Engineering Software Engineering Teams

Patrick Wagstrom
IBM TJ Watson Research Center
19 Skyline Dr,
Hawthorne, NY 10532 USA
pwagstro@us.ibm.com

ABSTRACT

This paper presents novel ideas for understanding how software engineering teams communicate and coordinate. We utilize these ideas to understand how these teams should be constructed and what individuals and managers can do to ensure that teams perform at high levels. Our view is based on numerous observations and interactions with enterprise software engineering teams and influenced by economic models of information sharing. We propose that neither a fully top-down nor bottom-up approach is entirely suitable for teams; rather teams must be cognizant of this issue and work to embrace both models of information flow. This, in turn, can be facilitated by the role of intercessor who seeks to properly guide, direct, and curate both top-down and bottom-up information flows.

Categories and Subject Descriptors

D.2.9 [**Software Engineering**]: Management – *life cycle, productivity, programming teams, software process models.*

General Terms

Management, Human Factors, Economics

Keywords

Software Engineering, Development Teams, Communication, Coordination, Intercessor

1. INTRODUCTION

Organizational composition and process/workflow design for software engineering teams is perhaps one of the most difficult aspects of building and managing a high performance software engineering organization. In addition to managing the day to day work of individual developers, project managers must balance the needs and desires of a wide variety of other individual stakeholders, both tightly and loosely aligned with the project [14].

A key aspects of maintaining this balance while building and running a software engineering team is managing the flow of information between participants in the team – a task that has been modeled as the primary requirement of organizations in any context [2, 4]. In brief, within an information processing view of an organization the participants in an receive pieces of information from a variety of different sources (e.g. people, events, artifacts, personal knowledge, etc) and it is the responsibility of the individuals that make up the organization to utilize that information to produce a useful output, typically by gaining enough information to complete a task [11]. For example, the combat operations of a military represent only a small portion of its responsibility. The bulk of the work of the military is in collecting information to plan operations, manage supply lines, and coordinate actions in a reliable and reproducible manner as often exemplified by teams working on the flight decks of aircraft carriers [13].

The mapping to a software engineering organization can be performed similarly. While developers are often thought of being at the core of the organization and a variety of tools exist to support both planned and ad-hoc interactions between developers, much of the planning and work within a project is done by non-developers: product managers, client executives, brand strategists, financial officers, and other external stakeholders. Among other tasks these stakeholders process and disseminate information about customer needs, future market states, financial model, and process workflows to support the developers coding the software project.

While there are numerous tools to propagate information through software engineering environments, such as enhanced IDEs, project planning software, requirements management tools, and general purpose wikis and bug trackers, these tools often face significant challenges that make consumption of their information useful across the entire team. For example, non-developers rarely use IDEs and may find bug trackers too specific; requirements management tools may rationalize decisions for project specifications, but not provide a way for developers to understand the context of the decision; and tools that provide real time notification of developer and stakeholder activities may prove less useful in the context of a temporally distributed team where all notifications are seen en masse when the tool is started at the beginning of the work day.

Strategic design of a software engineering organization to cope with temporal asynchronicity, divergent stakeholder requirements, distributed information, and, perhaps most importantly, cross-silo information flow is a critical need in any software engineering team with more than a handful of members. In essence, we must engineer the software engineering team.

This paper first provides an overview of extended stakeholders in software engineering organizations and how this expanded organization view is informed by macro-economic theories of organizations. In section 2 we discuss possible methods for integrating stakeholders based on the views of both Austrian and Keynesian economics. In section 3 we address different ways that tools and processes support these methods of integrating stakeholders. In section 4 we discuss how these stakeholders monitor and process information and finally we close the paper in section 5.

2. SOFTWARE STAKEHOLDERS AND MACROECONOMIC THEORIES

In a previous research project we identified a multitude of stakeholders beyond that of the traditional software engineering team [14]. These stakeholders include those that are involved with the daily process of software development, such as project managers, designers, and support teams, and also those who are responsible for the business of producing software while lacking frequent access to the code, such as brand managers, marketing, legal support, and executive level individuals.

