Establishing a Pattern Language for the Organization of Distributed Software Development

Shakirullah Waseeb
Waheedullah Sulaiman Khail
Valentino Vranić
shakir.waseeb@gmail.com
wsulimankhail@gmail.com
vranic@stuba.sk
Institute of Informatics, Information Systems and Software Engineering, Faculty of Informatics and Information Technologies, Slovak University of Technology in Bratislava
Slovakia

ABSTRACT

Despite considerable efforts to address organizational problems of distributed software development, currently available solutions do not seem to be sufficient. They are fragmented into individual patterns either not forming coherent pattern languages to address organizational distributed software development or being incorporated into extensive pattern languages for organizing software development in general. Another problem is their disconnection from the current technological support for collaboration. We attempt at overcoming these problems by providing a set of six organizational patterns for distributed software development. We relate them to each other and to other known patterns and practices practically establishing a pattern language for the organization of distributed software development. The overall idea of how this pattern language can be used is presented using a pattern story of a real company.

CCS CONCEPTS

• **Software and its engineering → Patterns**.

KEYWORDS

organizational patterns, distributed software development, distributed teams, offshore/multi-site development

ACM Reference Format:
Shakirullah Waseeb, Waheedullah Sulaiman Khail, and Valentino Vranić. 2021. Establishing a Pattern Language for the Organization of Distributed Software Development. In *European Conference on Pattern Languages of Programs (EuroPLoP'21), July 7–11, 2021, Graz, Austria.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3489449.3489979

EuroPLoP'21, July 7–11, 2021, Graz, Austria

ACM ISBN 978-1-4503-8997-6/21/07…$15.00
https://doi.org/10.1145/3489449.3489979

1 INTRODUCTION

Software development is becoming significantly multi-site, globally distributed, and multicultural [23, 25, 37]. The software industry is seeking lower costs [29] access to competencies worldwide, reaching proximity to market and mobility in resources [23, 37]. Software development in distributed context imposes challenges in communication, control and coordination. Agile software development has a significant impact on the software industry because of its potential to improve communication and, therefore, reduce coordination and control overhead in software projects. However, merging agile in distributed software development raises communication problems and increases the overhead of control and coordination. Practices and documented experience from numerous conferences, such as EuroPLoP, ICGSE, Euromicro, HICSS, COMPSAC, ASPEC, Agile, XP, and EuroSPI, show the successful implementation of agile values and principles in different distributed projects [38].

In this work, we document the best practices of distributed software development we have observed in the form of patterns. As Buschman et al. [5] point out, the cliché pattern definition, which says that a pattern is a solution to a problem that arises within a specific context, does not help much. Therefore, Buschman et al. [5] are more specific, explaining that patterns provide working, concrete, and adaptable solutions to problems that arise again and again in certain situations during software development, from organizational to programming contexts. In this spirit, we provide the following definition for a pattern of distributed software development:

> A pattern of distributed software development describes a recurring software development problem that arises due to dispersion and presents a well-proven generic approach for its solution.

The rest of the paper is structured as follows. Section 2 provides an insight into the organizational problems of distributed software development and what solutions to them are currently available. Section 3 brings the story behind the patterns of distributed software development we discovered and explains the format we used to present them. Sections 4–9 presents these patterns and relate them to each other and to other known patterns and practices practically establishing a pattern language for the organization of distributed software development. Section 10 draws conclusions and outlines further work.

2 DISTRIBUTED SOFTWARE DEVELOPMENT: PROBLEMS AND SOLUTIONS

Software development team members might be co-located, remote, and even distributed. Organizations are expanding their business by making use of the opportunities in different locations. Having all team members co-located is implausible due to the availability of local experts, business opportunities at the local market, the way of working, etc., causing a part of the team to be physically located in different places.

Experts with proven expertise and domain knowledge are sometimes challenging to be available in the local market due to high demand or sometimes not feasible financially. Reaching proximity to the market requires companies to expand their business and services into different locations. There might be a significantly high moving cost to have a co-located team by shifting locations of people. Remote working team members are more isolated rather than being co-located.

With advancements in network and communication technologies, remote collaboration became more robust and perceived as more natural. This brought significant opportunities for developing software in distributed settings. Distributed software development teams enable organizations to access highly specialized expertise across geographic locations [30]. Software development using distributed teams is being a subject of research for decades [8, 20, 25, 29, 37, 38, 42].

Global software development, distributed development teams, and offshoring are being the subject of research for more than a decade [8, 20, 25, 29, 36–38, 42]. However, few attempts are being taken to document them in the pattern format. Berczuk [3] presents a pattern language with four patterns named *Loose Interfaces*, *Parser/Builder*, *Hierarchy of Factories*, and *Handlers*, which are intended for developing ground software for satellite telemetry system with distributed teams. They focus on how to take organization into account in the architecture. Also, van Heesch [43] reported two collaboration patterns named *Experience Mix* and *Dissolve Geographical Boundaries*) for software projects with offshore contributions, which he observed in industrial practice and in the literature.

Sutherland et al. [42] analyze and recommended best practices for globally distributed agile teams. The two companies SirsiDynix and StarSoft situation of distributed Scrums is given as the *context*. Five complexity drivers (problem, solution, technical, compliance, and team) and six top issues (communication, cultural, strategies, project and process management, culture, technical, and security) are listed as *forces*. Integrated Scrums are given as a *solution*, which is further divided into: team formation, Scrum meetings, sprints, product specifications, testing, configuration management, XP practices, and measuring progress.

