SOCIO-TECHNICAL SELF-DEVELOPMENT USING A MICROSERVICE ARCHITECTURE

Johann Sell and Niels Pinkwart
Humboldt Universität zu Berlin, Germany

ABSTRACT

Considering *joint optimization* and *organizational choice*, socio-technical systems have to align their technical component with the ongoing change of the social system. Focusing on decentralized, loosely coupled socio-technical systems, this challenge becomes more complex, because the various parts of the social system will develop different requirements for the technical support. Focusing on a case study, we develop requirements and compare different possible architectures. We approach to use a microservice architecture to support maintenance and further development of the technical tool.

KEYWORDS

Microservices, Socio-Technical Systems, Evolutionary Organizations, Joint Optimization, Organizational Choice

1. INTRODUCTION

Our work aims to support a loosely coupled socio-technical organization in terms of self-development. We focus on ensuring compliance of *joint optimization* and *organizational choice* as most important challenges (Trist 2013; Emery 1959). *Joint optimization* implies that every change in one of both systems (technical or social) also requires changes in the opposite part. *Organizational choice* describes a needed degree of freedom, thus users are allowed to adjust their usage of the supporting system.

The case study Viva con Agua de St. Pauli e.V. (VcA, vivaconagua.org, April 2018) has developed very heterogeneous requirements regarding its technical part, because of the loosely coupled nature of its social system. Next to VcA there is an increasing set of organizations boasting the same characteristics, like foodsharing (foodsharing.de, May 09, 2018), Buurtzorg (buurtzorg.com, May 09, 2018) or FAVI (favi.com, May 09, 2018) (Laloux 2014). This paper discuses several possible architectures to address the issue of heterogeneous requirements considering a loosely coupled socio-technical system. We suggest to use microservices (Newman 2015; Zimmermann 2017; Dragoni et al. 2017) to tackle this issue.

VcA is a non-governmental organization raising awareness and collecting donations for WASH ("Water, Sanitation and Hygiene") projects. It is based on decentralized volunteering, organized in loosely coupled regional groups, called *crews* (Weick 1976). As already described by (Sell & Pinkwart 2016) its success is based on *open participation* and coordination using a web-based tool named *Pool*. Hence the tool became part of the social systems self-description and both joined into a socio-technical organization (Kunau 2006). Unfortunately, *joint optimization* and *organizational choice* became impossible to apply, due to limited men power and inflexibility of the *Pools* architecture as we will describe in this paper. Additionally, the current implementation is neither expandable nor maintainable.

We have identified three high level requirements for the technical support system: (R1) Since the *crews* are loosely coupled, new technical support may fit only requirements of a few *crews*. So, the possibility to add support for complete work processes only for a few *crews* is needed. (R2) Obviously, *crews* have to replace existing technical support of work processes. For example, the general technical support for handling finances has to be replaced for some *crews* due to alternative roles that have been evolved. Normally, a *crew* has one person responsible for finances, but there are *crews* separating between a person responsible for donations and another for housekeeping money. (R3) Since development of software requires a lot of time, and volunteers already invest a huge amount of time regarding to the goals of VcA, a distinct community of developers has to evolve. Such a community differs from Open Source communities by the motivation that is

not to help themselves and others, but to acquire new skills, and investigate ideas with a huge number of users. Thus, it is quite likely that it is too much effort to take the time to learn a new technology.

In the end, a decentralized, loosely coupled social system of software developer evolves. Each one focuses small parts of the technical system and participates in terms of *open participation*. We are convinced that such a social system is obliged to consider the ongoing change of VcA and will be able to apply *joint optimization* and *organizational choice*.

2. SOLUTION CONCEPT

We approach to foster self-development of VcA. We aim to create a new loosely coupled, decentralized and volunteering community of software developers that by themselves are able to build new support functions for the social system. Thus, dynamic integration of developing teams without the need for training of specific technologies (R3) or cumbersome bureaucracy is required. This new community of software developers (Pool-volunteers) has to be separated from the existing volunteers focusing the WASH projects (WASH-volunteers), since both activities require a lot of time.

2.1 Characterization of Possible Architectures

Basically, we have identified four types of possible architectures: Monolithic systems, modular implementations, app based extension of core functionality, and microservices. There are socio-technical communities using their own special purpose tools for loosely coupled volunteering (Dittus et al. 2016). Other communities compose various technical tools (Gray et al. 2016; Ambe et al. 2016; Reuter et al. 2015).

In general, monolithic architectures exhibit *high cohesion* between implemented functions (Newman 2015). Thus, software enhancements normally have to use the same set of technologies as the origin.

In contrast, there are applications implemented very modular, like Moodle (moodle.org, February 2018) or MediaWiki (mediawiki.org/wiki/MediaWiki, February 2018). In general, these modular applications have a core and can be extended by plugins. Free choice of technology is bounded to its purpose. While developers can use one technology to extend the core, a second can be used to implement plugins.

