An approach to align socio-technical dependencies in large-scale software development

Binish Tanveer, Ehsan Zabardast, and Javier Gonzalez-Huerta
Software Engineering Research Lab (SERL)
Blekinge Institute of Technology, Karlskrona, Sweden
binish.tanveer@bth.se, ehsan.zabardast@bth.se, javier.gonzalez.huerta@bth.se

***Abstract*—Seeking the advantages delivered by agile methods in small-scale software development, large organisations are also adopting agile methods. However, scaling results in a huge growth of socio-technical dependencies that can lead to waiting time, delays, and defects and hinder the teams' ability to recognize their own responsibilities. This research proposes an approach to enable teams' autonomy and clarifies teams' responsibility assignments by aligning socio-technical dependencies. By utilising compile-time, run-time, and task dependencies, our approach identifies the wasteful dependencies between the social structures (teams) and the corresponding technical structures (architecture) and also suggests improvements. The initial results suggest that the approach correctly identifies the wasteful dependencies that are hindering teams' responsibility assignments. The suggested solution proposals are also considered useful. Awareness of such wasteful dependencies is the first step toward being able to handle them successfully.**

I. Introduction

Seeking the advantages such as shorter lead times, increased quality, and customer value, delivered by agile methods in small-scale software development, large organisations that develop software-intensive products and services are also adopting agile methods [1], [2]. However, as the organisation scales, it results in an exponential growth of socio-technical inter-dependencies due to increased inter-team coordination, increased dependencies between organisational units, and the evolution of the software architecture [3], [4], [5], [6], [7], [8]. Such dependencies can lead to handoffs, waiting time, delays, defects, and increased cognitive load [9], [10]. With these dependencies, teams have to deal with additional tasks, and they might be unaware of their commitment, responsibilities, and ownership of their developed components. These dependencies are barriers to team autonomy [11], which is the key to agility.

An analysis of the scope and complexity of these dependencies presents an opportunity for directing improvements by identifying value-adding, and wasteful dependencies. Removing the wasteful dependencies (like unnecessary communication) among teams could be one strategy to promote teams' responsibility assignment and autonomy. However, this is not straightforward because there exists a strong association between technical structures (system architecture) and social structures (organisation communication structures) that design it [12]. An organisation's dependencies tend to mirror its architecture and vice versa [13], [14], [15]. Hence, removing a dependency on organisational structures cannot be done without impacting its architectural counterpart, since both influence each other [15], [14]. Moreover, it is imperative to have an alignment among the socio-technical structures because misalignment has a negative effect on the schedule and quality and can even lead to project incompletion [16].

Through conducting an empirical investigation, in this paper, we share our experiences of how a large company decided to take an improvement initiative of reorganising their teams in pursuit of enabling their teams recognise their actual responsibilities, and how we supported them to make this initiative more effective through our approach. Our contribution is an approach to align socio-technical dependencies and use the approach to enable teams recognise their responsibilities.

Conway recognised that the structure of architecture dictates the coordination needs among teams [17], therefore to enable the alignment, he suggested managing this coordination by *"splitting into components with limited technical dependencies, and these components would need to be assigned to no more than one team (resulting in a homomorphic relationship between the architecture and the organisation) "*[16]. Based on Conways' law, socio-technical congruence (STC) appeared that measure coordination by finding the fit or alignment between the technical dependencies and the social coordination in the project [18]. Several studies followed and proposed different metrics to measure STC [19], [20], [18]. Parnas [21] suggested using modular design as it enables independent decision-making. In his view, modular design will limit the interactions among modules and favour parallel development by independent teams with minimal interactions among the teams [16]. Another study found that the system architecture enabled the teams to make decisions independently and increased their autonomy [22]. Some researchers suggested striking a balance between organisational control and team autonomy [23]. It has been established that coordination by architecture [15] could strengthen team autonomy, but as the modules are "never truly independent", the effectiveness of such coordination seems limited [24]. It is observed having cross-functional teams support team autonomy [25], however when teams are "top-managed", such autonomy is compromised [25]. It has also been observed that *"the technical mechanisms that cause these task interdependency are invocations across modules (assuming a module is a task assignment to a single team) [16]."*