Coplien and Harrison [12] in their patterns book also cover the problem of distributed teams by relevant patterns such as *Organization Follows Locations*, *Organization Follows Market*, *Stand Linking Location*, *Standards Linking Location*, or *Face-to-Face Before Working Remotely*.

Sutherland et al. [41] in their Scrum Book provide a collection of patterns that describe the elements of the Scrum framework. This builds and expands on Coplien and Harrison's organizational patterns book [12]. For example, the *Scrum Team* pattern comprises both the *Collocated Team* and a *Cross-Functional Team* patterns, since a Scrum team operates as a small organization and is independent in making decisions in responding to stakeholders and the market [41].

It is not unusual to consider a pattern composition as a pattern [40].

Hvatum [26] provides a collection of twenty individual patterns—i.e., not connected into a pattern language—for distributed teams addressing human interaction with respect to personal, team, project, and company/organization issues.

Cavrak et al. [10] identified a set of collaboration patterns by analysing collaboration links within distributed students that can assist teachers in understanding dynamics in distributed projects.

3 THE PATTERNS

In this paper, we documented six patterns (highlighted blue in Figure 1) of distributed software development teams. We have observed these patterns in practice during the last three years while working in distributed software development teams. Iterative and creative approaches [18, 19, 31, 45] were being taken into account in writing and documenting these patterns. We used document analysis, interviewing, and expert collaboration to collect the most relevant and high-quality information, as is known in the literature [8, 20, 24, 25, 29, 30, 37, 38, 42].

3.1 The Story Behind the Patterns

Before presenting the patterns, let's explore them in a real world setting. We'll tell you a story of a company called TFEService.Inc, an international company with its headquarters in the USA. The name is fake (for privacy reasons), but everything else is real. In the story, we refer to the corresponding organizational patterns in parentheses.

Reaching for the proximity to the market, offshore development, and having high operational efficiency, the company has teams in UAE, India, Afghanistan, Jordan, and other countries (*Distributed Development Team*, Section 4). Following the opportunities in the market, TFEService.Inc ran a project in Kabul, Afghanistan (*Organization Follows Market* [12]). The goal of the project was to design and implement a system that collects and computes data at large scale from different mobile network operators in Afghanistan.

The company already had development a task force having experience of developing such data stream pipe-lining and distributed computing systems. Now, the company was required to build a distributed development team by assigning existing staff and hiring new staff to implement the project (*Distributed Development Team*, Section 4).

Initially, the company established two additional roles at the customer site: the project manager and assistant project manager (*Size the Organization* [12]). They were responsible for the coordination between the customer, development team, business team, and headquarters (*Distributed Team Configuration*, Section 5, and *Communication Bridge*, Section 6). The headquarters also had a regional coordinator for handling cultural and language communication (*Communication Bridge*, Section 6). In order to understand the mobile network operator systems, the company hired (*Phasing*

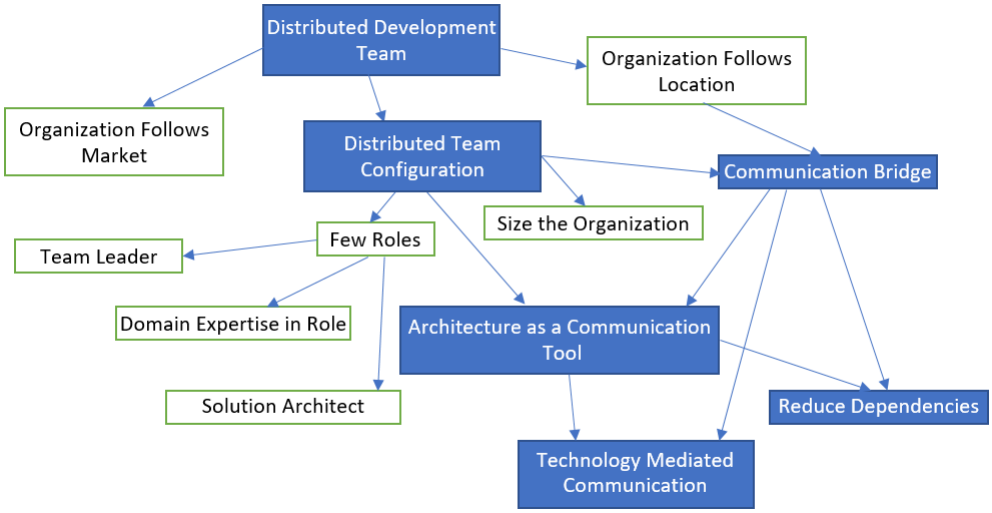

Figure 1: Organizational patterns of distributed software development.

It In [12]) telecommunication experts (*Domain Expertise in Role* [12]) who were working with mobile network operators and had sufficient domain knowledge.

Architecture is at the core of every system. Therefore, the solution architect traveled from India to Kabul for to closely analyze the situation (*Face-to-Face Before Working Remotely* [12]). He worked together with his local team and customer on-site to understand the requirements and develop a solid architectural solution. As the development team had competencies in the technology stack, the only issue was understanding the mobile network operator systems, which was streamlined by experts from mobile network operators in their local teams.

