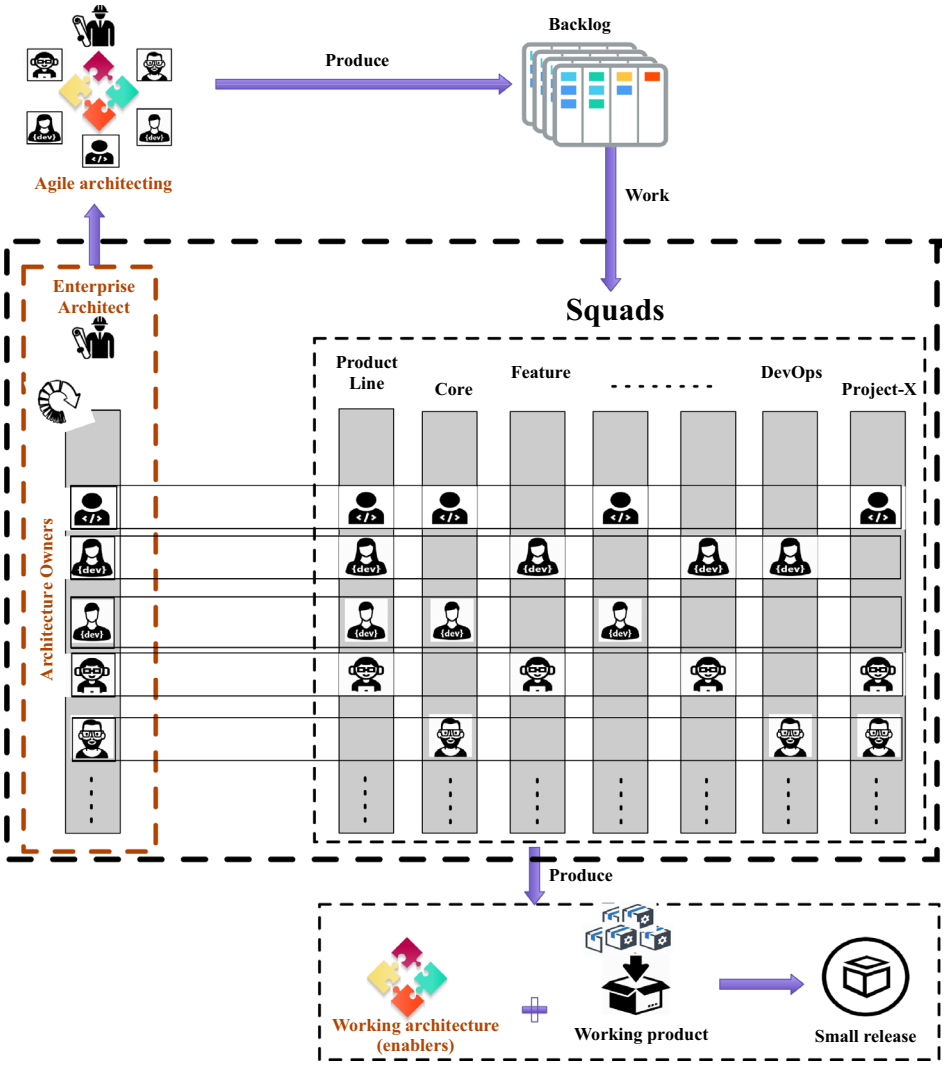

Fig. 3 Organisational structural change—the impact of the introduced intervention on the organisational structure is highlighted with brown colour

I also work a developer"–P4, Senior Developer and Chapter Leader.

"*Breaking down the role of the architect into Architecture Owners roles and distribute it among Chapter Leaders and other developers, based on their competency areas, transform decisions into the operational level, which is beneficial in aligning architectural based decisions*"–P1, Agile Coach.

Most of Architecture Owners time is spent on software development as they work in different squads and the rest of their time is spent on performing architecture activities. Architecture Owners' awareness of the technical and business roadmaps is crucial for aligning architectural decisions within Chapter based communities and hence across autonomous squads. In an Agile Coach's view: "*Architecture Owners should be aware of our business and technical challenges to work on them along the journey*"–P1. Discussing architectural aspects within a Chapter creates an alignment among the squads. For example, a practitioner said: "*discussing architectural decisions within my Chapter facilitates building an agreement among the squads about how to do stuff and when to do it*"–P5, Senior Developer and Chapter Leader.

Architecture Owners are responsible for sharing architectural knowledge among autonomous squads. These Architecture Owners are responsible for creating technical guidelines such as coding, database, and security guidelines. Also, they are responsible for mentoring and coaching the members of their Chapters concerning architectural and design skills. A practitioner said: "*disseminating technical and architectural based knowledge within my Chapter is one of my responsibilities… I create some coding guidelines, review code, and coach my Chapter whenever needed*"–P4, Senior Developer and Chapter Leader.

Resolving conflicts in architectural decisions and mitigating key technical risks across squads are other responsibilities of the Architecture owners. Developers might encounter

conflicts in architectural decisions and not always agree. In making this comment, a practitioner admitted that "*many developers are smart and strong-willed where they do not always come to an agreement… Someone should lead and facilitate the evolution of the architecture*"–P4, Senior Developer and Chapter Leader.

Architecture Owners might encounter challenging architectural tasks that require exploration. Hence, they might either explore such architectural work on their own or could ask a Chapter member, who has encountered this challenge, to explore it. Then, both sides could discuss it further. For instance, practitioners say:

> *"I use an architectural spike to write just enough code to explore the benefits of a specific technology or technique that other members of my Chapter are not familiar with"*–P5, Senior Developer and Chapter Leader.
> *"Sometimes, I do ask other members to explorations on their own… Yet, we do discuss the results within our chapter before taking a final decision"*–P4, Senior Developer and Chapter Leader.

Architecture Owners collaborate with the Enterprise Architect as well as other Architecture Owners within the virtual architecture squads. This collaboration is crucial to get the best out of the Architecture Squad and to utilise better alignment across the organisation. A practitioner said: "*The main reason behind creating a virtual Architectural squad, which consists of Chapters Leaders who have the Architecture Owner roles is to have proper technical and architectural based alignment through the organisation… Meeting whenever needed is important to resolve encountered technical or architectural issues*"–P1, Agile Coach.