A recent trend is the adoption of microservice architecture due to its characteristics like faster delivery, improved scalabil-

ity, and greater autonomy [24]. The microservice architecture builds software as "a suite of small services". Though theoretically, the strict autonomy of microservices is known to enforce modularisation, whether it actually works in practice is yet to be explored [26]. Ideally, individual microservices are owned by individual teams [26]. However, it is seldom the case that, in reality, an individual team is the sole contributor to a microservice. Due to scaling or lack of resources or expertise, staff turnover, or personnel reallocation, teams end up having additional dependencies in the form of extra assignments and responsibilities for several different services they do not own or do not have experience in, and even for non-technical work. Faced with a similar scenario, a large company decided to take an improvement initiative in pursuit of assigning teams their responsibilities while reducing the dependencies among them and we are supporting them to make this initiative effective. We have investigated the initiative in the next section.

The paper is structured as follows: Section II presents the study, and Section III highlights the results and discusses them. Section IV narrates our approach where Section V discusses the preliminary results, the practical implications of our approach and concludes the paper.

II. Study with industrial partner

We conducted a study with an industrial partner as a part of an ongoing collaboration. This section describes the context, goals, sample, data collection, and analysis.

A. Context

A large company decided on an improvement initiative where they proposed a major reorganisation by separating and reallocating the teams into two main units: 1) development unit - related to the core development of products and services. They develop features for internal customers (developers) for company-wide usage and 2) product unit - related to developing features for end customers (clients), creating business strategies, long-term company's goals, and road maps. There are three states of the organisation with respect to the initiative:

- Pre-initiative: The past state before the initiative was proposed.
- Partial-initiative: The current state where the initiative has partially taken effect, i.e., some of the roles and teams still need to be reallocated.
- Post-initiaitve: The future state when the initiative will ultimately be effective.

In this study, we are referring to the "partial-initiative" state which is the state at the time of writting this paper. Though the company proposed reorganising the social structures, they still need to cater to the impacted technical structures, without which they will not achieve the expected outcomes.

Starting with the initiative itself, we conducted an mixed-methods study (i.e., interviews and archival analysis) to investigate *What is the improvement initiative, and the rationale behind it?* Before investigating how the technical structures might be impacted by the initiative, we also asked the interview participants their views about the current architecture, teams' allocation, and main responsibilities. We then conducted analysis of the technical structures in the data with the research question: *what socio-technical dependencies are refraining teams' from realising their responsibilities and hindering autonomy?*. The purpose was to find the impacted technical structures, due to changed social structures, that are wasteful and suggest some improvements. We believe that identifying and removing these dependencies will ensure the effectiveness of the change in terms of team responsibility assignments.

We take a stance that in order to enable teams to recognise their responsibilities and be autonomous, it requires more than just emphasizing having a modular design, adopting architectural principles, operationalising microservice architectures, or examining the team's inner structure and workings. It is imperative to handle the barriers like the wasteful socio-technical dependencies that hinder autonomy leaving teams unclear about their responsibilities.

B. Goal

The study had two goals. As mentioned earlier, firstly, we wanted to investigate the improvement initiative (i.e., the change), the rationale behind the change, the expected benefits, and the anticipated challenges, if any. Secondly, we were interested in investigating the technical structures associated with the changed social structures that needed to be adjusted accordingly. In particular, we wanted to identify dependencies that might still exist in the technical structures despite the reorganisation of the teams and affected their responsibility assignment and autonomy negatively.

C. Sample and Population

The company we conducted the study with is a large company that wants to remain anonymous. The company's domain is the banking sector, with around 700 employees. It develops smart banking and financial solutions. We have selected the case and the sample by convenience (i.e., due to availability and access) as the company wanted to improve its ways of working by learning from its results and hence was willing to participate. The company employs agile practices and DevOps with teams working with practices like Scrum and Kanban. It is using a microservice architecture and is currently in the process of scaling. The company has recently proposed a change, i.e., it has reorganised itself in pursuit of clarifying teams of their responsibilities, promoting their autonomy, and improving ways of working at scale.

There are total 37 teams. Four representatives from three teams (Dev-A, Dev-B, and Dev-C in Table I) participated in the study. The teams Dev-A and Dev-B are planned to work under *the development unit*, whereas team Dev-C is planned to work under *the product unit*. In the pre-initiative state, teams Dev-A and Dev-B were working under one single unit i.e., *the product unit* with no separation of concerns, whereas now

in the partial-initiative state, they proposed to have separate units. Dev-C is not affected in terms of reallocation.