Once the software architecture was finalized, the development team began its implementation. The solution architect followed the Design by Contract approach [32], where the system is viewed as a set of communicating components whose interaction is based on the specification of the mutual obligations called contracts (*Architecture as Team Coordination Tool*, Section 7). The teams were structured and configured according to the system's architecture (*Conway's Law* [12, 41]). For easier management and lower communication overhead, the teams were partitioned based on technology and architecture.

There were three teams: front-end (UX) developers, back-end developers, and DevOps. The teams were configured in a two-two-one fashion: two senior, two junior, and one leader (*Distributed Team Configuration*, Section 5)). The Design by Contract approach is observed to cut down the communication and system integration overhead between teams (*Reduce Dependencies*, Section 8). Design by Contract approach has helped assigning responsibilities of each design and architect entity to different teams. This supported independent development of different components while guaranteeing smooth integration. This approach is a proven technical counterpart of the bindings between parties involved in distributed and concurrent development of a system [2].

The solution architect traveled several times to the customer's site to ensure the deliverable meets the requirements. The company established a well structured corporate technological communication channels (*Technology Mediated Communication*, Section 9). Skype and WhatsApp were used for instant, spontaneous, and other ad hoc communication. E-mails were used for formal communication.

Microsoft Teams was used for meetings, discussions, file sharing, notifications, etc. Through Microsoft Teams, the company successfully built trust between members by sharing their profile details and giving them an equal chance to participate in virtual meetings. Several technology means were used to enable team members to communicate and better know each other.

3.2 Pattern Format

We expressed the patterns (Sections 4–9) in Coplien and Harrison's pattern format [12] with the conflict of the the most prominent contradicting forces expressed in the *but* form proposed by Vranić and Vranić [44]. This is the format:

<Pattern Name>

… – The context in which the pattern occurs.

✣✣✣ – The text in bold describes the actual problem as a conflict of the two most prominent contradicting forces.

Therefore – The solution.

✣✣✣ – An optional part with resulting consequences upon applying the given pattern.

Description optional description to explain the pattern.

4 DISTRIBUTED DEVELOPMENT TEAM

. . . due to various reasons such as physical distance, business settings, and business requirements, it is neither feasible nor efficient to continue with co-located teams.

✣✣✣

Co-located teams are efficient and effective, but sometimes barriers, such as the physical location of team members, prevent the team from being co-located. In addition, sometimes business opportunities or availability of experts step up part of the team to be physically located in different places.

Developing complex software requires experts with proven experience and domain knowledge, but creating a team of experts with such competencies is challenging. These challenges often include a lack of experts in the local market or the local experts in very high demand, which might not be feasible financially.

Co-located teams are very effective and productive, but limited resources, tight budgets, and time constraints motivate the organizations to have multi-site software development.

With a co-located team, we have synchronized but limited working hours. A distributed team can benefit from round-the-clock development, but synchronizing the communication and collaboration of different sites is difficult to manage.

Therefore:

Create a distributed team where team members are physically in different locations but still work towards a common goal.

Establish a distributed team with sites at different locations that reach market proximity and access a large pool of experts. This should be based on business opportunities, the availability of experts and resources, and the feasibility of round-the-clock development in different time zones (see Figure 2).

Build sites where business opportunities are at maximum (see the *Organization Follows Market* and *Organization Follows Location* patterns [12]). Onboard high skilled and competence experts (see these patterns: *Distributed Team Configuration*, Section 5, *Domain Expertise In Roles* [12], *Size the Organization* [12], and *Few Roles* [12]). Choose locations where round-the-clock development is feasible and prepare a proper plan for managing works dependencies between teams in various time zones. Ensure that team members collaboration is synchronized as much as possible by bridging the communication gaps (see the *Communication Bridge* pattern, Section 6).

✣✣✣

Communication is essential for team collaboration and coordination. Distributed development of software fraught consequences due to the temporal, spatial, cultural, and configurational dispersion that are not experienced in traditional systems development [14, 22, 25]. Subsequently, affecting the communication, coordination and control of the distributed team [1]. Temporal distance is a measure of the dislocation in time experienced by two actors wishing to interact; geographical distance is a measure of the effort required for one actor to visit another, and sociocultural distance is a measure of an actor's understanding of another actor's values and normative practices [1].

Cross-organizational and cross-national collaboration is very common in the marketplace. Organizations are having projects across the state, country, and even across the globe. A project might be distributed over more than one site. Organizations struggle to reach market proximity, expand through acquisitions, cut costs, work round-the-clock, increase operational efficiency, and many others. Such attainments would be impossible by staying on-site.

5 DISTRIBUTED TEAM CONFIGURATION

. . . when a distributed development team is determined, it is time to take care of its setup.

✣✣✣

A distributed team opens up plenty of opportunities, but configuring and building such a team brings various challenges.

Too many sites will expand the market coverage but create coordination complexity. As the number of sites increases, so does the technical and social complexity of coordination, interaction, and communication. Similarly, members at minority sites (few sites in neighboring) may feel more isolated and thus face more significant communication challenges.

In a distributed team, people from different ethnic, professional, technical, functional, and organizational backgrounds have to cooperate. This may result in inter-group relations problems that may engender less trust and more conflicts (teamness within the team).

Similarly, the team dispersed across different time zones makes synchronous collaboration difficult and reduces real-time problem-solving.

Therefore:

Begin with a small team. Grow it smartly, build trust, and organize sites independent of spatial and temporal distances.

