# A Hub-and-Spoke Model for Tool Integration in Distributed Development

Fabio Calefato
Dipartimento Jonico
University of Bari
Taranto, Italy
fabio.calefato@uniba.it

Filippo Lanubile
Dipartimento di Informatica
University of Bari
Bari, Italy
filippo.lanubile@uniba.it

***Abstract*—Today distributed development depend on an ever-growing plethora of tools that provide a continual stream of updates and place developers into a situation of channel overload and information fragmentation. In this paper, we present our initial work on the definition of a model, named hub-and-spoke, for a loosely-coupled integration of development tools that can help developers cope with these issues, while also increasing their overall situational awareness.**

***Keywords—tool integration; information fragmentation; channel overload; distributed development; awareness; devops***

## I. Introduction

Today an ever-growing plethora of different tools are needed to develop and manage distributed software projects that keep growing in both size and complexity [2]. Developing and managing software projects is not easy and doing it remotely is an even greater challenge. To enable software developers to work more effectively, other tools are often introduced, which end up causing *channel overload* [19]. The effect of more and more tools producing more and more information is placing developers into a situation of *information fragmentation* [19] and *overload* [16]. As such, the productivity of software developers is constantly undermined by a growing flow of information available at different places: API documentation to read, source code to traverse, build and deployment updates, email, RSS feeds and social media notifications, all provide a continual stream of updates that is difficult to keep track of.

Keeping up with these updates, however, is as consuming as vital because they provide developers with different information elements that are needed to keep aware of what is happening within a software project, especially if distributed. In fact, *awareness*, defined as "an understanding of the activities of others, which provides a context for your own activity" [8], is fundamental in distributed software development as it provides mechanisms to coordinate group activities [13]. According to Gutwin *et al.* [10], the members of a group typically seek information on coworkers, tasks and artifacts.

Following these information needs, four types of awareness have been acknowledged so far: (i) *informal* or *presence awareness*, i.e., who is around and their availability; (ii) *group-structural awareness*, i.e., members' roles and teams' internal structure; (iii) *workspace awareness*, i.e., who changed a shared artifact and when; (iv) *social awareness*, i.e., the understanding about existing social connections within a group [4]. Taken together, these four types help individuals build and maintain their *situational awareness*, a tem used in cognitive psychology to refer to a state of mind where a person is aware of the elements of their immediate environments. Hipikat [6], Mylyn [16] and Palantir [17] are three successful examples of tools that help developers maintain a situational awareness in software development environments [1].

In this paper, we present our initial work on the definition of a model for a lightweight, loosely-coupled integration of software development tools that can help distributed developers cope with the overload of channels and the fragmentation of information coming from their usage, while also increasing their overall situational awareness. We call this model *hub-and-spoke* because we identify a few central tools, i.e., the hubs, to which the other integrated tools are connected through the spokes. These hubs first aggregate the information flowing through the spokes and then dispatch it to either other hubs or team members after performing filtering and ranking on its elements.

The remainder of the paper is structured as follows. In Section 2 we review the background on software engineering tool integration. In Section 3, we illustrate the hub-and-spoke model in general, whereas in Section 4 we instantiate the model to illustrate a future case study. Finally, in Section 5 we present our future work.

## II. Software Engineering Tool Integration

The topic of tool integration flourished during the '90s, following the desire to produce software engineering environments that would support the entire development lifecycle through the combination of different tools, each addressing a different aspect of the development process. According to Wasserman [20], such environments combine tools along five dimensions of integration: (i) platform, (ii) control, (iii) process, (iv) data and (v) presentation. Accordingly, building an integrated environment results into

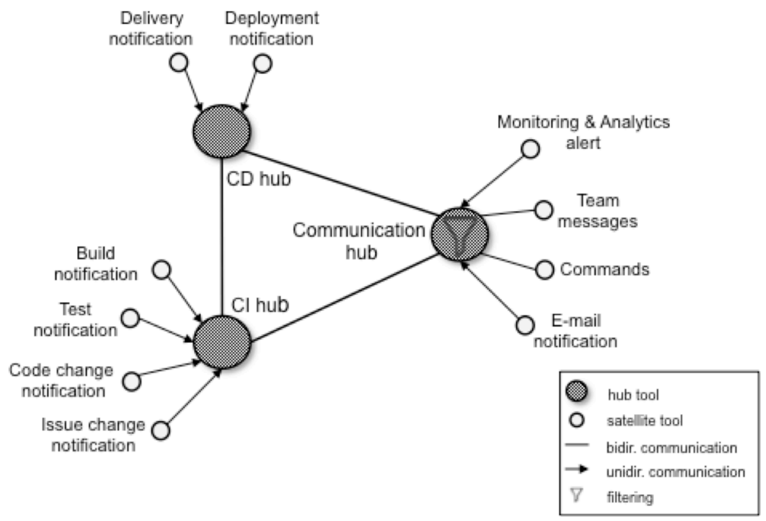

Fig. 1. The hub-and-spoke integration model

one *platform* that allows project teams to *control* the entire development *process* and its *data* through a common *presentation* layer.