TABLE I
PARTICIPANT INFORMATION

Role	Team	Experience (years)
Development Manager	Dev-A	3
Product Owner	Dev-A	1
Architect & Product Owner	Dev-B	5
Architect & Senior Developer	Dev-C	3

We interviewed one development manager (team Dev-A) and two product owners (PO) of two teams, Dev-A and Dev-B, respectively (See Table I). In the post-initiative state, the PO (Dev-B) will have a new and dedicated role of an architect, whereas in pre-initiative state, the PO (Dev-B) was also an architect of the whole development unit. We also interviewed one senior developer (SD) in team Dev-C, who is also the architect, and like the pre-initiative state, their team is still planned to work under the product unit.

The interviews were conducted online for 30-40 mins each, in which the two first authors participated, one asked the questions, and the other took notes. With the consent of the participants, the interviews were also recorded. We selected these four representatives to capture a different perspective on the change since the participants were either the change initiators or being directly impacted or indirectly impacted by the change. Hence they possess the relevant knowledge and experience about the change, the rationale, the outcomes, and its impact. Regarding the archival data, we focused on the components and services developed by these teams as well as the associated socio-technical dependencies.

D. Design

To achieve the study's first goal, we designed and conducted interviews with four participants. The interview included demographic questions and other open-ended questions related to the change and the rationale behind initiating the change. To achieve the second goal, we needed archival data from the development tools that the company uses, to identify the dependencies that might be confusing the responsibility assignments, hindering team autonomy, and thus are needed to be adjusted accordingly. We considered the following dependencies in our approach:

- **Compile- and run-time dependencies:** A compile-time dependency is when a component X is using code from another component Y (e.g., through an *import* in Java, then X depends on Y [27]. Compile-time dependencies are typically hardcoded in the code. A run-time dependency is when a service A is calling another service B (e.g., by invoking functionality through a REST [28] API), then A depends on B.
- **Task dependencies:** In order to identify task dependencies between two components, we use the tasks' assignments that occur among the developing teams when components are being developed. We use Jira tickets as a proxy to identify such task dependencies. In other words, when two components appear in the same Jira ticket, there is a task dependency between them.

E. Data Collection

In this section, we describe the data collected for this study.

1) Qualitative Data: Through interviews, we collected data about their opinions and perspectives regarding the change itself and the rationale behind the change The interviews were recorded and transcribed for analysis. Information regarding the architecture, teams allocation to specific areas, and, their main tasks were also collected. Using thematic mapping [29], the interviews were coded, and themes were generated to identify the rationale and expected outcomes of the change.

2) Quantitative Data: The following data was collected for identifying the dependencies. The collected data represents one year of development activity from November 2021 to November 2022.

- **Compile- and run-time dependencies:** We extract compile-time dependencies through build files associated with each component. The data was collected from the company repositories. Run-time dependencies were captured using Jaeger[1] tracing tool, through the Jaeger API.
- **Task dependencies:** We extracted the task dependencies through Jira[2] tickets. The data was collected through the Jira API. We collected complementary data from the BitBucket[3] API to identify the task dependencies.

We processed the collected raw data to identify the compile- and run-time dependencies, task dependencies, and potential wasteful dependencies among them which we refer to as *potential problematic cases*. The data processing procedure is illustrated in Fig. 1.

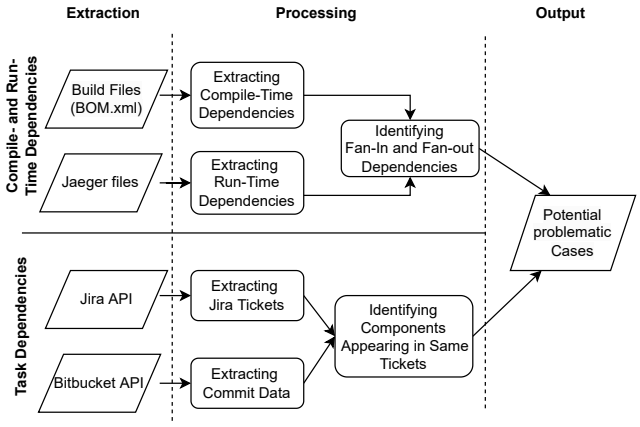

Fig. 1. The data processing procedure.