Begin with few business sites in central locations where business opportunities are at maximum. Slowly expand the business sites in the neighboring based on the experience of existing sites. Choose development sites where development task force availability is high and are close to the client site (see near-shoring).

Arrange the team members across sites independent of spatial and temporal distances among them.

Some potential arrangements of members across different sites include; multiple sites without isolation (e.g., three sites with 4-3-3 and 2-3-3), balanced isolation (e.g., 2-2-2, 3-3-3), and highly isolated (e.g., 3-1-1, and 1-2-1). The degree of isolation decreases other team members' awareness of their activities.

Keep the teams as small as possible or break existing teams into small subteams. Use the *Small Teams* pattern [41] of people working on serialized work rather than striving for false parallelism [41]. Try to keep the number of roles low (consider the *Few Roles* pattern [12]) by identifying the value of various roles. Have a team leader and a combination of senior and junior members in subteams (consider the *Experience Mix* pattern [43]). Make sure the appropriate expertise and the ability to work together can lead to a cross-functional team (see the *Cross-Functional Team* pattern [41]).

Also, get to know that crossing multiple boundaries does not have to impact the team negatively. At each site, have people who

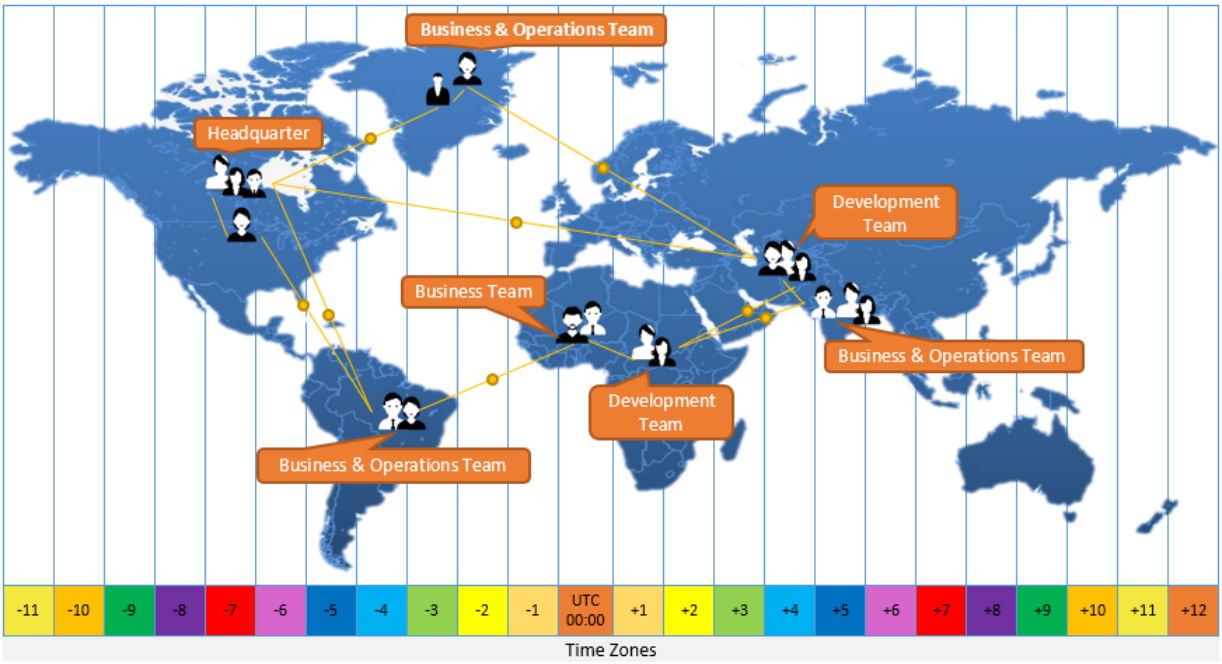

Figure 2: Teams distributed across different locations and time zones.

understand and communicate in at least one common language and have previous working experience in a distributed team.

Time zones are based on east-west spatial distances as can be seen in Figure 2. If the teams are dispersed in the east-west directions, the number of work hours the team members have to communicate synchronously may be limited. For example, the Northeastern United States and South Asia are nine time zones apart. Therefore, select sites with few overlapping work hours and follow the time-shifting-around practice [7] to adjust the working hours for synchronous communication.

Make sure that means of both synchronous and asynchronous communications are established and also consider coordination mechanisms during the team configuration to overcome communication challenges (see the *Communication Bridge* pattern, Section 9).

6 COMMUNICATION BRIDGE

…the distributed team is established, and members are dispersed spatially, temporally, or even culturally in different sites. Control, coordination, and collaboration require strong communication between sites.

✣✣✣

Proximity in co-located team leads to effective spontaneous communication, but the spatial, temporal, cultural, and configurational dispersion in a distributed team makes communication difficult.

The co-located team can have Daily Stand-up Meetings or other spontaneous communications, but this could not be possible in a distributed team due to physical distances between team members. The geographic distances among team members reduce spontaneous communication and therefore affect the frequency and ease of face-to-face meetings. On-site members share the same time zone, but off-sites may be located in different time zones, which may become burdensome the synchronous communication. Sites with a minority of the team members may feel out of the loop and produces more significant communication challenges.

Therefore:

Sort out technology mediums for synchronous and asynchronous interactions and practice coordination mechanisms to counteract the communication challenges raised due to team dispersion.

Make use of conferencing and instant messenger software for synchronous and e-mails for asynchronous communication.