Facebook (facebook.com, February 2018), Instagram (instagram.com, February 2018), Slack (slack.com, February 2018), and Trello (trello.com, February 2018) allow to extend the functions of its software by third-party apps. Therefore, those apps get user data and a sandbox area to deploy their functionality.

The last opportunity are microservices. First, microservices are independent processes and cohesive in bounded contexts (Dragoni et al. 2017; Namiot & Sneps-sneppe 2014; Newman 2015; Viennot et al. 2015). Second, they use message based communication via well-defined interfaces, like RESTful webservices (Dragoni et al. 2017; Rodriguez 2008). A set of connected services is called a microservice architecture (Dragoni et al. 2017; Namiot & Sneps-sneppe 2014). Such architectures force the designer to minimize communication between microservices (*loosely coupling*) and to implement related behavior inside the same scope (*high cohesion*) (Newman 2015).

2.2 Choosing an Architecture Fitting the Requirements

Volunteers of VcA use several different tools to balance their needs (Sell & Pinkwart 2016). However, VcA is an organization, thus there are important and sensible processes, like the handling of finances. Furthermore, the need for different user accounts and invisibility of used tools are serious obstacles considering *open participation*. Therefore, the usage of many independent tools breaks the organizational structures. A monolithic or module based architecture requires specific technologies. Thus, (R3) becomes impossible to fulfill, since developers would need to learn about the technologies.

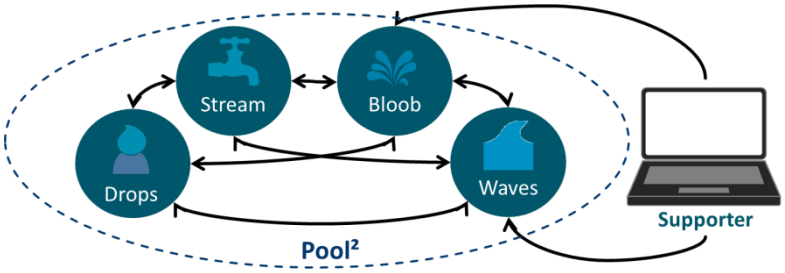

Figure 1. Microservice architecture for Pool² with arrows indicating possible communication channels. Drops *(user management)*, Stream *(finances)*, Bloob *(email broadcasting)*, *and* Waves *(activities and events)*

Adopting the concept of apps for VcA requires its enhancement. First, apps are normally introduced to allow additional functions, but not to replace existing ones. Thus, the hard separation between core application and apps has to be softened due to (R1). Furthermore, it should be possible to replace all core functions with new implementations. Thus, apps become loosely coupled, as in the case of microservices.

Microservices spread responsibility for maintains and support among different teams. Thus, it supports the management of growing complexity and delegates systems modularity from a technical implementation to an organizational layer. Modularity represented in organizational structures supports independent development and deployment (Dragoni et al. 2017) (R1; R2). Furthermore, a microservice architecture allows different programming paradigms, languages, and database technologies for different parts of the system (Dragoni et al. 2017; Namiot & Sneps-sneppe 2014; Viennot et al. 2015). Thus, developers with various expertises can contribute to the new system (R3). SAP, for instance, is developing an additional mobile App.

In the end, self-development of a socio-technical organization will be possible, if the organization needs decentralized further development. Pool-volunteers can implement specific requirements of individual *crews* without changing the whole system.

2.3 Implementation

First, we implement the existing features of the *Pool* using the new architecture. We have decided to introduce four microservices addressing the functional requirements, as shown in Figure 1. *Drops* handles the representation of volunteers and the authentication. *Bloob* implements communication support, while *Stream* supports the handling of finances and *Waves* the coordination of volunteers. The new system is called *Pool²*.

The microservice architecture implies several challenges. As described in section 2.1, exchange of data between services is implemented using RESTful webservices, while the event message broker nats (nats.io, February 2018) informs about updates. OAuth 2 (oauth.net/2/, February 2018) is used to implement a shared session to prevent different accounts. Docker (docker.com, March 2018) wraps the microservices inside standalone container next to each other and allows deployment of heterogeneous technologies.

One central microservice handling the user interface would imply one team of developers has a lot of responsibility. Thus, every microservice has to implement its own user interface. Ensuring a consistent user interface across service boundaries becomes a serious obstacle. We propose to adopt the *UI Fragment Composition* (Newman 2015). The microservice *Dispenser* distributes templates containing basic HTML elements, like a global menu, header or footer. Additionally, it allows reused CSS and a corporate design.

Furthermore, all microservices have to implement the concept of widgets. A widget is a HTML element with optional CSS. It is used by other services to show content or a function that is in *high cohesion* to the widgets provider. Therefore the service is responsible for definition, delivery, and maintenance of the widget. Its delivery takes up the idea of *transclusions* (Nelson 1965; Kolbitsch & Maurer 2006). For instance, the *Drops* service should implement a widget for an auto completion field that could be used to select a user. We will implement a JavaScript function, that watches an HTML input field, communicates with *Drops*, and shows a list of users. Both concepts apply the paradigms *loose coupling* and *high cohesion* to the view layer.