We distinguish between fan-in and fan-out dependencies for compile-time and run-time dependencies. Fan-in dependencies are the incoming dependencies to a particular component, i.e.,

[1]https://www.jaegertracing.io
[2]https://www.atlassian.com/software/jira
[3]https://bitbucket.org/product

the other components that are dependent on a component. Fan-out dependencies are outgoing dependencies from a component, i.e., the other components that a component depends on.

III. Results and Discussions

This section describes interviews' results with discussion.

A. Rationale for the Change

The results of the interviews show the following rationale for doing the change:

- *Clarity and ownership of responsibilities:* According to the participants, the teams in the development unit were unable to focus on their own work since they were supporting other teams in the product unit with different tasks than fulfilling their own commitments. They were being overburdened with unauthorised and even non-technical tasks. As a result, they were unclear about their own work assignments and responsibilities.
- *Centralised decision-making :* The participants mentioned that major decisions were made by managers in the product unit and sort of pushed to the developers for implementation. Being experts on the technical complications and implications of the decisions, developers should decide on when and how to implement certain things. Hence, it was imperative to decentralise decision-making, especially with respect to the technical aspect, and give them the authority to derive things or have their opinions on board while making such decisions, nonetheless.
- *Lack of right communication channels:* The participants mentioned that before the company started to scale, informal communication was common and teams used to solve problems through discussions during coffee breaks. Queries related to overall architecture were a common topic and people depended on others to find answers as there was no one responsible for it. As the company grew, such unrestricted coordination diminished, leaving the teams with no forum to raise their issues with respect to the overall architecture. Hence, there arose a need to have a formal communication channel by defining a dedicated team in the development unit, responsible for issues related to architecture.
- *Lack of unified view of architecture:* Related to the previous point, the participants also mentioned that teams missed a bigger picture of the overall architecture since there was no dedicated team responsible for providing a unified view of architecture. The company is using microservice architecture for its new development as it supports scalability. They are also maintaining and using a monolith architecture of legacy code. Hence it is imperative to have a dedicated team whose responsibility is to provide a unified view of the overall architecture and solves issues of teams working with different services and using the old architecture.

The above-mentioned points show the reasons behind initiating the change, i.e., separating the product and development units and assigning teams to their proper units. Among our participants we had both 1) change initiators and 2) the ones who were either directly or indirectly impacted by the change. Since the participants had different roles, it was interesting to find their different perspectives on the reasons for the change. For example, the development manager and PO (Dev-A) felt that certain decisions were sort of pushed on them instead of having autonomy and hence they initiated the change. Whereas the SD (Dev-C), (being indirectly impacted by the change product unit), acknowledges the need for the change and realises that the teams in development unit had different issues to bear. According to SD (Dev-C), the team and product managers of product unit are in direct contact with end customers. They know market trends, create features for the customers, need to respond to their problems immediately, and also create long-term road maps and company goals. It was getting difficult for the product unit to create road maps for the teams in the development unit since the development teams had so many tasks to take care of in addition to providing support to other teams and re-prioritising tasks to satisfy customer demands and solving issues simultaneously. SD (Dev-C) anticipates a positive impact of the change but still wants to maintain the collaboration in some formal, yet controlled manner, with the development unit considering them experts in technical tasks.

The PO (Dev-B), being directly impacted by the change and will have a dedicated role of an architect in a newly created team of architects, also had interesting insights. According to PO (Dev-B), since the company is scaling, they needed a more unified view of the architecture. PO (Dev-B) expressed that teams have architects in different areas, but they were diverging from the main architectural idea. They had their own ideas and diverse ways of doing things which made things unclear about how we actually would build services. With the change, PO (Dev-B) will have a dedicated role in the new team of architects, with a lot less burden, more focus, and time to provide a unified way of managing and orchestrating the microservice architecture. Moreover, the PO (Dev-B) shared the view of SD (Dev-C) that the roadmaps from the product unit were not fitting to their ways of working as the teams in the development unit were occupied with several technical tasks on priority in addition to supporting other teams. As a result, (Dev-B) have always felt detached from the product unit, and thus with this change they feel more satisfied as they have been reallocated to the development unit where they felt actually belonged.