Enterprise Architect role and responsibilities:

The role of Enterprise Architect is assigned to the Architect. We have adapted the Architect's responsibilities to be of enterprise-based architectural focus. According to the Agile Coach, "*the Enterprise Architect has great knowledge about the technical and the business roadmaps of our organisation… He should continue focusing on the Enterprise architectural tasks*"–P1, Agile Coach. Since the Enterprise Architect is leading the Architecture squad, the Enterprise Architect should have a strong commitment and provide Architecture Owners with the necessary support. "*It is crucial to have the required commitment and support from our Enterprise Architect in taking enterprise architectural decisions such as integrating two intercorrelated components or even specifying how to expose some APIs*"–P5, Senior Developer and Chapter Leader.

The Enterprise Architect works with the solution management team (i.e., Product Owners and Key Account Managers) and close to the Architecture Owners. This close collaboration aligns architectural decisions over the organisational roadmap and solution intent. A practitioner said: "*the Enterprise Architect spends much time collaborating with senior stakeholders across the organisation to create proper technical and architectural alignment across the squads*"–P3, Product Owner. This collaboration, in turn, facilitates applying proper enterprise architectural decisions at the right time according to the business values.

Creating technical and architectural alignment for the full software product is delegated to the Enterprise Architect. In contrast to the Enterprise Architect, the Architecture Owners are responsible for specific components within the product and concerned about specific technical competency areas. The Enterprise Architect reports, "*I started focusing on architectural based tasks that are related to the full solution to create technical and architectural alignment*"–P2, Enterprise Architect.

The duties of the Enterprise Architect also include promoting enterprise engineering and architectural practices and driving architectural initiatives. Instead of forcing specific architectural decisions, the Enterprise Architect facilitates making enterprise architectural decisions and aligning them across autonomous squads. Thus, the Enterprise Architect facilitates reusing ideas, components, and aligning proven patterns across squads while collaboratively working with squads to develop and evolve the architecture. Practitioners say:

> "*The Enterprise Architect leads our Enterprise Architecture… This role requires proposing architectural initiatives, promoting architectural practices among the squads, and creating technical and architectural alignment on the enterprise level*"–P1, Agile Coach.
> "*Our Enterprise Architect does not force us adopting a specific architectural decision… Instead, he drives architectural initiatives and facilitates them among us (i.e., Architecture Owners)*"–P4, Senior Developer and Chapter Leader.

4.2.2 An architecture change management process

Our introduced architecture change management process was adapted throughout our conducted intervention. This change management process aimed to guide the involved stakeholders (i.e., developers, Architecture Owners, Enterprise Architects, and Product Owners) in governing and aligning architectural-based decisions. This process is comprised of those activities illustrated in Fig. 4. The evaluated architecture change management process activities are as follows:

Activity 1: Discover possible architectural change: When a developer encounters a possible architectural change, the developer will determine the impact of the architectural change. To initiate the Architectural Analysis process, the involved developer should create a Kanban card (i.e., ticket)

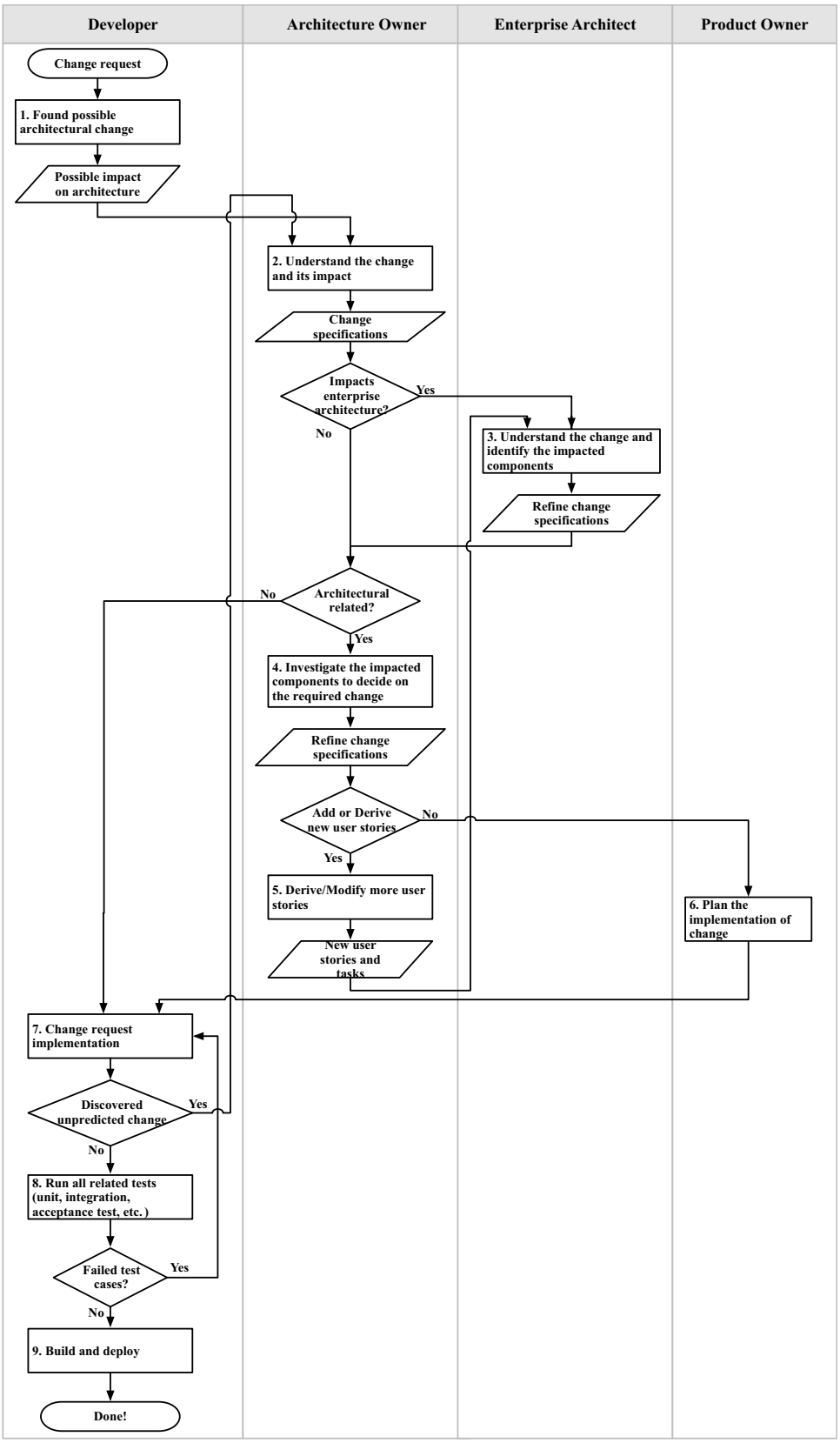

Fig. 4 An architecture change management process

that describes the change request with more technical specifications and visualise it as a WIP in the analysis phase. Such Kanban cards are perceived as enablers for other working items. This activity is followed by Activity 2.