3. CONCLUSION AND OUTLOOK

This paper derives requirements from the characterization of the decentralized, loosely coupled, socio-technical organization VcA. Different architectures for a technical system are compared to support

socio-technical self-development. A microservice architecture has been identified assisting *joint optimization* and *organizational choice* for organizations like VcA. Additionally, a rough introduction to the implementation is given and solutions to important challenges are outlined.

Currently, we are implementing a proof of concept. Afterwards, the implications for the social system of WASH-volunteers have to be explored, like a possibly needed quality assurance or coordination support for the Pool-volunteers. Moreover, the process of further development has to be investigated: Can a new loosely coupled social system of Pool-volunteers be established and does the *Pool²* really enable loosely coupled and decentralized technical artifacts to form a coherent collaboration tool for daily work?

First, we have to figure out how much additional work is required to integrate a new technology. Effort can be measured by tracking or estimating time needed by experts. Valid estimations need complete descriptions of the non-functional requirements regarding the integration of a microservice. Such descriptions can be analyzed qualitatively, with focus on *loosely coupling* and *high cohesion*. The new architecture has to enforce these principles, since new developers could have few experiences in software architecture.

Obviously, we also have to investigate further adaption by the volunteers. Basic questions are, if they formulate new requirements and find developers to implement those requirements. At least, our approach should be validated by applying it to other organizations. Our further research will focus these questions.

REFERENCES

Ambe, A.M.H., Brereton, M.F. & Rittenbruch, M., 2016. Vendors- Perspectives of Coordination in the Information Technology Offshore Outsourcing Industry: An Exploratory Study from the Philippines. In *Proc. CSCW '16*. New York, USA: ACM Press, pp. 318–333.

Dittus, M., Quattrone, G. & Capra, L., 2016. Analysing Volunteer Engagement in Humanitarian Mapping: Building Contributor Communities at Large Scale. In *Proc. CSCW '16*. New York, USA: ACM Press, pp. 108–118.

Dragoni, N. et al., 2017. Microservices: yesterday, today, and tomorrow. Available at: https://arxiv.org/pdf/1606.04036.pdf [Accessed February 10, 2017].

Emery, F.E., 1959. *Characteristics of sociotechnical systems: A critical review of theories and facts about the effects of technological change on the internal structure of work organisations; with special reference to the effects of higher mechanisation and automation*, London.

Gray, M.L. et al., 2016. The Crowd is a Collaborative Network. In *Proc. CSCW '16*. New York, USA: ACM Press, pp. 134–147.

Kolbitsch, J. & Maurer, H., 2006. Transclusions in an HTML-Based Environment. *CIT*, 14(2), pp.161–173.

Kunau, G., 2006. *Facilitating computer supported cooperative work with socio-technical self-descriptions*. Technische Universität Dortmund. Available at: http://hdl.handle.net/2003/22226.

Laloux, F., 2014. *Reinventing Organizations* 1st ed., Brussels: Nelson Parker.

Namiot, D. & Sneps-sneppe, M., 2014. On micro-services architecture. *INJOIT*, 2(9), pp.24–27.

Nelson, T.H., 1965. Complex information processing: a file structure for the complex, the changing and the indeterminate. In *Proc. ACM '65*. New York, USA: ACM Press, pp. 84–100.

Newman, S., 2015. *Building Microservices* 1st ed. M. Loukides, B. MacDonald, & K. Brown, eds., O'Reilly Media.

Reuter, C. et al., 2015. XHELP: Design of a Cross-Platform Social-Media Application to Support Volunteer Moderators in Disasters. In *Proc. CHI '15*. New York: ACM Press, pp. 4093–4102.

Rodriguez, A., 2008. RESTful Web services: The basics. Available at: https://www.ibm.com/developerworks/library/ws-restful/ [Accessed March 17, 2017].

Sell, J. & Pinkwart, N., 2016. Rambla: Supporting collaborative group creativity for the purpose of concept generation. In T. Yuizono et al., eds. *Proc. CRIWG '16*. Kanazawa, Japan: Springer, pp. 81–97.

Trist, E.L., 2013. *Organizational Choice: Capabilities of Groups at the Coal Face under Changing Technologies - The Loss, Re-discovery & Transformation of a Work Tradition* 1st ed., London: Routledge.

Viennot, N. et al., 2015. Synapse: a microservices architecture for heterogeneous-database web applications. In *Proc. EuroSys '15*. New York, USA: ACM Press, pp. 1–16.

Weick, K.E., 1976. Educational Organizations as Loosely Coupled Systems. *ASQ*, 21(1), pp.1–19.

Zimmermann, O., 2017. Microservices Tenets: Agile Approach to Service Development and Deployment. *CSRD*, 32(3), pp.301–310.