On the negative impact of team independence in microservices software development

Valentina Lenarduzzi
Tampere University of Technology, Finland
valentina.lenarduzzi@tut.fi

Outi Sievi-Korte
Tampere University of Technology, Finland
outi.sievi-korte@tut.fi

ABSTRACT

Microservices allow teams to work in isolation, and reduce the need of communication among the teams. While this aspect could be considered a short-term benefit due to the reduction of the communication effort, the reduced amount of communication could create issues in the general synchronization of the whole project. In this work we start to depict the potential negative impact of the reduced need of communication in the development of microservices-based systems. We propose the design of an empirical study to understand if this reduction is beneficial, and when and how communication between teams should be enforced.

ACM Reference Format:
Valentina Lenarduzzi and Outi Sievi-Korte. 2018. On the negative impact of team independence in microservices software development. In *XP '18 Companion: XP '18 Companion, May 21–25, 2018, Porto, Portugal.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3234152.3234191

1 INTRODUCTION

Microservices (MSs) are relatively small and autonomous services deployed independently, with a single and clear purpose [1]. Each MS is independent from other services, can be developed in different programming languages, can scale independently from other services, and can be deployed on the hardware that best suits its needs. Moreover, because of the limited size, MS are easier to maintain and more fault-tolerant since the failure of one service will not break the whole system, which could happen in a monolithic system.

MSs enable large projects to be splitted into smaller MSs. Individual MSs are mainly independent from each other. Each team works on different MS, and develops user stories that affect only their own MS. Thus, when a team encounters some issues, they do not influence or slow down the other teams. Therefore, communication between developers is commonly very efficient, since there are no barriers and no need to wait for other teams to take decisions[21][2].

In this work we aim at discussing communication issues among teams developing MS-based systems and to propose the design of a collaborative set of case studies to understand if the reduced communication need is beneficial or not. The goal is to raise some discussion points on the negative effect of reduced communication in Agile Software Development (ASD) in the context of MS software development. Moreover, we aim at discussing the possible design of a research study, possibly collaborative between workshop participants, with the goal of identifying effective communication patterns and solutions to overcome communication barriers [4].

This paper is structured as follows. Section 2 introduces the background and related work. In Section 3, we introduce the communication issues in MS development. In Section 4 we describe the study protocol we are aiming to adopt. In Section 5 we present our roadmap while in Section 6, we draw conclusions and present an outlook on future work.

2 BACKGROUND AND RELATED WORKS

Communicating and constantly sharing information among the teams and customers are some of the most important activities during ASD [6], [7].

Communication plays a strategic role from the beginning of the project, when teams and customers defined the Minimum Viable Product (MVP) [8]. Communication in Agile can be classified in different ways based on the actors involved and channels used. We can distinguish between internal and external communication, if it involves only developers (internal) or developers and stakeholders (external) [6]. Communication can also be classified into three different levels, as suggested by [9], one level (primary) between customers and team members to elicit the requirements, a second level (mid-iteration) between team members and sometimes customers in order to reduce any possible requirements disagreement, and the last one (end iteration) among team members to provide feedbacks and requirements details. This approach is useful especially during the requirement elicitation process [10].

Considering the adopted communication channels, we can classify communication as active, based on synchronous activities, or passive, based on asynchronous activities. Synchronous activities involve face-to-face meetings, telephone conversations, and video conferencing, while asynchronous include mail or documentation [9].

Several studies have investigated communication aspects in Agile processes [11], [12], [13], [14],[23] focusing on key success factors [7], [16]. One of the most important findings is that selecting good communication strategies can increase the quality of the development process [11], [12],[16], [17].

Taking into account the size of the different teams involved in the development process, an appropriate workspace environment and a common area generally improve communication aspects especially in small teams [7]. Considering the teams' locations, a higher communication effort is required in case of distributed development [18], [19], [20], [23], [22].

XP '18 Companion, May 21–25, 2018, Porto, Portugal

ACM ISBN 978-1-4503-6422-5/18/05…$15.00
https://doi.org/10.1145/3234152.3234191

3 COMMUNICATION ISSUES IN MICROSERVICES DEVELOPMENT

MS development falls in the same domain of global software development (GSD), where independent teams develop different parts of the same system. In both cases, the teams are essentially separated, and the attempt is that there is as little common ground as possible that developers need to consider. In MS-based development, this is a direct result of the who MS philosophy, that the services are independent, while in GSD, the need for separation comes from practical experience on how difficult it is to manage a project where distributed teams need to work on the same piece of software [5]. Thus, while the drivers for the separation of teams are very different, the resulting team structure is essentially the same. Even if the communication among teams in MS-based systems is reduced, the same issues are experienced as in GSD. Therefore, we could experience the common division of GSD communication issues as [18], [19], [21]. Here we list the possible communication scenarios among teams working on different MS.