Tools such as Microsoft Teams, Gather, Discord, Muraly, Miro, Slack, Jitsi, Google Meet, Zoom, Skype, WhatsApp, etc., are considered valuable to work together remotely. Regular use of these means reduces the communication overhead (see the *Technology Mediated Communication* pattern for more details, Section 9).

A product-centric approach; i.e. organize the team around a product and tend to be geographically local [4] to reduce communication between team members at distance (see the *Architecture as Team Coordination Tool*, Section 7, and *Reduce Dependencies*, Section 8, patterns). Mechanisms such as direct supervision, standardization (see *Standards Linking Locations* pattern [12]), and software architecture (see the *Conway's Law* pattern [12, 41]) can be used which requires minimal communication between teams [4].

Management by Time Shifting Around; stay in place, but time-shift to different locations by adjusting or scattering their workday,

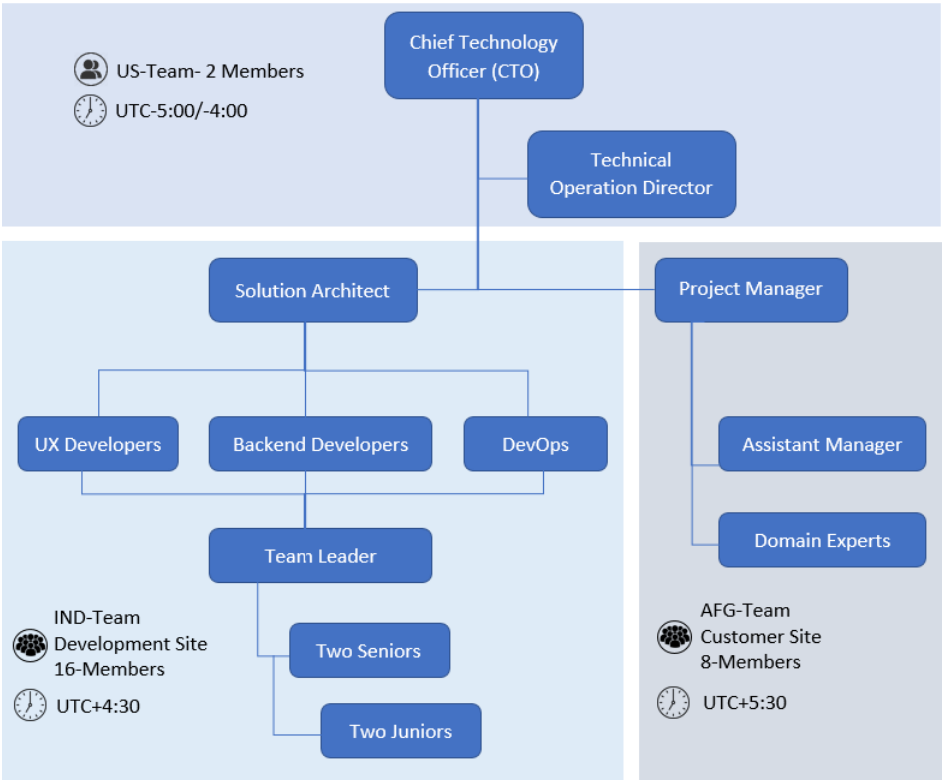

Figure 3: Configuration and dispersion of teams.

i.e., adjusting one's work hours to accommodate another's schedule. This will increase the chances of synchronous communication between sites in different time zones.

If feasible, begin with a face-to-face meeting to know those who work together and establish project unity (see the *Face-to-Face Before Working Remotely* pattern [12]).

Spatial dispersion (miles separating members) poses the challenge of frequent face-to-face communication and has the highest impact on spontaneous communication.

Communication issues because of different time zones (e.g., interaction of teams from the US with teams from Asia with a 12.5 hour time difference). On the other hand, cultural differences result in miscommunication. Software architecture also poses challenges in the case of distributed software development teams. Teams configuration in terms of the number of sites, members, members separated from their leader and isolated from each other. Developing trust between remote team members also challenges communication. Communication has always turned up as the most important and most challenging in distributed software development.

7 ARCHITECTURE AS TEAM COORDINATION TOOL

…the software architecture mirrors the structure of the organization; while the team is distributed, the architecture can be used as a means of coordination between sites.

A distributed software development team needs to coordinate its activities very closely, but the distance between its members prevents achieving sufficiently intense communication necessary for this.

Coordination in software development means coordinating the development activities [6, 8, 27], but in a multi-site context, this is not enough. More efforts should be put into coordinating interdependencies between the activities to achieve a common goal [21]. Communication is a prominent part of coordination. However, team dispersion affects communication frequency and subsequently affects coordination.

Therefore:

Use software architecture as a coordination mechanism that describes the coordination of the activities and their interdependencies in terms of components and their relationships.

Spatially dispersed teams can develop software when the software architecture is defined such that its components are independent of each other [34].

Consider coordination as the coordination of dependencies between activities (management of inter-dependencies between activities) toward a common goal [35].

The primary mechanism for coordination is the software architecture, which describes the activities and interdependencies in terms of components and their relationships [17]. From activities

(and subsystem) toward system-level dependencies requires software architects and developers to have a common understanding of the software architecture [35]. Minimize the communication barriers between the teams by an architect who acts as boundary spanner between teams [13] (see *Architect Controls Product* pattern [12]), translating customer needs into terms understood by software developers. In their pattern named *Architecture, Plan, and Processes*, Herbsleb and Grinter observed as a vital coordination mechanisms in distributed software projects [21]. Use software design approaches such as Design by Contracts [2], Domain Driven Design [16], Layered Architecture [15], and others which streamline the communication and coordination between the team members working at different sites in a standard way.