In a follow-up piece of research we examined the communication pattern within these large software development organizations. While there were a multitude of issues regarding coordination and collaboration one overarching issue found was the problem of private information [8]. In short, this model, which builds on the Austrian school of economics as typified by Hayek, views coordination as an issue of multiple independent actors each possessing a small amount of the knowledge necessary to complete the task [6]. The independent actors then exchange their private information to obtain a clear picture of the complete situation. In economic theory those individuals with the most information will be able to extract additional rents from the market. In the software engineering sense, individuals with the most information will be able to coordinate the best, minimize the amount of time spent on information seeking, and maximize their performance. Unlike the purely economic model, in a software context instead of extracting additional rents, the most successful individuals are free to contribute to other areas of the project, improving the overall project.

However, there is an alternative way to think about economics that is just as easily applied to software engineering teams, that of Keynesian economics. In brief, Keynesian economics asserts that relying on private enterprise can often lead to inefficient allocation of resources and therefore there is a role of a large contribution from the public sector [7]. In such an environment a large central agent collects information and provides concrete direction to the private parties. This minimizes the perceived inefficient process of knowledge transfer and replaces it with the possibly inefficient allocation of resources by a central planner.

With respect to the design of a software engineering team a Keynesian model has a central project manager that serves as the primary, but non-exclusive, arbiter of information flows while assigning tasks and working to coordinate developer working on different modules, project managers, and external stakeholders. In essence, a Keynesian software engineering organization is top-down controlled. This is in contrast to an Austrian software engineering organization that allows control to percolate up from the bottom of the organization.

Perhaps one of the best known works that addresses these issues of top-down against bottom-up control of the software engineering organization is Eric Raymond's work "The Cathedral and the Bazaar", which first put into writing many of the common practices of Open Source projects in stark contrast to the formal waterfall engineering process that typified many organizations in the mid-late 1990's [12]. While Raymond's work was largely based on anecdotal evidence from his own participation in a medium-sized Open Source project, it had a significant impact on the state of software engineering as organizations sought to reap the benefits of Open Source software and the Open Source style of software development within their organizations.

While the Open Source style of software development empowers individual developers to exchange information, for example through the use of public bug trackers, mailing lists, synchronous chat, and verbose code, it does so at the expense of extended stakeholders. Indeed, one of the classical models of Open Source participation proposes that in order to understand a project more and become more core to its development a developer moves through successive layers from non-technical communication mediums such as mailing lists to the technical medium of code development [9]. While this process may be an efficient method of transferring private information between stakeholders who are directly involved with the development process, the role of external non-developer stakeholders is marginalized by this process. A non-developer stakeholder who wishes to have impact on the direction of a project is required to work deep in the code of the project and collaborate at a level they may not otherwise be comfortable. For example, an individual who is concerned about the accessibility features of the software for those with vision or hearing impairments may be asked to file bugs documenting where the feature is lacking.

This makes it no small wonder that many projects that are Open Source or operate using a development strategy similar to the Open Source process are populated primarily by individuals that are also developers of the software. Also, it explains why many of the most successful software projects are those that are designed for software developers. Indeed, this is made very obvious by dual user/developer nature of participants in many Open Source projects [3].

Therefore, while Open Source-like strategies can serve as a base level mechanism for engineering a software engineering team, we need to expand the model to include stakeholders who are not well served by the user/developer duality that is focus of many Open Source projects.