Through the interviews, we have seen that the company realised that with scaling the old ways of working needed adaptation. The participants, in their own perspectives, shared some common issues like the teams were supporting others and losing focus, remaining unclear of their own responsibilities, and their agility was being hampered due to which they decided to take the initiative. There was a pressing need for a unified architectural view and some formal way of communicating and managing architectural matters. After analysing the interviews, we have observed that the company is aware of the problems they are facing in the wake of

scaling. Therefore, to enable agility and be able to scale, the company needs to adapt to the new ways of working and hence they initiated the change. They also mentioned that time and communication are the main challenges for this initiative to take effect. Finally, we observed that the company has adopted microservice architecture due to its characteristics like faster delivery, improved scalability, and greater autonomy. As mentioned earlier, the strict autonomy of microservices is known to enforce modularisation that supports autonomy at least in theory [26]. However, we observed that it is not really working in practice. The reason, according to our analysis is, the teams have additional dependencies in the form of extra assignments and responsibilities for several different services or even for non-technical work which is refraining them to realise their responsibilities and be autonomous despite the use of microservices. Hence, to get the benefits promised by microservice architecture, an organisation has to provide support mechanisms too like removing barriers (wasteful socio-technical dependencies, in this case) that hinder team autonomy, for example.

IV. Our Approach

As mentioned earlier, though the company has decided on taking the initiative by reorganising its social structures, there is a need to reorganise the corresponding technical structures, since there exists an association between them [12]. Hence, to achieve the second goal of our study we propose our two-step approach. We propose aligning socio-technical dependencies through 1) identifying the wasteful dependencies existing in the impacted technical structures and 2) providing solutions to the wasteful dependencies. While working with the company data, we have observed that the components developed by the teams are, as one can expect, not truly independent. There were invocations across components that were required to be changed in the wake of the initiative. In our previous study, we have shown with evidence that in some cases, the teams that contribute to the components the most are not always the one who own them [30]. After examining the data in light of the change initiative, we have identified certain potential wasteful dependencies that hamper team responsibility assignment and autonomy. The steps and cases are explained below:

A. Identifying Potential Wasteful Dependencies

Using the process as shown in Fig 1, we performed an initial analysis of the company data, where we investigated 272 components belonging to 37 teams. Overall, we have identified a total of 5779 dependencies out of which 1055 are found to be potential problematic cases in the company. Due to data privacy and anonymity, we can only show a proxy of potentially problematic cases.Hence for this paper, we are presenting an example illustration in Fig. 2 that reflects a few of the identified original problematic cases. Fig. 2 illustrates an example of two teams **A** and **B** in the company planned to work for the development unit and product unit respectively. Team **A** owns 5 components and team **B** owns 4 components. The size of each component represents the number of dependencies on it. The solid lines between the components represent compile- and run-time dependencies and dashed lines represent task dependencies. Compile- and run-time dependencies are directed, i.e., they represent fan-in and fan-out technical dependencies between two components.

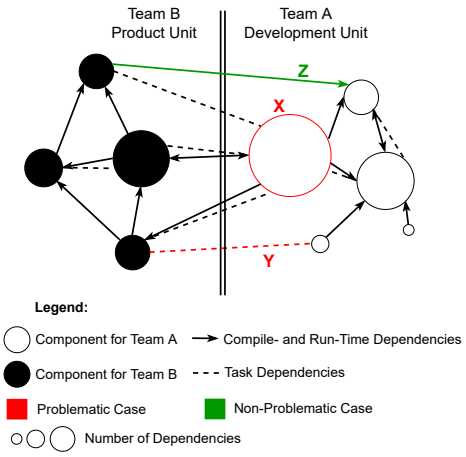

Fig. 2. An example with identified problematic cases by investigating the dependencies.