The relationship between coordination and software architecture was first coined by Melvin Conway [11]. The software architecture influences the communication requirements between project members. Companies are driven to distribute development resources around the globe for marketing purposes. Because of acquisitions, cost considerations, and the availability of needed expertise, geographic distribution challenges coordination mechanisms and informal communication by requiring robust across distances.

8 REDUCE DEPENDENCIES

…the passing of knowledge, work, and resources between people, or between sites needs to be coordinated and their dependencies should be managed appropriately.

Dependencies are inevitable for the division of labor, but they come with a huge overhead in distributed settings.

Dependencies can disrupt the progress of a software project if they constrain the flow of work or an increment of value. For example a person or site cannot make progress until another person or site finishes its work or otherwise solves a problem. Sometimes dependencies result from poor or highly complex organizational structure, but sometimes they are essential for collaboration, risk management, or even technical reasons. Distributed collaboration demands that there be some dependencies even in some cases collaboration will have highly dependent tasks.

The technical or task information (expertise) is known only by a particular person or group but not immediately available in a given time and space.

Likewise, there is a situation where who is doing what and when is not known. These all lead to knowledge dependencies between sites where some information is required to progress in a project.

There is a situation when an activity cannot proceed until another activity is completed or an existing business process causes activities to be carried out in a certain order.

Shared resources are available (a person, place, or thing) when two tasks require the same resource for completion. For example, a software component must interact with other software components, while its presence or absence affects the progress.

Strode and Huff's [39] developed a taxonomy of agile software development dependencies and provided three different categories of dependencies: task, resources, and knowledge dependencies. This taxonomy can consider dependencies that could hinder the projects and to make appropriate and timely mitigating actions.

Therefore:

Use strategies such self-serve capability, systematic swarming, formal engagement models, or dependency management, to mitigate and reduce dependencies. Manage the flow of work and realign people and sites around value streams.

Develop self-serve capability within the team to remove the dependencies. For example, if a site is dependent upon a particular person's skills, then invest time and effort into developing those skills within a given site.

Use a role that handle any information to be shared within the team and also proactively working on knowledge sharing across all the sites (see the *Remote Connector* pattern [26]).

Use systematic swarming by moving people with the right skills (subject matter expertise) between delivery sites to deliver the dependent requirements.

Suppose a team is dependent upon an API or component from another site. In that case, fake objects, stub, or mock enable them to continue development instead of waiting for them to become available.

Draft the engagement models with service level agreements (SLAs) around common request types so that consumers can reason about the deliverable.

Have proper dependency management to recognize, anticipate, and manage dependencies between tasks, people, processes, and systems. This will help in reducing process variability, subsequently increasing predictability. Use dependency mapping and visualization to understand the dependencies better.

Come up with a team structure where tasks are substantially independent between different sites. In such cases, the interfaces between the sites are well architected so that the dependencies are minimized.

Have a team leader and a combination of senior and junior members in subteams (see the *Experience Mix* pattern [43]). Make sure the appropriate expertise and the ability of working together that can lead to a cross-functional team (see the *Cross-Functional Team* pattern [41]). Reduce dependencies using a number of organizational designs for distributing tasks across distant sites (see the *Feature Assignment* and *Loose Interfaces* patterns [12]).

Keep work that requires similar functional expertise in one place [9]. A team could be structured around products (e.g., following the component architecture, where each site works on its own components) [9]. Inevitably though, reducing dependencies too much would cause the team size to increase, which might make the specialist skills dispersed to many sites.

Mechanisms such as direct supervision, standardization (see the *Standards Linking Locations* pattern [12]), and software architecture (see the *Conway's Law* pattern [12, 41]) can be used which requires minimal communication between teams [4].

9 TECHNOLOGY MEDIATED COMMUNICATION

...sites are dispersed and for some reason rarely meet in person consequently minimizing the chances of face-to-face communication and coordination.

Close and natural communication is necessary to get work done, but (physical) face-to-face communication is not possible in distributed settings.

Co-located teams naturally and informally share knowledge and information, thereby supporting each other, but geographical distances between teams obstructs natural communication.

Therefore:

Use technology-supported communication by employing rich communication means for messaging, conferencing, interaction, and collaboration. Establish technology-mediated communication comprises a series of interaction events. Sequence these events to frame the regular face-to-face events using various media.

Arrange conferencing, instant messaging, and e-mail software for synchronous and asynchronous communication. Tools such as Microsoft Teams, Discord, Slack, Google Meet, Zoom, Skype, WhatsApp, etc., are considered useful to support remote collaboration. Regular use of these means reduces communication overhead.

Use interactive and collaborative surface based systems, such as DigiMetaplan [28] and Domino [33], that facilitate simultaneous co-located and remote collaboration with both synchronous and asynchronous work.

Use version control and team collaboration tools such as GitHub, BitBucket, Team Service Foundation, Jira, etc., to support code development.

Tools such as Kanban and Trello can be used for visual project management. These tools support real-time team collaboration by sharing tasks, information, and comments anywhere and anytime.