An example of such tool integrations are the IDEs, that is, integrated development environments built around a source code editor, a compiler, a debugger and a build automation tool. As they integrate the core applications of a developer, IDEs have proved to be helpful to developers in avoiding the effort of switching back and forth between different applications [3]. Instead, as further tools were integrated (e.g., UML diagrams, communication tools, automated testing tools, version control systems), developers started to feel IDEs as unnecessarily bloated [18]. Besides, this form of heavyweight integration turned out to be problematic since software development tools and frameworks are generally designed for extension rather than combination [5],[14]. Thus, despite the promises of productivity and quality improvements for developers and teams as well as products and processes, research in the field faded progressively over the years, leaving unanswered questions about the benefits of or even the need for integrated environments [21].

In software engineering research, it is not uncommon to observe that a thesis proposed in one decade is replaced by its antithesis during the next one [2]. Therefore, as software first started to turn into Web services, with the diffusion of Service-Oriented Architectures (SOA), and then moved to the cloud, with the spread of the Software-as-a-Service (SaaS) delivery model, heavy weight integrations started to be replaced by a loosely-coupled integrations of multiple standalone services into a new compound one. These lightweight interconnections of services occur typically through Web APIs, either REST or RPC-like.

## III. A Hub-and-Spoke Tool Integration Model

Fig. 1 shows our model for integrating software development tools. Instead of proposing the integration along the five dimensions suggested by Wasserman [20], this model relies on the lightweight interconnections of services. In other words, the proposed model ditches the idea of building a single platform with a unified user interface, instead proposing the idea of (i) *controlling the development process* and (ii) *accessing project data* through a few central tools.

Such ‘central’ tools are called *hubs* because they act as collectors of information produced by *satellite* tools such as issue tracking and version control systems, email and IM clients. In our model, developers still have access to the satellite tools as usual, without having to use them through a unified platform. Project-related information flows through the *spokes* that connect the satellites to the hubs. In our model, we envision three types of hubs:

- Continuous Integration (CI) hub
- Continuous Delivery/Deployment (CD) hub
- Communication hub

The *Continuous Integration* (CI) hub represents the toolset that a development team adopts to support the frequent (i.e., several times a day) integration of code changes in the main line of development [9]. Typically, it is implemented through a CI server that automatically builds and tests the code base upon changes. As for the CD hub, it may refer to either *Continuous Delivery* or *Continuous Deployment*, depending on the development process adopted [12]. Continuous Delivery means that a new product release is tested in a stagin environment that is similar to that of production and, thus, it is ready to ship. Continuous Deployment, instead, indicates that, after passing quality assurance tests, the new release is actually and automatically deployed into production. Both Continuous Delivery and Deployment are natural extensions of CI because they presume that developers perform continuous integration of code changes. Finally, we note that CD and CI hubs are represented in Fig. 1 as logically separated, although in practice they might be available through one software solution (e.g., Travis[1]).

[1] https://travis-ci.org

All the three hubs are interconnected, meaning that notifications flow either way between them. As such, team-wide communication happens through the *Communication* hub, meaning that notifications from the CI and CD hubs – e.g., a failed build alert – are not dispatched to the team directly, but rather through the Communication hub. One benefit of this dispatching solution is that teams have (ideally) one place to go for checking all project events notifications (more on this later). The second benefit is the possibility of applying recommendations to notifications, that is, to filter and rank them for each developer in order to reduce the information overload [15]. Implemented by bots, these filtering and ranking operations can be tailored upon developers' activities and roles. More specifically, the information filtering and ranking may depend on developers' own *awareness network*, that is, the set of relevant teammates whose actions one monitors and to whom one's actions are displayed [7]. An awareness network is the means by which developers keep up to date their overall situational awareness about teammates, tasks and shared artifacts; as such, it is highly dynamic because the set of relevant colleagues constantly changes over time, depending on one's task assignments or the current stage of the software development process. Therefore, the relevance and priority of notifications dispatched from the Communication hub to developers will vary depending on the current configuration of their awareness network.

Furthermore, all satellite nodes are connected to one of the hubs through an arrow to indicate a unidirectional communication flow. Instead, between the Communication hub and the *Team messages* node there is a bidirectional communication. This node represents the intra-team communication tool of choice, which can receive notifications from as well as send messages to the Communication hub. Having only one tool handling all the intra-team communication avoids having separate conversation silos and relevant information split across them. Thus, all the important project information from various data sources – messages, commits and deployments notifications, performance alerts – are gathered and displayed in one place.