In this example, we present three such problematic cases:

- **Case X (Problematic):** The component in this case is owned by team **A**. It has a significant amount of both compile- and run-time and task dependencies most of which come from the components owned by team **B**. This is a problematic case because while the component is owned by team **A**, there are more dependencies from the components owned by team **B**. This case might be introducing delays due to team B being responsible for code reviews, or handovers if the task at hand cannot be resolved by team A, or it might. X is a component with many dependencies (as represented by the size). There are task dependencies between X and other components without compile- or run-time dependencies. The mentioned dependencies are across the teams. The combination of these factors determines the selection of such case.
- **Case Y (Problematic):** There is a task dependency between two components, one component belongs to team **A** and the other belongs to team **B**, without any compile- or run-time dependency between the components. This case is problematic because neither team (i.e., whoever is deemed responsible for the task) would expect these two components to be connected. The task can be delayed due to cross-team code reviews or potential handovers.
- **Case Z (Non-Problematic):** There is a compile- or run-time dependency between two components, without task dependencies between them. The component owned by team **B** is dependent on the component owned by team **A**. This is not problematic, from the point of socio-technical perspective, because the only dependency is a compile- or run-time dependency between components. Of course, these dependencies might need to be scrutinised, if they

are not needed. Z is a single compile- or run-time dependency. Such dependencies are common and, most of the time, unavoidable.

Our analysis showed the initiative would result in the misalignment of socio-technical dependencies since the company did not consider the impacted technical structures while deciding the change. Hence, there is a need to align the socio-technical dependencies by resolving the problematic cases.

B. Potential Solutions to Wasteful Dependencies

Here, we propose potential solutions to the cases:

- **Potential solution for cases like Case X:** To resolve cases like X and alike, we suggest moving it from team **A** to team **B**. The ownership of the component should be transferred to team **B** if their degree of contribution is found to be more than team **A**. Our suggestion is based on our findings in our previous study [30] in which we analysed the component's ownership and contribution alignment using our OCAM model [30]. For resolving cases like these, the company can apply the model to analyse such cases. OCAM will provide more information on the degree of contribution from each team.
- **Potential solution for cases like Case Y:** In general, when there is a task dependency among components, there exists a corresponding either compile- or run-time dependency. We have observed cases like Case Y which are very peculiar since two components have only task dependencies. According to our analysis, a single solution might not be possible for cases like this. Instead, a deeper investigation is required to reach a solution. It is because each case requires considering several other factors such as the type of components, type of task, and significance of the identified task dependency along with the historical data. It is possible that down the road the task got obsolete but never got removed from Jira, or due to reallocation of tasks, this task was moved to the non-priority list, etc. In the earlier case, one possible solution could be to remove the task dependency and in the latter case, OCAM could be used to decide the ownership of the component having this task dependency, however, certain criteria are also needed to decide the ownership in such cases. In short, a thorough root cause analysis is required to analyze and decide on the solutions to such cases.

V. Preliminary results, practical implications, and conclusion

In Nov 2022, we presented the initial results of our approach to the partner company. The purpose was to get preliminary validation on the correctness of the identified problematic cases and the overall usefulness of the approach. We conducted a focus group session in which practitioners who were the change initiators along with the authors participated. The cases like X and Y specifically related to the change were visualised and the potential solution was discussed. The practitioners confirmed the correctness of the identified cases. They appreciated the initial findings and were keen on finding the analysis of socio-technical dependencies in the wake of the change for the whole company. They considered the approach highly useful for the effective execution of the change. However, for cases like Case Y, the company also appreciates conducting a deeper investigation, and a root cause analysis to know why such cases exist and what can be done to avoid them. We are currently working on a complete analysis of the data to identify all such problematic cases. The identification of social structures could be made comprehensive by complementing it with approaches like FLOW modeling [31] which helps in identifying challenges and improvements related to information flow in processes. Techniques like change impact analysis [32] could be complemented in our approach to identify additional dependencies like the number of co-changed components affected by implementing a certain task or the number of test cases required etc. After completing the analysis, we plan to conduct an extensive evaluation regarding the perceived usefulness of our approach [33] where we could present the potential solutions and incorporate the company's detailed feedback to improve the approach. We intend to apply the approach in practice when the change will actually takes effect. Furthermore, together with our industrial partner, we intend to define processes, metrics, and measures that we will use to perform the final evaluation to find the effectiveness of the proposed approach.

Our approach has several practical implications. Using this approach enables the organisations to reduce socio-technical dependencies in order to remove waste and improve ways of working continuously in a flexible agile manner. It can also be used for planning future improvement initiatives more effectively. In this particular case, we are operationalising it to enable teams' autonomy and their ability to recognise their responsibilities however, in other contexts it enables the practitioners to find the impact of the change they made. The visualisations of these socio-technical dependencies provide transparency which is beneficial for the practitioners as it provides awareness of dependencies and the ability to identify the wasteful ones. Awareness of such wasteful dependencies is the first step toward successfully handling them and making informed decisions.