10 CONCLUSIONS AND FURTHER WORK

Despite large efforts to address organizational problems of distributed software development, currently available solutions do not seem to be sufficient. They are fragmented into individual patterns either not forming coherent pattern languages to address organizational distributed software development or being incorporated into extensive pattern languages for organizing software development in general. Another problem is their disconnection from the current technological support for collaboration.

We attempt at overcoming these problems by providing a set of six organizational patterns for distributed software development. We relate them to each other and to other known patterns and practices practically establishing a pattern language for the organization of distributed software development. The overall idea of how this pattern language can be used is presented using a pattern story of a real company.

Further research is required in real distributed software development settings to uncover further patterns and connections between them.

ACKNOWLEDGMENTS

We would like to thank Veli-Pekka Eloranta for being our shepherd and for his constructive remarks. Our sincere thanks also go to our writer's workshop group members: Michael Weiss, Christian Kohls, Eduardo Guerra, Stefan Holtel, Niels Seidel, Luciana A.M. Zaina, and Sabine Varetza-Pekarz.

The work reported here was supported by the Scientific Grant Agency of Slovak Republic (VEGA) under grant No. VG 1/0759/19 and by the Operational Programme Integrated Infrastructure for the project Research of Effective Methods for the Development of Adaptive Software Ecosystems (EMEVYS, ITMS: 313012S803), co-funded by the European Regional Development Fund (ERDF).

REFERENCES

[1] Par J. Agerfalk, Brian Fitzgerald, Helena Holmstrom Olsson, Brian Lings, Bjorn Lundell, and Eoin Ó Conchúir. 2005. A Framework for Considering Opportunities and Threats in Distributed Software Development. In *Proceedings of the International Workshop on Distributed Software Development, DiSD 2005.* Austrian Computer Society.

[2] Albert Benveniste, Benoît Caillaud, Dejan Nickovic, Roberto Passerone, Jean-Baptiste Raclet, Philipp Reinkemeier, Alberto Sangiovanni-Vincentelli, Werner Damm, Thomas Henzinger, and Kim G. Larsen. 2018. *Contracts for System Design (Foundations and Trends(r) in Electronic Design Automation).* Now Publishers.

[3] Stephen P Berczuk. 1996. Organizational Multiplexing: Patterns for Processing Satellite Telemetry with Distributed Teams. *Pattern Languages of Program Design* 2 (1996), 193–206.

[4] Jan Bosch and Petra Bosch-Sijtsema. 2010. Coordination Between Global Agile Teams: From Process to Architecture. In *Agility Across Time and Space.* Springer, 217–233.

[5] Frank Buschmann, Kelvin Henney, and Douglas Schimdt. 2007. *Pattern-Oriented Software Architecture: On Patterns and Pattern Language.* Vol. 5. Wiley.

[6] Erran Carmel. 1999. *Global Software Teams: Collaborating Across Borders and Time Zones.* Prentice Hall.

[7] Erran Carmel. 2010. MBTA: Management By Timeshifting Around. In *Agility Across Time and Space.* Springer, 167–170.

[8] Erran Carmel and Ritu Agarwal. 2001. Tactical approaches for alleviating distance in global software development. *IEEE software* 18, 2 (2001), 22–29.

[9] Erran Carmel and Paul Tjia. 2005. *Offshoring Information Technology: Sourcing and Outsourcing to a Global Workforce.* Cambridge University Press.

[10] Igor Čavrak, Marin Orlić, and Ivica Crnković. 2012. Collaboration Patterns in Distributed Software Development Projects. In *34th International Conference on Software Engineering, ICSE 2012.* IEEE, 1235–1244.

[11] Melvin E Conway. 1968. How do Committees Invent. *Datamation* 14, 4 (1968), 28–31.

[12] James O. Coplien and Neil B. Harrison. 2004. *Organizational Patterns of Agile Software Development.* Prentice-Hall.

[13] Bill Curtis, Herb Krasner, and Neil Iscoe. 1988. A Field Study of the Software Design Process for Large Systems. *Commun. ACM* 31, 11 (1988), 1268–1287.

[14] Daniela Damian, Filippo Lanubile, and Heather L Oppenheimer. 2003. Addressing the Challenges of Software Industry Globalization: The Workshop on Global Software Development. In *Proceedings of the 25th International Conference on Software Engineering.*

[15] Florian Echtler and Gudrun Klinker. 2008. A Multitouch Software Architecture. In *Proceedings of the 5th Nordic Conference on Human-Computer Interaction: Building Bridges.*

[16] Eric Evans and Eric J Evans. 2004. *Domain-Driven Design: Tackling Complexity in the Heart of Software.* Addison-Wesley.

[17] David Garlan and Dewayne E. Perry. 1995. Introduction to the Special Issue on Software Architecture. *IEEE Transactions on Software Engineering* 21, 4 (1995), 269–274.

[18] Neil B Harrison. 1999. The Language of Shepherding. *Pattern Languages of Program Design* 5 (1999), 507–530.

[19] Neil B. Harrison. 2006. Advanced Pattern Writing Patterns for Experienced Pattern Authors. Avaya inc..

[20] James D Herbsleb. 2007. Global Software Engineering: The Future of Socio-Technical Coordination. In *Proceedings of the IEEE Conference on Future of Software Engineering, FOSE'07.* IEEE.

[21] James D Herbsleb and Rebecca E Grinter. 1999. Architectures, Coordination, and Distance: Conway's Law and Beyond. *IEEE software* 16, 5 (1999), 63–70.