3. INTEGRATING EXTENDED STAKEHOLDERS

Perhaps the near polar opposite of the bazaar-like ad hoc coordination structure of Open Source projects is the realm of formal process structure and formal process definitions. These systems are typically designed to ensure that the viewpoints from all stakeholders are accounted for during the development process. An example of this is the Integrated Product Development (IPD) process from IBM [5]. This is a stage/gate process that requires a product to go through a series of votes at different points. Each vote is cast be a set of stakeholders who have an established and explicit role to play and a business purpose to represent. For example, in addition to individuals representing development, marketing, sales, support, and a variety of other roles are also represented at votes to move to the next stage of a project. While this helps, but does not guarantee, that the project has heard and addressed the viewpoints and concerns of the different stakeholders in the project, the rigid stage gate structure often hinders an organization from acting in an agile method and requires a level of control and information processing that may not be possible in all environments, particularly those that are fast moving.

An approach to allow for stakeholder integration without overburdensome processes is to provide instrumentation within tools that allows for dynamic tool composition. One such approach is collaborative application lifecycle management (C/ALM) as implemented through the Open Services for Lifecycle Integration (OSLC) [10]. OSLC seeks to provide a common framework for tools to share information between them by, for example, allowing a bug filed on a bug tracker to easily refer to the original requirement in the requirements document, the work item which instantiated the requirement, and a build in which the bug related to the requirement was found. This allows project managers to integrate all of the tools used by the team and in some cases even provide for ad-hoc integration of new tools through standardized interfaces, provided, of course, that the tools support the current standard and that project members understand how to make use of the additional information.

One potential downside of C/ALM systems is that interaction is limited by the features provided in the existing tools – tools that are typically designed for utilization by a specific class of stakeholder. Thus, while it allows for some information to be passed between tools it lacks the ability to adapt to the situations and pass relevant information to stakeholders who may not use such a tool – for example, a brand executive who conducts most of his work in email and word processing documents. Furthermore, they require that the individuals who implement the C/ALM layer of each tool and propose the overall integration understand and anticipate possible use cases of the additional information, which is likely to be difficult.

4. MONITORING AND PROCESSING INFORMATION

Thus far we have argued that a software engineering team should be structured in such a way that each member of the team and extended stakeholders can easily pass information to those individuals who need access to the information. As the problem landscape around software projects is continually changing this must be done in an agile manner such that dynamic changes in team structure do not hinder the process. While integration of tools can assist to a moderate degree, we require a framework that allows for lightweight coordination outside of tools, especially when the set of stakeholders is unknown or a stakeholder does not utilize standard tools.

A further consideration for the organization is to easily facilitate the monitoring and processing of information as it is passed to team members. When an individual on a team receives a new piece of information they can do one of three things with the information: ignore the information, store the information for possible future action, or immediately act on the information. Each of these three options takes time on the part of the individual and has the potential to distract the individual from their current work. Therefore it is desirable to pass information in a way that team members are able to fully consider relevant information and easily defer or ignore less relevant information.

In an organization that favors bottom-up communication and coordination there exists few checks to ensure that this information has been properly processed. Likewise, in an organization with primarily top-down communication and coordination while it may be easy for an overseer to ensure the information is acted on, it can be difficult to ensure that the information is routed properly to right person.

This strongly hints that information processing organizations require an individual to guide and curate organizational knowledge. Working from the top-down they can ensure that the concerns from external project stakeholders are routed to the correct core team members. While a bottom-up perspective allows them to monitor the flow of information to external stakeholders and direct it to the correct individual. These individuals have various terms in the literature, such as intercessors, connectors, or structural holes [1], but they are commonly known by a name which has almost become derogatory: middle management. In an effort to avoid unnecessarily loading the term, we will use the term intercessor for this role.

However, just as not everyone can be a master gardener not everyone is naturally suited to play the role of intercessor. Indeed a master gardener must have an innate sense and knowledge of how the rain and sunlight coming from above interact with the seeds, soil, and nutrients from below. Likewise, an intercessor in an organization must be more than someone who knows people and can direct information to other people in the organization; they must be someone who has an intimate knowledge of the needs of the different members of the organization. This is not to say that an intercessor needs a complete, and possibly unobtainable, view of all interactions in the organization or that they must know and understand exactly which pieces of knowledge individuals have and how they can share them. Rather, the goal of an intercessor is to direct the information in a software engineering team to the individual for which it matters most and will have the greatest impact.