Activity 2: Understand the change and its impact on the software project and its architecture: The architecture owner and the involved developers should understand the nature of the change and determine its potential architectural impact. The Architecture Owner updates the Kanban card with more accurate technical specifications. The outcome of this activity should provide plausible data about the impacted parts of the product and its architecture.

If the identified changes can impact the architecture, one of two possible actions should be performed:

1. If the work requires a change of enterprise architecture, the architecture owner should discuss the necessary change with the enterprise architect and if needed within the architecture squad– should follow Activity 3. Thus, the card is moved in Kanban board as WIP in the enterprise analysis phase.
2. Determining whether the change request could affect the architecture or not.

    (a) If the change request affects the architecture, the architecture owner and the involved developers should investigate the impacted components and decide on the required change, follow Activity 4.
    (b) Otherwise, the task can either be forwarded for planning—follow Activity 6, or forwarded to the related squad for implementation—follow Activity 7).

Activity 3: Understand the change and its impact by the Enterprise Architect: In this activity, the involved stakeholders (i.e., developers, Architecture Owner, and Enterprise Architect) should discuss the received data about the impacted parts of the software product and its architecture. If the architectural change request impacts other parts of the project, those Architecture Owners who work on the respective part are invited to this session. In this activity, the architecture owner presents the provided data in the Kanban card, which is the outcome of Activity 2. Then, an impact analysis is initiated. However, an iterative impact analysis process can be conducted based on the encountered challenges.

The outcome of this activity is an in-depth data about the architecture impact. This outcome includes the identification of the impacted components and the specifications of newly introduced scenarios and requirements. Afterwards, the stakeholders decide whether to proceed to a low-level investigation by following Activity 4 or by proceeding in the change request implementation—when an architectural change is rejected, and a new solution is introduced and replaces the architectural change request—by following Activity 7.

Activity 4: Investigate the impacted components to decide the required change: During this activity, a meeting is carried out between the architecture owner and involved developers to investigate the specifications of the architectural change request deeply. In this meeting, they break down the specifications and requirements into scenarios and then document the details into tasks that can be planned. This activity might identify new impacted components. Hence, the architecture owner might create new user stories for unpredicted changes.

If identifying unpredicted changes was possible, follow Activity 5. Otherwise, the Kanban card, which includes the user story and its tasks, should be moved into To-Do state and forwarded to POs for planning, follow Activity 5.

Activity 5: Derive/modify more user stories: When the architecture owner and the developers identify newly impacted components, the architecture owner will create a new user story for unpredicted changes. These user stories are then moved to WIP in the enterprise analysis phase, which is followed by Activity 3. This way, a new iteration of impact analysis is initiated. In this case, an informal meeting is conducted to discuss such changes.

Activity 6: Plan the implementation of change: If the change request was approved by the architecture squad and the architecture owner, the Kanban card should be available for planning and development. Thus, POs can plan the implementation of this change request and forward the user story and its tasks to the relevant squads for implementation, follow Activity 7.

Activity 7: Implement the change request: In this activity, the implementation of the change is carried out. The squads utilise a hybrid process of Behaviour Driven Development and Test Last Development (Hammond and Umphress 2012). Since the user stories describe the behaviour of the introduced scenarios and requirements, developers are expected to implement the described behaviour in the required test coverage. This implementation might impact negatively other existing test cases. These test cases are subject to modification to accommodate the new requirements. Yet, two cases could encounter the development team while implementing the change request:

1. If the solution is to modify or add new requirements that would require an architectural change, developers should inform the PO and Architecture Owner about the suggested changes to the requirements. In this case, the suggested changes by developers should be declared as unexpected changes. Afterwards, the architecture owner and developers should have an informal meeting to investigate the unexpected changes, follow Activity 2.
2. Otherwise, follow Activity 8.

Activity 8: Run the related tests: The developers should utilise continuous integration to avoid delays caused by integration problems. Subsequently, a continuous testing process should be initiated to obtain immediate feedback on the possibility of violating architectural countermeasures to prevent unreasonable risks associated with a software release. The scope of testing should be extended from test cases to behaviour requirement to validate architectural goals and product behaviour. In case of any violation of requirements after running the tests, developers should check the implementation and failed test cases, follow Activity 7. Otherwise, follow Activity 9.

Activity 9: Build and deploy: This activity should be followed once the continuous testing is completed successfully. A new release can be planned for deployment on production.

4.3 Adapting the Heterogeneous Tailoring approach to govern architectural decisions

This section presents how the Heterogeneous Tailoring approach was adapted to accommodate the developed approach for architectural governance, which consists of structural change and an architecture change management process. Figure 5 illustrates the impact of our approach on the Heterogeneous Tailoring approach by highlighting the introduced changes with brown colour. Consequently, the