In this paper, we shared our experiences of how a company at scale, realising the old ways of working are not functioning, decided to reorganise itself. Our analysis showed the reorganisation resulted in the misalignment of socio-technical dependencies since the company did not consider the impacted technical structures. Misalignment of socio-technical dependencies can have disastrous consequences [16], whereas aligning them at a large scale is hard. Handling the wasteful socio-technical dependencies is critical as such dependencies have negative effects. Hence, we supported the company to make their initiative effective by proposing an approach to align socio-technical dependencies.

Acknowledgment

This research was supported by KK foundation through KKS Profile project SERT 2018/010 at BTH, Sweden.

REFERENCES

[1] Ö. Uludag, M. Kleehaus, C. Caprano, and F. Matthes, "Identifying and structuring challenges in large-scale agile development based on a structured literature review," in *2018 IEEE 22nd International Enterprise Distributed Object Computing Conference (EDOC)*. IEEE, 2018, pp. 191–197.

[2] B. Fitzgerald, K.-J. Stol, R. O'Sullivan, and D. O'Brien, "Scaling agile methods to regulated environments: An industry case study," in *2013 35th International Conference on Software Engineering (ICSE)*. IEEE, 2013, pp. 863–872.

[3] T. Dybå and T. Dingsøyr, "Empirical studies of agile software development: A systematic review," *Information and software technology*, vol. 50, no. 9-10, pp. 833–859, 2008.

[4] T. Dyba and T. Dingsoyr, "What do we know about agile software development?" *IEEE software*, vol. 26, no. 5, pp. 6–9, 2009.

[5] J. A. Livermore, "Factors that significantly impact the implementation of an agile software development methodology." *J. Softw.*, vol. 3, no. 4, pp. 31–36, 2008.

[6] S. C. Misra, V. Kumar, and U. Kumar, "Identifying some critical changes required in adopting agile practices in traditional software development projects," *International Journal of Quality & Reliability Management*, 2010.

[7] K. Rolland, T. Dingsoyr, B. Fitzgerald, and K.-J. Stol, "Problematizing agile in the large: alternative assumptions for large-scale agile development," in *39th International Conference on Information Systems*. Association for Information Systems (AIS), 2016, pp. 1–21.

[8] K. Conboy and N. Carroll, "Implementing large-scale agile frameworks: challenges and recommendations," *IEEE Software*, vol. 36, no. 2, pp. 44–50, 2019.

[9] H. Edison, X. Wang, and K. Conboy, "Comparing methods for large-scale agile software development: A systematic literature review," *IEEE Transactions on Software Engineering*, pp. 1–1, 2021.

[10] T. Sedano, P. Ralph, and C. Péraire, "Software development waste," in *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*. IEEE, 2017, pp. 130–140.

[11] N. B. Moe, B. H. Dahl, V. Stray, L. S. Karlsen, and S. Schjødt-Osmo, "Team autonomy in large-scale agile," in *Proceedings of the Annual Hawaii International Conference on System Sciences (HICSS)*. AIS Electronic Library, 2019, pp. 6997–7006.

[12] M. Cataldo, J. D. Herbsleb, and K. M. Carley, "Socio-technical congruence: a framework for assessing the impact of technical and work dependencies on software development productivity," in *Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement*, 2008, pp. 2–11.

[13] S. Betz, S. Fricker, A. Moss, W. Afzal, M. Svahnberg, C. Wohlin, J. Börstler, T. Gorschek *et al.*, "An evolutionary perspective on socio-technical congruence: The rubber band effect," in *2013 3rd International Workshop on Replication in Empirical Software Engineering Research*. IEEE, 2013, pp. 15–24.

[14] M. E. Conway, "How do committees invent," *Datamation*, vol. 14, no. 4, pp. 28–31, 1968.

[15] J. D. Herbsleb and R. E. Grinter, "Architectures, coordination, and distance: Conway's law and beyond," *IEEE software*, vol. 16, no. 5, pp. 63–70, 1999.