[22] James D. Herbsleb and Audris Mockus. 2003. An Empirical Study of Speed and Communication in Globally Distributed Software Development. *IEEE Transactions on Software Engineering* 29, 6 (2003), 481–494.

[23] James D Herbsleb and Deependra Moitra. 2001. Global Software Development. *IEEE software* 18, 2 (2001), 16–20.

[24] Helena Holmstrom, Eoin Ó Conchúir, J Agerfalk, and Brian Fitzgerald. 2006. Global Software Development Challenges: A Case Study on Temporal, Geographical and Socio-Cultural Distance. In *Proceedings of the 2006 IEEE International Conference on Global Software Engineering (ICGSE'06)*. IEEE, 3–11.

[25] Helena Holmström, Brian Fitzgerald, Pär J Ågerfalk, Eoin Ó Conchúir, et al. 2006. Agile Practices Reduce distance in Global Software Development. *Information Systems Management* 23, 3 (2006), 7–18.

[26] Lise B. Hvatum. 2020. Patterns for Distributed Teams. In *Proceedings of the 25th Conference on Pattern Languages of Programs, PLoP 2020*.

[27] Robert E Kraut and Lynn A Streeter. 1995. Coordination in Software Development. *Commun. ACM* 38, 3 (1995), 69–82.

[28] Khanh-Duy Le, Paweł W Woźniak, Ali Alavi, Morten Fjeld, and Andreas Kunz. 2019. DigiMetaplan: Supporting Facilitated Brainstorming for Distributed Business Teams. In *Proceedings of the 18th International Conference on Mobile and Ubiquitous Multimedia*.

[29] Ian Marriott. 2010. Gartner's 30 Leading Locations for Offshore Services, 2010–2011. *Stamford, CT: Gartner* (2010).

[30] Likoebe M Maruping. 2010. Implementing Extreme Programming in Distributed Software Project Teams: Strategies and Challenges. In *Agility Across Time and Space*. Springer, 11–30.

[31] Gerard Meszaros and Jim Doble. 1997. *A Pattern Language for Pattern Writing*. Addison-Wesley, 529–574.

[32] Bertrand Meyer. 1997. *Object-Oriented Software Construction* (second ed.). Prentice Hall.

[33] Thomas Neumayr, Hans-Christian Jetter, Mirjam Augstein, Judith Friedl, and Thomas Luger. 2018. Domino: A Descriptive Framework for Hybrid Collaboration and Coupling Styles in Partially Distributed Teams. In *Proceedings of the ACM on Human-Computer Interaction*, Vol. 2. ACM, 1–24.

[34] Judith S Olson and Stephanie Teasley. 1996. Groupware in the Wild: Lessons Learned from a Year of Virtual Collocation. In *Proceedings of the 1996 ACM Conference on Computer Supported Cooperative Work*.

[35] Päivi Ovaska, Matti Rossi, and Pentti Marttiin. 2003. Architecture as a Coordination Tool in Multi-Site Software Development. *Software Process: Improvement and Practice* 8, 4 (2003), 233–247.

[36] Rafael Prikladnicki, Jorge Luis Nicolas Audy, and Forrest Shull. 2010. Patterns in Effective Distributed Software Development. *IEEE Software* 27, 2 (2010), 12–15.

[37] Rafael Prikladnicki, Jorge Luis Nicolas Audy, and Roberto Evaristo. 2003. Global Software Development in Practice Lessons Learned. *Software Process: Improvement and Practice* 8, 4 (2003), 267–281.

[38] Darja Šmite, Nils Brede Moe, and Pär J Ågerfalk. 2010. *Agility Across Time and Space: Implementing Agile Methods in Global Software Projects*. Springer Science & Business Media.

[39] Diane E. Strode and Sid L. Huff. 2012. A Taxonomy of Dependencies in Agile Software Development. *Information Systems Frontiers* (2012), 23–46.

[40] Waheedullah Sulaiman Khail and Valentino Vranić. 2017. Treating Pattern Sublanguages As Patterns with an Application to Organizational Patterns. In *Proceedings of the 22nd European Conference on Pattern Languages of Programs, EuroPLoP '17*. ACM, Irsee, Germany.

[41] Jeff Sutherland and James O. Coplien. 2019. *A Scrum Book: The Spirit of the Game*. Pragmatic Bookshelf.

[42] Jeff Sutherland, Anton Viktorov, Jack Blount, and Nikolai Puntikov. 2007. Distributed Scrum: Agile Project Management with Outsourced Development Teams. In *Proceedings of the 40th Annual Hawaii International Conference on System Sciences, HICSS'07*. IEEE.

[43] Uwe van Heesch. 2015. Collaboration Patterns for Offshore Software Development. In *Proceedings of the 20th European Conference on Pattern Languages of Programs, EuroPLoP 2015*. Irsee, Germany.

[44] Valentino Vranić and Aleksandra Vranić. 2019. Drama Patterns: Extracting and Reusing the Essence of Drama. In *Proceedings of the 24th European Conference on Pattern Languages of Programs, EuroPLoP 2019*. ACM, Irsee, Germany.

[45] Tim Wellhausen and Andreas Fießer. 2011. How to Write a Pattern? A Rough Guide for First-Time Pattern Authors. In *Proceedings of the 16th European Conference on Pattern Languages of Programs, EuroPLoP 2011*. Irsee, Germany.