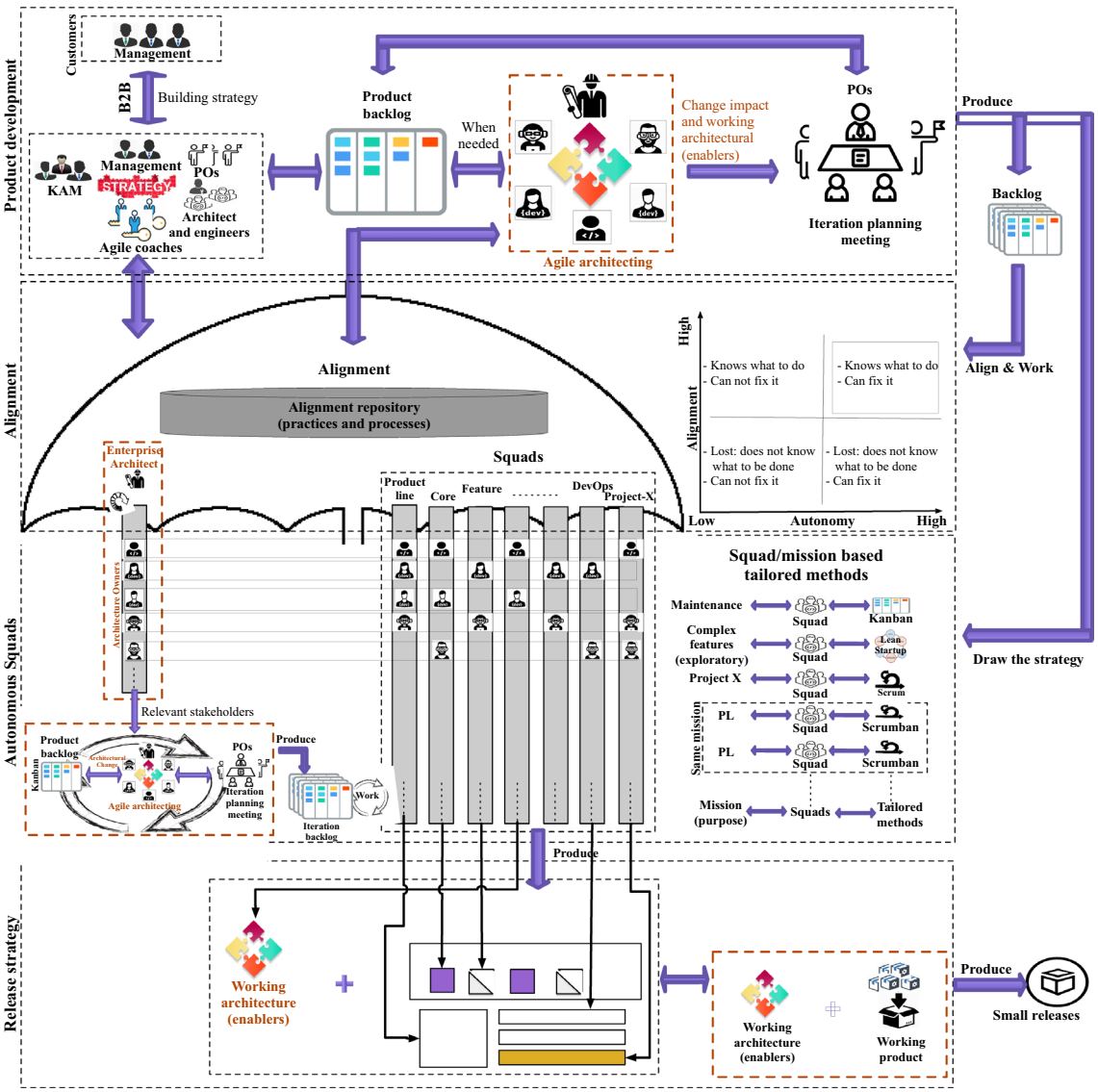

Fig. 5 Adapting the heterogeneous Tailoring approach—the impact of the introduced intervention on the heterogeneous Tailoring approach is highlighted with brown colour

Heterogeneous Tailoring approach was illustrated using four key features: Product development, alignment, autonomous squads, and release strategy.

Product development, which is depicted at the top part of Fig. 5, is impacted by introducing our approach for architectural governance to the case study organisation. This impact is perceived in the outcome of the employed architecture change management process, which produces architectural-based user stories (i.e., enablers) in the backlog. These enablers should be prioritised and planned. A practitioner said: "this change management process results in sometimes new architectural based tickets (i.e., enablers) that need prioritisation and planning"–P3, Product Owner.

The influential factors on aligning autonomous squads are presented in previous research (Salameh and Bass 2018, 2019b). These identified factors and their related practices can guide Agile practitioners in aligning autonomous squads. Our intervention improves the alignment of the autonomous by considering architectural aspects as depicted at the second part of Fig. 5. The alignment and governance of the software architecture is depicted in the introduced change to the organisational structure—through the Architecture Owners and Enterprise Architect roles—as well as the architecture change management process. Our structural change has shifted the boundaries and transformed the architectural-based decisions into the operational level, which is aligned through Chapter communities. A practitioner said: "*discussing architectural-based aspects with our Architecture Owner (Chapter Leader) speeds up the process and puts all members of our Chapter on the same page*"–P7, Senior Developer. The introduced change management process has aligned the Agile architecting process among all involved stakeholders (developers, Architecture Owners, Enterprise Architect, and Product Owners) to enhance squads' autonomy. Practitioners say:

> "*Following this standardised process across the squads helps all parties to organise their architectural work instead of being dependant on some members from other squads or even relying only on the architect*"–P1, Agile Coach.
> "*The nature of the user stories requires responding quickly to our customers… The process is now more disciplined and organised, which consequently increased the speed of taking architectural decisions*"–P8, Senior Developer.

The autonomy of the squads, which is depicted in the third part of Fig. 5, was improved by utilising our approach for architectural governance as described in the following section. The introduced approach has strengthened squad's autonomy by decentralising architectural-based decision-making, which is depicted in the introduced structural change. Also, an architecture change management process was introduced to balance the Agile and architecture-based disciplinaries and hence create a holistic approach that enables squad's autonomy. A practitioner said: "*the employed rules in our Agile architecting process, which manages the interactions among all stakeholders, balances the Agile and architecture activities… These rules along with the structural changes have mitigated the dependencies and in turn has improved the autonomy of the squads*"–P1, Agile Coach.

The employed release strategy, which is depicted in the last part of Fig. 5, is impacted by aligning and governing architectural decisions. This impact is perceived in continuous delivery of working architecture (i.e., enablers) along to the working product. A practitioner said: "*we release the developed architectural-based tickets (i.e., enablers) whenever they are finished… Releasing such tasks enables the upcoming sprints*"–P7, Senior Developer.

4.4 Benefits and challenges of the architectural governance approach

This section presents the identified benefits and challenges of our proposed approach for architectural governance, which incorporates a structural change and an architecture change management process.