- Collocated teams: all teams are in the same location. In this case, communication is easy and, even if teams are not working together on the same project, they can easily exchange project information during non-developing hours.
- Distributed development with overlapping work hours: teams work in different geographical locations but they have only few hours during the workday in which they interact with each other. If needed, meetings can be held during the overlapping time.
- Distributed development with NO overlapping work hours: Teams have no interaction during their working hours, therefore synchronization and meetings require more organization effort.

A possible solution to overcome the communication issues is to introduce at least one level of hierarchy; adding one member that will act as teams coordinator. Moreover, the complexity of MS system development requires to rely on a software architect. In some companies, the architect could also play the role of a coordinator.

Issues could be solved in different ways:

- *Strict one-level hierarchy.* In case of problems the coordinator will take a decision, and he/she will be the sole responsible of the decision.

  Pros: This approach speeds up the decision time, as only one person needs to take the decisions without the need of coordinating among other members.
  Cons: Non-democratic decisions always have the issue of not being widely accepted by the members. Moreover, the coordinator should have a higher level of expertise, since the decision comes from one single person, other solutions (in some case more efficient) could apply to the same problem.

- *Two-Level Hierarchy.* Decisions taken by the coordinator supported by the responsible person of each MS involved. The responsible person of each MS could be the Scrum master (in case of Scrum processes) or a similar figure.

  Pros: This approach enables teams to have their representative included in the decision processes. Decisions could be more beneficial for all the teams.
  Cons: Synchronizing different coordinators could take a relatively long time depending on the locations, time zones, and availabilities of the different MS responsible. Moreover, in some cases, MS responsible could need more time to discuss with their team members before taking a decision.

- *Full Democracy.* Decisions are taken with a discussion that involves all team members (or a reasonably big representative) of each MS.

  Pros: In this approach, no members are excluded. Due to high level of information sharing, teams aim at gathering all the opinions before taking any decisions.
  Cons: The more there are people involved in the decision making process, the longer the discussion time will be. Therefore, in case of agile teams who need to make quick decisions, this approach, beside its pros, could provide more drawbacks, creating deadlocks in the development processes.

Different communication issues could be also solved by different communication means. Synchronous communication, including face-to-face meeting, online chat and videoconference or telephone call, are by far the most effective approaches[21]. However, the main drawback is that they require the simultaneous presence of all the actors involved in the communication itself. Asynchronous communication includes email messages, but also time to write and read documentation. Asynchronous communication has the benefit of being more traceable than synchronous and not requiring the availability of the actors at the same time. However, the main drawback is the time needed to get answers.

Considering the different roles inside the development and management team, some issue can also be solved by a proactive approach of the entire management team and by including the customers as much as possible[2].

4 THE EMPIRICAL STUDY

In this section, we propose the design of an empirical study [24] aimed at comparing the communication issues and benefits of the team communication between monolithic and microservices. Therefore, we designed this study as multiple case study [25], that will be conducted in different companies that migrated from monolithic systems to microservices.

According to our expectation, we formulated the study goal according to the GQM approach [26] as follows:
Analyze the communication process
For the purpose of comparing
With respect to its effort, benefits and issues
From the point of view of software developers
In the context of the development of monolithic and microservices-based systems.

This leads to the following research questions:

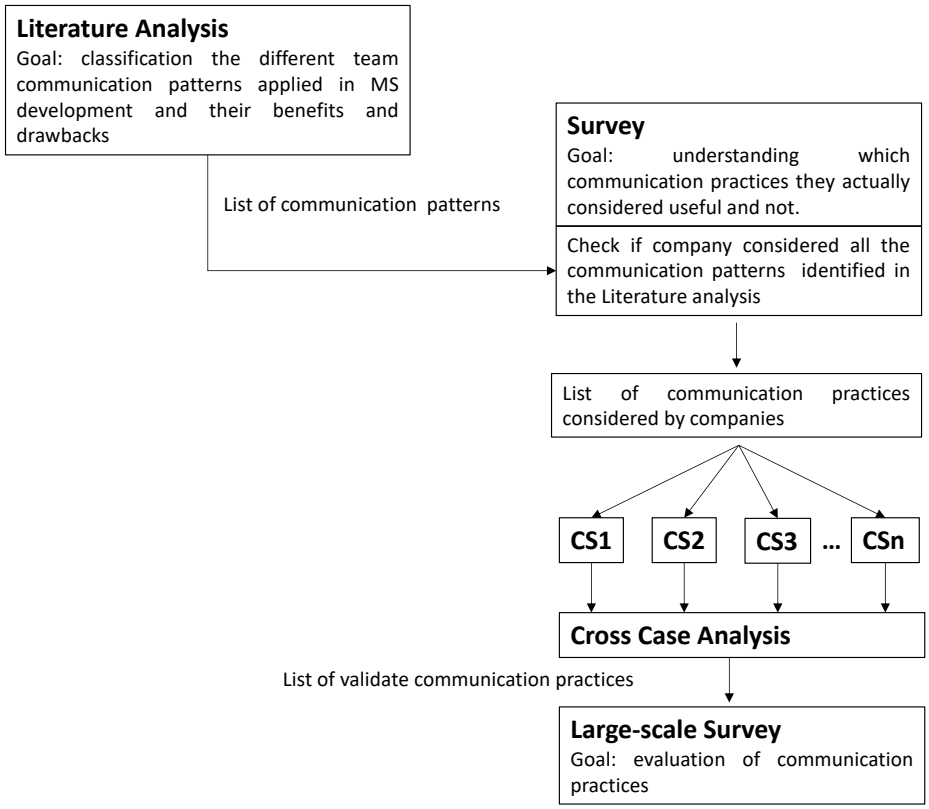

Figure 1: The Approach

Q1 : Which development process requires more communication time? We expect that teams developing microservices, because of microservice nature, require less communication time than team working on monolithic systems.

Q2 : Which development process (monolithic vs microservices) has the highest communication overhead? Also in this case, we expect that the teams working on microservices will have a lower overhead. We propose the following metrics to calculate the overhead:

- M2.1 Development effort (hours)
- M2.2 Total effort. Includes development and communication effort.
- M2.3 Communication overhead. We define communication overhead as ratio between development time and communication time.

Q3 : Which communication issues and benefits have microservices?

In order to compare the communication time among different processes, we designed this study as multiple case study with replication design [25]. We aim to involve different teams from different companies, measuring the process by means of a questionnaire to ask their feedbacks. The study population must be composed by experienced developers.

5 ROADMAP

In this Section we illustrate how we propose to carry out the study, as depicted in Figure1.

(Step 1) **Literature Analysis**. We will perform a literature review with the goal of classifying the different team communication patterns applied in MS development and their benefits and drawbacks.

(Step 2) **Survey**. We survey practitioners to understand which communication practices they actually considered useful and not.

(Step 3) **Multiple Case Studies**. The multiple Case Studies are composed by two sub-steps:
Cases Selection and execution. We will select a set of companies to run our case studies. Companies will be of different domains but working with microservices for at least five years. We will perform the same case study to all the companies independently, so the results of one case study will not influence the others.
Cross Case Analysis. We will analyze all case studies independently. Moreover, we will perform a cross-case analysis to identify commonalities and differences in the communication practices.

(Step 4) **Large-scale survey**. We will test the results obtained by performing a large-scale online survey. The survey

will be composed of closed-ended questions, which allow collecting a large number of useful information about the benefits and issues on each communication practice.

6 CONCLUSION

Compared to monolithic systems, MS development requires less communication among different teams.

In this work we aimed at raising some discussion points about the negative aspects of the reduced need of communication in MS-based software development. The communication issues are partially similar to those highlighted in global software development, when teams need to communicate with different means, in order to synchronize their work. Different communication issues could be addressed with different communication means and strategies. Moreover, we aim at designing an empirical study to validate the hypothesis that the reduced communication need in teams developing microservices does not imply negative effects on the synchronization.

This work will be carried out in collaboration with other research teams, and sharing the results with other projects such as [27] and [28].

The results of this work is mainly aimed at sharing our experience and thoughts on the need of communication in ASD and microservices, so as to define a shared research road-map together with the workshop participants.

REFERENCES

[1] J. Lewis and M. Fowler. MicroServices. available at: www.martinfowler.com/articles/microservices.html (2014)

[2] D.Taibi, V. Lenarduzzi and C. Pahl. Processes, Motivations and Issues for Migrating to Microservices Architectures: An Empirical Investigation. IEEE Cloud Computing. Vol. 4, Issue 5, Article number 8125558. pp. 22-32. (2017)

[3] D. Taibi and V. Lenarduzzi. On the Definition of Microservices Bad Smells. IEEE software. March 2018 (in press).

[4] D. Taibi, V. Lenarduzzi, P. Diebold, and I. Lunesu. 2017. Operationalizing the Experience Factory for Effort Estimation in Agile Processes. 21st International Conference on Evaluation and Assessment in Software Engineering (EASE'17). pp. 31-40. (2017)

[5] M. Cataldo, J. Herbsleb, and K. Carley. Socio-technical congruence: a framework for assessing the impact of technical and work dependencies on software development productivity. Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement (ESEM'08), pp. 2-11. (2008)

[6] Pikkarainen, M., Haikara, J., Salo, O., Abrahamsson, P., and Still, J., The impact of agile practices on communication in software development. Empirical Software Engineering. Vol. 13(3), pp. 303-337, 2008

[7] Mishra, D., Mishra, A., and Ostrovska, S., Impact of physical ambiance on communication, collaboration and coordination in agile software development: An empirical evaluation. Information and Software Technology. Vol 54(10), pp.1067-1078, 2012.