The exact role and actions of an intercessor vary from project to project. In successful Open Source projects it is common to have an individual or group of individuals triage incoming bugs. These individuals serve to bridge the gap between the external stakeholders of end users and the development teams by categorizing bugs and requesting additional information where necessary. In open content environments, such as Wikipedia, an intercessor may work to curate the content and better organize articles to meet style guidelines. This allows experienced editors to focus strictly on the content of articles while providing ample room for new editors to provide suggestions and augment articles with minimal fear of disturbing the workflow of experienced editors.

However, in many commercial projects the role of intercessor falls to the project manager. While the project manager often can serve as an intercessor for the project, they are often burdened with rote tasks such as ensuring that the technical integration between tasks was successful or estimating the date of project completion. Introduction of an individual, whether dedicated or not, to serve as an intercessor to facilitate the knowledge transfer process could have significant benefits.

5. IMPLICATIONS FOR SOFTWARE ENGINEERING TEAMS

Assembling and building a high performance software engineering team involves more than just selecting the “best” engineers for the job. It involves an in-depth analysis to understand the complex relationships within the team of software developers and also the broader set of extended stakeholders who may contribute to, or otherwise affect change on, the project. While often times the project manager is tasked with understanding these relationships we argue that an additional intercessor role may be more suited to the task.

In future work we seek to further examine and clarify the role of intercessors in a software organization. We are currently examining a variety of projects to develop a method to empirically identify those individuals who serve as intercessors on projects and quantify the impact of intercessors on long term project performance.

6. REFERENCES

[1] Burt, R.S. 1992. *Structural holes: the social structure of competition*. Harvard University Press.

[2] Cohen, M.D. et al. 1972. A Garbage Can Model of Organizational Choice. *Administrative Science Quarterly*. 17, 1 (Mar. 1972), 1-25.

[3] Fitzgerald, B. 2006. The Transformation of Open Source Software. *MIS Quarterly*. 30, 3 (Sep. 2006), 587-598.

[4] Galbraith, J. 1974. Organization Design: An Information Processing View. *Interfaces*. 4, 5 (May. 1974), 28-36.

[5] Grzinich, J.C. et al. 1997. Implementation of an integrated product development process for systems. *Innovation in Technology Management - The Key to Global Leadership. PICMET '97: Portland International Conference on Management and Technology* (Jul. 1997), 427-430.

[6] Hayek, F.A. 1945. The Use of Knowledge in Society. *The American Economic Review*. 35, 4 (1945), 519-530.

[7] Keynes, J.M. 1936. *The General Theory of Employment, Interest and Money*. Palgrave Macmillan.

[8] Krein, J.L. et al. 2011. The problem of private information in large software organizations. *Proceedings of the 2011 International Conference on Software and Systems Process - ICSSP '11* (Waikiki, Honolulu, HI, USA, 2011), 218.

[9] von Krogh, G. et al. 2003. Community, joining, and specialization in open source software innovation: a case study. *Research Policy*. 32, 7 (Jul. 2003), 1217-1241.

[10] Open Services for Lifecycle Collaboration: *http://open-services.net/*. Accessed: 2011-06-23.

[11] Radner, R. 1993. The Organization of Decentralized Information Processing. *Econometrica*. 61, 5 (1993), 1109-1146.

[12] Raymond, E.S. 1999. *The Cathedral and the Bazaar*. O'Reilly & Associates.

[13] Weick, K.E. and Roberts, K.H. 1993. Collective Mind in Organizations: Heedful Interrelating on Flight Decks. *Administrative Science Quarterly*. 38, 3 (Sep. 1993), 357-381.

[14] Williams, C. et al. 2010. Supporting enterprise stakeholders in software projects. *Proceedings of the 2010 ICSE Workshop on Cooperative and Human Aspects of Software Engineering* (Cape Town, South Africa, 2010), 109–112.