4.4.1 Benefits

Our approach for architectural governance has transformed architectural decision-making from a centralised decision-making process into decentralised process. This transformation has benefited the case study organisation in different aspects. A practitioner stated that "*I do not need to wait for the architect anymore… Instead, I can now get directly in touch with our Architecture Owner*"–P6, Senior Developer. However, enterprise architectural decisions need to be discussed within the architecture squad. A practitioner said: "*Taking decisions about how to integrate different components, or APIs might require a deep investigation by multiple Architecture Owners and the Enterprise Architect*"–P4, Senior Developer and Chapter Leader.

The proposed approach has resolved conflict in architectural decisions and mitigated technical risks among autonomous squads. In the proposed architectural governance approach, *"many complex technical and architectural aspects are left to evolve through iterative and incremental development and learning*"–P5, Senior Developer and Chapter Leader. Thus, complex technical and architectural decisions are finalised later in the lifecycle as depicted in the change management process. In addition, the squads are trusted to make local architectural decisions on their own without waiting for the Enterprise Architect. A practitioner said: "*we are encouraged to make architectural decisions on our own and with the support of our Architecture Owner*

without waiting for the architect as the technical details are left to evolve"–P8, Senior Developer. Architecture Owners usually try to avoid dictating specific architectural directions in favour of a collaborative team-based approach. A practitioner said: "*I try to encourage the members of my Chapter to collaborate and discuss architectural-based decisions to make the right architectural decisions*"–P5, Senior Developer and Chapter Leader.

However, developers might have conflict in architectural decisions and might not always come to an agreement. A practitioner said: "*Many developers are smart and strong-willed where they do not always come to an agreement… Someone should lead and facilitate the evolution of the architecture*"–P4, Senior Developer and Chapter Leader. Architecture Owners work with the Enterprise Architect to tackle enterprise-based architectural decisions, which impact two or more components in the software project. The Enterprise Architect said: "*Our Architecture Owners get back to me when they encounter a situation that requires making an enterprise-based architectural decision… Sometimes we discuss such enterprise architectural changes with other Architecture Owners if needed*"–P2, Enterprise Architect.

Furthermore, decentralising architectural-based decisions provided the Enterprise Architect with the opportunity to focus on creating technical and enterprise architectural alignment for the full software product. A practitioner said: "*I focus right now on aligning the enterprise architectural direction with the solution intent rather than being concerned for specific components*"–P2, Enterprise Architect. The alignment of enterprise architecture involves making sure that project vision and roadmap are carried out across architectural-based working items. "*The Enterprise Architect ensures that all Architecture Owners support the desired architectural capabilities and directions of the overall solution*"–P4, Senior Developer and Chapter Leader. This is achieved through close collaboration within the Architecture Squad by utilising the architecture change management process. A practitioner said: "*The introduced architecture change management process facilitates the alignment of architectural decisions across the whole organisation*"–P5, Senior Developer and Chapter Leader.

Moreover, our architectural governance approach has facilitated sharing architectural knowledge among autonomous squads. The Enterprise Architect works on transitioning architectural skills and training the Architecture Owners. A practitioner said: "*Our Enterprise Architect started arranging and conducting workshops to train and coach our squads in architectural related aspects*"–P4, Senior Developer and Chapter Leader. Also, Architecture Owners focus on creating technical alignment and disseminating technical and architectural-based knowledge. Architecture Owners arrange formal and informal sessions for architecture knowledge sharing. Practitioners say:

> "*our Architecture Owner arranges sometimes formal classes and other times brown-bag lunch sessions… In such sessions, we discuss planned subjects of interests which are related to our Chapter*"–P6, Senior Developer.
>
> "*we provide technical guidance around coding, security, architectural based aspects, monitoring the work, and so on*"– P5, Senior Developer and Chapter Leader.

In addition, our approach has improved software quality and mitigated obstacles to aligning architectural decisions across autonomous squads. Our case study organisation tries to balance the effort for architecting, by utilising the introduced change management process while having a decentralised architectural-based decision-making. This balance, in turn, facilitates the creation of generic software features, minimises wasted effort in architectural refactoring, and governs the architecture while strengthening squads' autonomy. A Senior Developer said: "*conducting proper Architectural Analysis within our Chapter and then evaluating and discussing the results, if needed, with the Enterprise Architect improved the quality of our produced work*"–P4. However, this process can be time-consuming. In this respect, a Senior Developer is warning that "*it can be time-consuming to make a good architectural decision that can be maintained easily in future without making a lot of refactoring… overlooking some architectural aspects that can be considered at the time being can cause much waste because of the needs for refactoring*"–P6, Senior Developer.

4.4.2 Challenges

The practitioners in our case study reported two challenges of our introduced approach. First, prioritising user stories without considering the architectural aspects can impact the planning activity negatively. The introduced architecture change management process does not facilitate screening the user stories by the Architecture Squad before or during the planning activity. A practitioner said: "*We do not go through the user stories, in our Architecture Squad, before planning… Yet sometimes we discuss them informally upon POs request*"– P5, Senior Developer and Chapter Leader. However, our change management process handles such a situation when encountering unpredicted architectural changes by moving from activity 6 to activity 1. A practitioner said: "*The architecture squad members do not join our planning session because we should be able to handle our work autonomously. We expect that our squad members should initiate the process of investigation and provide good input to either of the Architecture Owner or the Enterprise Architect*"–P3 Product Owner.

Second, handling an architectural spike might require making provisional architectural decisions or even multiple

possible decisions to conduct required exploration. Our case study organisation considers a spike as an investment to find out what should be built and how to build it. A practitioner said: "*We allocate some resources for complicated work items, ahead of the targeted delivery deadline, to find out what needs to be done… Such investments are considered a necessity to solve architectural issues, which work as an enabler for the next Sprint*"–P3, Product Owner. The complexity and uncertainty of spikes can result in taking multiple provisional architectural decisions. A practitioner said: "*Sometimes when we discuss a user story with the enterprise architect and the architecture owner, the outcome can be ambiguous as there is no concrete decision that can be taken… We have to explore multiple solutions*"–P6, Senior Developer. Hence, Architecture Owners or even senior developers could write just enough code to explore the architectural change before proceeding with development. This exploration process can be expensive for complicated architectural spikes. A practitioner said: "*Architecture owners might pair with another developer to explore complicated architectural spikes*"–P2, Enterprise Architect. Hence, the practitioners might sometimes need to employ an iterative and incremental way of architecture evolution by utilising the introduced change management process. This iterative and incremental way of architecting can be a time-consuming and yet powerful technique for risk-reduction.