[16] M. Bass, V. Mikulovic, L. Bass, J. Herbsleb, and M. Cataldo, "Architectural misalignment: An experience report," in *2007 Working IEEE/IFIP Conference on Software Architecture (WICSA'07)*. IEEE, 2007, pp. 17–17.

[17] M. E. Conway, "How do committees invent," *Datamation*, vol. 14, pp. 28–31, 1968.

[18] J. M. Sierra, A. Vizcaino, M. Genero, and M. Piattini, "A systematic mapping study about socio-technical congruence," *Information and Software Technology*, vol. 94, pp. 111–129, 2018.

[19] W. Mauerer, M. Joblin, D. A. Tamburri, C. Paradis, R. Kazman, and S. Apel, "In search of socio-technical congruence: A large-scale longitudinal study," *arXiv preprint arXiv:2105.08198*, 2021.

[20] M. Kamola, "How to verify conway's law for open source projects," *IEEE Access*, vol. 7, pp. 38 469–38 480, 2019.

[21] D. L. Parnas, "On the criteria to be used in decomposing systems into modules," in *Pioneers and their contributions to software engineering*. Springer, 1972, pp. 479–498.

[22] T. Dingsøyr, N. B. Moe, T. E. Fægri, and E. A. Seim, "Exploring software development at the very large-scale: a revelatory case study and research agenda for agile method adaptation," *Empirical Software Engineering*, vol. 23, no. 1, pp. 490–520, 2018.

[23] N. B. Moe, D. Šmite, M. Paasivaara, and C. Lassenius, "Finding the sweet spot for organizational control and team autonomy in large-scale agile software development," *Empirical Software Engineering*, vol. 26, no. 5, pp. 1–41, 2021.

[24] J. H. Gundelsby, "Enabling autonomous teams in large-scale agile through architectural principles," in *Proceedings of the 19th International Conference on Agile Software Development: Companion*, 2018, pp. 1–4.

[25] T. Gustavsson, "Voices from the teams - impacts on autonomy in large-scale agile software development settings," in *Agile Processes in Software Engineering and Extreme Programming – Workshops*, R. Hoda, Ed. Cham: Springer International Publishing, 2019, pp. 29–36.

[26] S. Baškarada, V. Nguyen, and A. Koronios, "Architecting microservices: Practical opportunities and challenges," *Journal of Computer Information Systems*, vol. 60, pp. 428–436, 9 2020.

[27] S. Arandi, G. Michael, P. Evripidou, and C. Kyriacou, "Combining compile and run-time dependency resolution in data-driven multithreading," in *2011 First Workshop on Data-Flow Execution Models for Extreme Scale Computing*. IEEE, 2011, pp. 45–52.

[28] R. T. Fielding and R. N. Taylor, *Architectural Styles and the Design of Network-Based Software Architectures*. University of California, Irvine, 2000, aAI9980887.

[29] D. S. Cruzes and T. Dyba, "Recommended steps for thematic synthesis in software engineering," in *2011 international symposium on empirical software engineering and measurement*. IEEE, 2011, pp. 275–284.

[30] E. Zabardast, J. Gonzalez-Huerta, and B. Tanveer, "Ownership vs contribution: Investigating the alignment between ownership and contribution," in *2022 IEEE 19th International Conference on Software Architecture Companion (ICSA-C)*. IEEE, 2022, pp. 30–34.

[31] N. B. Ali, K. Petersen, and K. Schneider, "Flow-assisted value stream mapping in the early phases of large-scale software development," *J. Syst. Softw.*, vol. 111, pp. 213–227, 2016. [Online]. Available: https://doi.org/10.1016/j.jss.2015.10.013

[32] B. Tanveer, A. M. Vollmer, and U. M. Engel, "Utilizing change impact analysis for effort estimation in agile development," in *43rd Euromicro Conference on Software Engineering and Advanced Applications, SEAA 2017, Vienna, Austria, August 30 - Sept. 1, 2017*. IEEE Computer Society, 2017, pp. 430–434. [Online]. Available: https://doi.org/10.1109/SEAA.2017.64

[33] F. D. Davis, "Perceived usefulness, perceived ease of use, and user acceptance of information technology," *MIS Quarterly*, vol. 13, no. 3, pp. 319–340, 1989. [Online]. Available: http://www.jstor.org/stable/249008