[8] D. Taibi and V. Lenarduzzi. MVP explained: A Systematic Mapping on the Definition of Minimum Viable Product. 42th Euromicro Conference on Software Engineering and Advanced Applications (SEAA2016). pp. 112-119. (2016)

[9] Bhalerao, S., Puntambekar, D. and Ingle, M., Generalized agile software development life cycle. International Journal of Computer Science and Engineering. Vol I(3), 2009.

[10] D. Taibi, V. Lenarduzzi, A. Janes, M.O. Ahmad and K. Liukkunen. Comparing Requirements Decomposition Within the Scrum, Scrum with Kanban, XP, and Banana Development Processes. Agile Processes in Software Engineering and Extreme Programming (XP2017). pp. 68-83. (2017)

[11] Korkala M., Abrahamsson P., and Kyllonen P., A case study on the impact of customer communication on defects in agile software development. AGILE 2006, pp. 76-88, 2006.

[12] Sarker, S., and Sarker, S., Exploring Agility in Distributed Information Systems Development Teams: An Interpretive Study in an Offshoring Context. Information Systems Research, Vol.20(3), pp.440-461, 2009.

[13] Wang, X., Conboy, K., and Pikkarainen, M., Assimilation of agile practices in use. Information Systems Journal. Vol 22(6), pp. 435-455, 2012

[14] D. Tosi, L. Lavazza, S. Morasca, and D. Taibi. On the definition of dynamic software measures. ACM-IEEE international symposium on Empirical software engineering and measurement. pp. 39-48. (2012)

[15] L. Lavazza, S. Morasca, D. Taibi, and D. Tosi. Applying SCRUM in an OSS Development Process: An Empirical Evaluation. Agile Processes in Software Engineering and Extreme Programming (XP2010). pp. 147-159. (2010)

[16] Koskela, J., and Abrahamsson, P., On-Site Customer in an XP Project: Empirical Results from a Case Study. Torgeir DingsÃÿyr (Ed.) Software Process Improvement, Springer, Berlin Heidelberg, pp.1-11, 2004.

[17] Melnik, G., and Maurer, F., Direct Verbal Communication as a Catalyst of Agile Knowledge Sharing. AGILE 2004, pp.21-31, 2004.

[18] D. Taibi, P. Diebold, C. Lampasona. Moonlighting Scrum: An Agile Method for Distributed Teams with Part-Time Developers Working during Non-Overlapping Hours. ICSEA International Conference on Software Engineering and Advances , IARIA. (2013)

[19] V. Lenarduzzi, I. Lunesu, M. Matta and D. Taibi. Functional Size Measures and Effort Estimation in Agile Development: A Replicated Study. Agile Processes in Software Engineering and Extreme Programming (XP2015). pp. 105-116. (2015)

[20] H. C. Estler, M. Nordio, C. A. Furia, B. Meyer and J. Schneider. Agile vs. structured distributed software development: A case study. Empir. Software Eng. Vol. 19(5), pp. 1197-1224, 2014

[21] D. Taibi, V. Lenarduzzi, M. O. Ahmad, and K. Liukkunen. Comparing Communication Effort within the Scrum, Scrum with Kanban, XP, and Banana Development Processes. 21st International Conference on Evaluation and Assessment in Software Engineering (EASE'17). pp. 258-263. (2017)

[22] L. Lavazza, S. Morasca, D. Taibi, and D. Tosi. An empirical investigation of perceived reliability of open source Java programs. In Proceedings of the 27th Annual ACM Symposium on Applied Computing (SAC '12). pp.1109-1114. (2012)

[23] L. Lavazza, S. Morasca, D. Taibi, and D. Tosi. Predicting OSS trustworthiness on the basis of elementary code assessment. In Proceedings of the International Symposium on Empirical Software Engineering and Measurement (ESEM '10). Article 36 , 4 pages. (2010)

[24] P. Runeson, and M. Höst. Guidelines for Conducting and Reporting Case Study Research in Software Engineering. Empirical Softw. Engg. vol.14(2). (2009)

[25] R.K. Yin. Case Study Research: Design and Methods, Applied Social Research Methods. SAGE Publications, Inc. (2009)

[26] V.R. Basili, G. Caldiera, and H.D. Rombach. The Goal Question Metric Approach. Encyclopedia of Software Engineering, Wiley. (1994)

[27] V. Lenarduzzi and O. Sievi-Korte. Software Components Selection in Microservices-based Systems. 19th International Conference on Agile Software Development (XP2018). (2018)

[28] F. Auer, M. Felderer and V. Lenarduzzi. Towards Defining a Microservice Migration Framework. 2th International Workshop on Microservices: Agile and DevOps Experience (MADE18). (2018)