5 Discussion

Coordinating and aligning software architecture among autonomous teams is identified as a challenge to Agile development (Martini and Bosch 2016; Salameh and Bass 2020; Stray et al. 2018). The Spotify model is an example of an Agile approach that is driven by creating autonomous yet aligned squads (Salameh and Bass 2018). However, the Spotify model does not provide guidelines for aligning and governing architectural-based decisions across the autonomous squad.

To explore *how software architecture can be governed and aligned by scaling the Spotify model,* a longitudinal embedded case study was conducted in a multinational FinTech organisation to have a deep understanding of how the Spotify model is being used. In this longitudinal embedded case study, we did a direct observation of Agile practices over 21 months and performed 14 semi-structured open-ended interviews. We found that the case study organisation was utilising a centralised architectural process by employing an architect because of the complexity of the software product. This centralised architectural decision-making has impacted squads' autonomy negatively. The authority of decision-making should exist at the operational level to enable teams' autonomy (Moe and Dingsøyr 2008; Salameh and Bass 2018). However, the Spotify model lacks practices for governing Agile architecting across autonomous squads.

5.1 Contribution to research and practice

In this study, we highlight a current challenge for architectural governance and alignment in large-scale Agile development programme (2–9 teams with less than 100 people) when using the Spotify model. This gap points us to contribute by (1) developing and evaluating an approach for architectural governance and (2) adapting the Heterogeneous Tailoring approach to accommodate our proposed approach.

Our approach for architectural governance introduces specific architect roles (i.e., Architecture Owners and Enterprise Architect) to transform architectural decision-making into the operational level. The role of Architecture Owner is assigned to Chapter Leaders or experienced developers of different skillsets and responsible for coordinating and handling Agile architecting. Since Chapters are formed based on competency areas—to align people horizontally—and Squads are aligned on the product-level while sharing the product, the Architecture Owners are aligned accordingly. The role of Enterprise Architect is a key role in resolving enterprise Agile architecture and scaling Agile architecting in large-scale organisations. Enterprise Architect works with solution management (i.e., Product Owners) and close to the Architecture Owners to resolve enterprise-based architectural decisions. In this approach, Architecture Owners are located in a virtual squad that is led by the Enterprise Architect.

We introduced an architecture change management process to guide the involved stakeholders (i.e., developers, Architecture Owners, Enterprise Architects, and Product Owners) in governing and aligning architectural-based decisions across the autonomous squads of the Spotify model. The architecture change management process comprises a set of activities and practices, which cover seven activities out of 11 architecting activities that have been identified by Yang et al. (2016). These architecting activities are Architectural Analysis and Synthesis (Activity 1 and 2), Architectural Evaluation and Impact Analysis (Activity 3), Architectural Refactoring (Activity 6), Architectural Maintenance and Evolution (moving from Activity 6 back to Activity 1). However, the activities of Architectural Description and Understanding are used to some extent at the enterprise architecture level. Also, Architectural Reuse is practised, according to our observation, within the squads and encouraged by Chapter Leaders.

We adapted the Heterogeneous Tailoring approach to accommodate our approach for architectural governance using the Spotify model. The adaptation of the Heterogeneous Tailoring approach revealed two new key features: product development and release strategy. Consequently, the

Heterogeneous Tailoring approach incorporates now four key features: product development, alignment, autonomous squads, and release strategy. First, product development is mainly impacted by practising the change impact analysis, which is introduced in the architecture change management process. This impact analysis produces new work items that enable software architecture. The emerged architectural-based work items need planning to enable future sprints. This change in product development facilitates the alignment of autonomous squads and hence improves their autonomy. Second, the alignment of architectural decisions was facilitated by employing both the structural change and the architecture change management process. Thirdly, the introduced change to the organisational structure has improved the squads' autonomy by governing and aligning architectural decision. In addition, the change management process has balanced the Agile and architecture processes to strengthen squads' autonomy. Finally, these autonomous squads work independently and collaborate to produce a working architecture, which enables future sprints, or/and a working product.

In particular, we found improvements mitigating the gaps highlighted before the introduction of our architectural governance approach, related to the following aspects as reported by the practitioners:

- Transforming architectural decision-making into decentralised-based decision-making.
- Creating technical and enterprise architectural alignment for the full software product, which in turn resolves conflicts in architectural decisions and mitigates key technical risks across squads.
- Sharing architectural knowledge among autonomous squads.
- Minimising wasted effort in architectural refactoring.
- Improving software quality.

5.2 Related work

Yang et al. (2016) conducted a systematic mapping study on the combination of software architecture and Agile development. The authors identified 11 architecting activities and 41 Agile architecting practices. The Spotify model has become influential among Agile proponents and hence formed the basis of Agile methods used in several other large-scale organisations of different contexts (Salameh and Bass 2018, 2019b, 2020). However, the Spotify model does not provide guidelines or Agile practices for aligning Agile architecture decision across autonomous teams (Salameh and Bass 2020). Whereas the new research we present here provides Agile activities, which utilise the identified architecting activities by Yang et al., that are employed in the introduced change management process. In addition, our approach introduces new roles within the organisation.

Bellomo et al. (2014) identified Agile architecture patterns and tactics that (1) influence the time and cost of software development, (2) improve project scalability, (3) focus on flexibility in deployment and controlling the cost and time for testing. However, the authors do not provide a framework or an approach for Agile architectural governance and alignment.

In a more abstract level, Nord et al. (2014) explored architectural tactics that support scaled Agile development and improve the alignment of the architecture and the development. They proposed an Agile architecture alignment using vertical and horizontal decomposition of the software architecture as well as matrix augmented-role team structures by conceptualising Scrum as an example. However, their work is not supported by scientific investigation following a rigorous research process. Our research does so by conducting an empirical investigation of the proposed approach in a specific domain (i.e., FinTech) of large-scale, which tailors the Spotify model.

Martini and Bosch (2016) have identified three architect roles, four sorts of teams, and some architecture practices, which are used to develop and evaluate a framework for Agile architecture in large-scale organisations developing embedded software projects. Their framework employs an Architect within each team, which works with a Governance Architect who functions as an intermediate role (i.e., coordinator) between the Chief Architect and the Agile teams. Also, their framework does not consider aligning the teams horizontally, which is the case in our evaluated approach. Our approach considers aligning the teams horizontally through Chapters and based on individuals' skillsets.

Through SaaS technology, FinTech organisations aim to (1) increase their agility to adapt quickly to new market opportunities, (2) improve data security, (3) offer unlimited scalability to upsize or downsize as needed, and (4) comply with industry protocols (Gimpel et al. 2018; Knewtson and Rosenbaum 2020). To satisfy these aims and objectives, our approach decentralises the architectural decision-making process and creates a technical and enterprise architectural alignment among the autonomous squads. Other software development contexts might not be subject to the aforementioned aims and objectives. However, our research did not identify any evidence of which our approach could be limited to specific contextual or project factors. Our evaluated approach for architectural governance introduces vertical and horizontal alignment by tailoring the Spotify model, which was not the case in the previous empirical research (Bellomo et al. 2014; Martini and Bosch 2016; Nord et al. 2014).

5.3 Limitations and threats to validity

Even though the practitioners reported improvements concerning architectural risk management across autonomous squads, some practitioners reported that architectural debt management remained a challenge that needs well-defined practices. This architectural debt management is perceived in the absence of practices that facilitate the investigation of architectural aspects before prioritising new user stories, which might impact the planning activity negatively. However, complex architectural decisions are finalised later in the lifecycle as depicted in the architecture change management process. Such a phenomenon is concerned with risk-informed architecture decisions about which architecture changes need to be conducted for having an acceptable ratio of cost or impact. This phenomenon is already studied from different angles (Edith et al. 2013; Nord et al. 2014).

Our study is subject to population bias. The introduced approach for architectural governance was evaluated in a single case study organisation, which develops a FinTech project using the Spotify model. However, we performed a triangulation in data collection (observation data (i.e., memos) and semi-structured open-ended interviews) and analysis. The participant interviewees were holding different roles: developers, Architecture Owners, Enterprise Architect, Product Owner, and Agile Coach. The observed ceremonies include backlog grooming, planning sessions, retrospectives, daily stand-ups, POs synchronisation meetings, and Architecture squad meetings.

The research limitations are discussed by considering the criteria introduced by Lincoln and Guba for judging the quality of research through credibility, dependability, transferability, and confirmability (Lincoln and Guba 1985). *Credibility* is concerned with compatibility between the respondents' opinions and the researcher's interpretation. The first researcher was embedded in the case study organisation and observed employed Agile practices for 3 months after introducing the developed approach for architectural governance. Then, we have iteratively interviewed the practitioners—triangulating perspectives from different roles. *Dependability* is concerned with the ability to replicate the conducted research. This study was limited to the FinTech industry, in which the first author works as senior software engineer. Also, the selection of participant interviewees was limited by their willingness to participate in this study. *Transferability* refers to the extent to which the findings from one context are applicable to another while understanding the circumstances that affect the studied context. We cannot fully transfer the results to large-scale Agile software development since we conducted a single case study. Hence, we limit our results to large-scale (2–9 teams with less than 100 people) organisations using the Spotify model to develop SaaS, such as FinTech services. These large-scale organisations focus on technological innovation and agility to adapt quickly to new market opportunities while complying with industry-specific regulations, such as AML, PCI and KYC. *Confirmability* examines the researcher's objectivity in relation to the studies context. The proposed approach to architectural governance was validated in a real-world organisation. In this work, also, two researchers were involved in the study and a triangulation analysis was conducted. In this analysis, the interview data (practitioners' perceptions) were analysed and compared with the observation data (i.e., memos) to prevent any suspected deviation between "semi-structured interviews" view of matters and the "real" case.

Given the time limitation of this study, we could not aim at a complete evaluation within the whole organisation, which is, however, in the researchers' long-term goal. The preliminary evaluation gave the researchers and the practitioners, in the case study organisation, valuable insights on how the developed approach for architectural governance can be used in practice. In fact, our analysis did not identify any evidence of which our approach could be limited to specific contextual or project factors.

6 Conclusion

Software architecture is one of the key technical advances in the field of software engineering over recent decades. The role of software architecture is important in large-scale Agile development because several teams need to work together to release a single product without degrading teams' autonomy. We have identified a challenge in governing and aligning Agile architecture across autonomous squads when using the Spotify model. The Spotify model lacks practices addressing Agile architecture governance.

Our study addresses the research question: *How can software architecture be governed and aligned by scaling the Spotify model?* To answer this question, we conducted a longitudinal embedded case study in a multinational FinTech organisation using the Spotify model to gain a deeper understanding of how the Spotify model is used. Then, we developed and evaluated an approach for architectural governance by conducting an intervention embedded case study. This embedded case study lasted 3 months, during which eight semi-structured open-ended interviews were conducted. The collected data were analysed using Thematic Analysis and informed by selected Grounded Theory techniques.

There are two key contributions of this paper. First, we developed and evaluated an approach for architectural governance when using the Spotify model. We have presented in this paper the characteristics of our proposed architectural governance approach, its benefits, and challenges. Second,

we tailored the novel Heterogeneous Tailoring model to accommodate our architectural governance approach.

Our architectural governance approach aims to improve the alignment of squads (i.e., teams) without compromising their autonomy. This approach incorporates a structural change and an architecture change management process. The structural change comprises (1) empowering Chapter Leaders and experienced developers with the novel role of Architecture Owners, (2) changing the responsibilities of the Architect to focus on Enterprise Architecture, and (3) locating the new Architecture Owners in a virtual squad that is led by an Enterprise Architect. Compared to previous approaches and frameworks (Bellomo et al. 2014; Martini and Bosch 2016; Nord et al. 2014), our approach introduces horizontal architectural alignment, based on the individuals' skill sets, as well as vertical alignment on the product level. To streamline the architecture process, we also introduced an architecture change management process, which aims to guide stakeholders (i.e., developers, Architecture Owners, Enterprise Architects, and Product Owners) in governing and aligning architectural decisions among autonomous squads.

Our proposed approach has influenced the Heterogeneous Tailoring approach by shifting its boundaries to emphasise on other aspects. Consequently, the Heterogeneous Tailoring approach comprises now 4 key features: Product development, alignment, autonomous squads, and release strategy. These key features interact together to enable and strengthen the autonomy of squads and ultimately increase squads' productivity and improve their innovation.

The practitioners in our case study reported several benefits of introducing the architectural governance approach to the Heterogeneous Tailoring approach. These benefits include: (1) decentralising the architectural decision-making process, (2) creating technical and enterprise architectural alignment for the full software product, (3) resolving conflicts in architectural decisions and mitigating key technical risks across autonomous squads, (4) sharing architectural knowledge among the squads, (5) minimising wasted effort in architectural refactoring, (6) improving product quality and mitigating obstacles to aligning architectural decisions across autonomous squads and (7) balancing the effort for architectural quality facilitates the creation of generic software features. Hence, the alignment and governance of architectural decisions have improved the autonomy of squads.

References

Abrahamsson P, Babar MA, Kruchten P (2010) Agility and architecture: Can they coexist? IEEE Softw 27(2):16–22

Anderson C, McMillan E (2003) Of ants and men: self-organized teams in human and insect organizations. Emergence 5(2):29–41

Bass J, Salameh A (2020) Architectural Governance Interview Guide. University of Salford, UK. https://doi.org/10.17866/rd.salford.12613424.v1

Bellomo S, Kruchten P, Nord RL, Ozkaya I (2014) How to Agilely architect an Agile architecture. Cut IT J 27(2):12–17

Booch G (2009) The defenestration of superfluous architectural accoutrements. IEEE Softw 26(4):7–8

Braun V, Clarke V (2006) Using thematic analysis in psychology. Qual Res Psychol 3(2):77–101

Buschmann F, Henney K (2013) Architecture and agility: married, divorced, or just good friends? IEEE Softw 30(2):80–82

Cockburn A, Highsmith J (2001) Agile software development, the people factor. Computer 34(11):131–133

Conboy K, Carroll N (2019) Implementing large-scale Agile frameworks: challenges and recommendations. IEEE Softw 36(2):44–50

Dingsøyr T, Dybå T, Moe NB (2010) Agile software development: current research and future directions, 1st edn. Springer Publishing Company (**Incorporated**)

Erdogmus H (2009) Architecture meets agility. IEEE Softw 26(5):2–4

Gimpel H, Rau D, Röglinger M (2018) Understanding FinTech start-ups—a taxonomy of consumer-oriented service offerings. Electron Mark 28(3):245–264. https://doi.org/10.1007/s12525-017-0275-0

Glaser BG (1998) Doing grounded theory: issues and discussions. Sociology Press

Hammond S, Umphress D (2012) Test driven development: the state of the practice. In: Proceedings of the 50th Annual Southeast Regional Conference, Tuscaloosa, Alabama

Hoda R, Noble J, Marshall S (2013) Self-organizing roles on Agile software development teams. IEEE Trans Software Eng 39(3):422–444

Kilu E, Milani F, Scott E, Pfahl D (2019) Agile software process improvement by learning from financial and fintech companies: LHV Bank Case Study. Software Quality: the complexity and challenges of software engineering and software quality in the cloud, Cham

Knewtson HS, Rosenbaum ZA (2020) Toward understanding FinTech and its industry. Managerial Fin 46(8):1043–1060

Kniberg H (2014) Spotify Squad framework. Retrieved July, 2020 from https://medium.com/project-management-learnings/

Kniberg H, Ivarsson A (2012). Scaling Agile @ Spotify with tribes, squads, chapters & guilds. Retrieved June, 2020 from https://blog.crisp.se/wp-content/uploads/2012/11/SpotifyScaling.pdf

Lincoln YS, Guba EG (1985) Naturalistic inquiry. Sage Publications, Berlin

Linders B (2016) Don't copy the Spotify model. https://www.infoq.com/news/2016/10/no-Spotify-model

Martini A, Bosch J (2016) A multiple case study of continuous architecting in large Agile companies: current gaps and the CAFFEA framework. In: 2016 13th Working IEEE/IFIP Conference on Software Architecture (WICSA)

Moe NB, Dingsøyr T (2008) Scrum and team effectiveness: theory and practice. Agile Processes in Software Engineering and Extreme Programming, Berlin

Moe NB, Dingsøyr T, Dybå T (2008) Understanding self-organizing teams in Agile software development. In: Proceedings of the 19th Australian Conference on Software Engineering (aswec 2008)

Moe NB, Olsson HH, Dingsøyr T (2016) Trends in large-scale Agile development: a summary of the 4th workshop at XP2016. In: Proceedings of the Scientific Workshop Proceedings of XP2016

Nord RL, Ozkaya I, Kruchten P (2014) Agile in distress: architecture to the rescue. Agile methods. Large-scale development, refactoring, testing, and estimation, Cham

Robinson H, Segal J, Sharp H (2007) Ethnographically-informed empirical studies of software practice. Inf Softw Technol 49(6):540–551. https://doi.org/10.1016/j.infsof.2007.02.007

Salameh A, Bass JM (2018) Influential factors of aligning Spotify squads in mission-critical and offshore projects—a longitudinal embedded case study. Product-Focused Software Process Improvement, Cham

Salameh A, Bass JM (2019a) Spotify tailoring for B2B product development. In: Proceedings of the 2019 45th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)

Salameh A, Bass JM (2019b) Spotify tailoring for promoting effectiveness in cross-functional autonomous squads. Agile processes in software engineering and extreme programming—Workshops, Cham

Salameh A, Bass JM (2020) Heterogeneous tailoring approach using the Spotify model. In: *EASE '20* proceedings of the evaluation and assessment on software engineering, Trondheim, Norway

Šmite D, Moe NB, Levinta G, Floryan M (2019) Spotify guilds: how to succeed with knowledge sharing in large-scale Agile organizations. IEEE Softw 36(2):51–57

Šmite D, Moe NB, Floryan M, Levinta G, Chatzipetrou P (2020) Spotify guilds. Commun ACM 63(3):56–61

Stray V, Moe NB, Hoda R (2018) Autonomous Agile teams: challenges and future directions for research. In: Proceedings of the 19th international conference on Agile software development: companion, Porto, Portugal

Yang C, Liang P, Avgeriou P (2016) A systematic mapping study on the combination of software architecture and Agile development. J Syst Softw 111:157–184