Finally, as for the *Commands* node, the tool selected for acting as the Communication hub should be extensible to let developers interact with the development infrastructure by simply typing commands. More specifically, this form of conversation-driven development, or *ChatOps* as GitHub popularized it [11], is intended to automate tasks, particularly Operations tasks, which become easy to the point that any member of the team can perform them by typing one command with a familiar user interface. Therefore, our hub-and-spoke model can be beneficial to foster DevOps practices in distributed projects through ChatOps. On one hand, ChatOps pushes towards the automation of delivery and deployment tasks – then, fostering CD. On the other hand, it can help organizations blur the line between the roles of Development and Operations staff and eventually eliminate the distinction – then, fostering collaboration. In fact, one common anti-pattern when introducing DevOps to an organization is to build a 'DevOps team', which conversely results in creating new silos that actually prevent DevOps collaboration [1]. Instead, our model for tool integration can help team members collaborate while improving their overall group awareness by encouraging shared responsibility between the roles of Development and Operations along the entire development process. In fact, DevOps collaboration demands making Development staff more aware of operational concerns such as system orchestration. Thus, by adopting new automation tools and practices like those of ChatOps, the Operations staff can help developers take care of a system not only during its build, but also when it is released and deployed.

TABLE I. A LIST OF TOOLS CURRENTLY EMPLOYED AT FOOBAR FOR PROJECT DEVELOPMENT

| **Category** | **Development tools currently employed** |
|---|---|
| IDE | Eclipse<br>WebStorm |
| ALM | GitLab<br>HP ALM |
| CI | Jenkins<br>TeamCity |
| CD | Ansible<br>Jenkins |
| Build manager | Maven |
| Testing | Jasmine, Karma, PhantomJS, Protractor, Selenium<br>Cucumber, Gatling, JUnit, JMeter, RestAssured<br>Capybara, Watir |
| Issue tracking | Jira |
| Package manager | Artifactory<br>Bower |
| Database | Liquibase |
| Task automation | Grunt |

## IV. THE INSTANTIATED MODEL

Foobar, a fictional name used hereafter to grant anonymity, is a medium-sized software company that works in the publishing industry. The company is distributed across Europe and the USA, and it has recently opened a development site in Italy. The company is facing channel-overload and information-fragmentation issues due to the large set of tools that developers use in their projects. In addition, the company is looking for opportunities to spread DevOps practices.

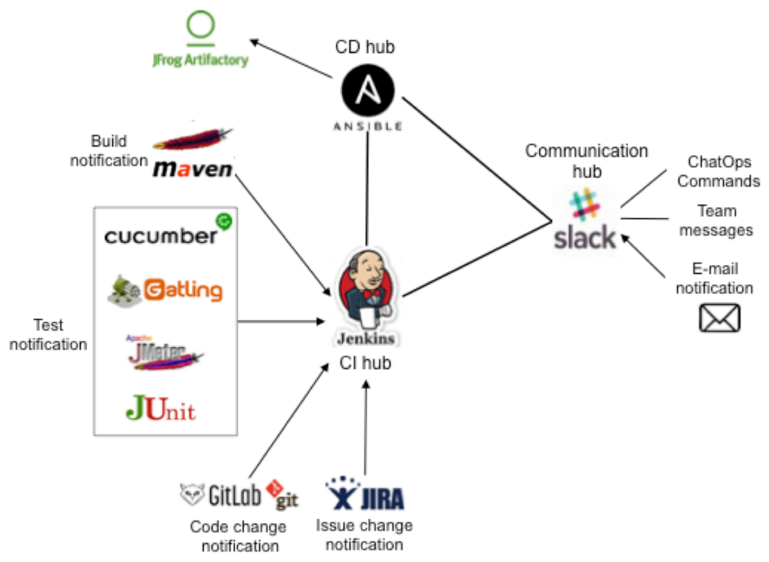

Fig. 2. The instantiated hub-and-spoke model

As shown in Table I, there are many software tools currently employed at Foobar, all of which generate a growing flow of information that both developers and managers find hard to keep up with. The company has adopted an agile development process and heavily relies on task automation, as witnessed by the extensive use of testing frameworks and CI/CD tools.

In Fig. 2, we show the hub-and-spoke integration model instantiated with respect to the needs of the Foobar company. Although the Italian site contributes to projects written in three main programming languages (Java, JavaScript and Ruby), the instantiated model refers in particular to the case of Java projects. For JavaScript and Ruby projects, the testing frameworks would change accordingly. In addition, in Fig. 2 we represent the case where Jenkins[2] is used as CI server, but our approach described next would not change for projects where TeamCity[3] is used instead

The instantiated model proposed here builds around the extensibility of the tools selected to act as the hubs; Jenkins, Ansible[4] and Slack[5]. These tools offer a great degree of extensibility and a large set (hundreds) of plugins already available. As shown above, Jenkins collects all the event notifications coming from the continuous integration of software changes, whereas Ansible handles the events related to the deployment of new releases to the Artifactory repository. These integrations are implemented by adding the related plugins to the CI/CD servers (e.g., the GitLab plugin) and configuring the location where resources are hosted (e.g., the Git repository URL from which to retrieve the code to build and test) or need to be published (e.g., the Artifactory repository location).

As for implementing the communication hub, we propose the adoption of Slack to achieve the goal of controlling the development process and accessing data from one central place. Albeit not currently used at the Foobar company, the variety of Slack APIs allows collecting both notifications from tools and developers, as well as implementing conversation-driven development through ChatOps commands. More specifically, Slack offers the following APIs, all of which enforce security by requiring authentication through app-specific access tokens or OAuth2 protocol:

- Webhooks API, allows the processing of both *outgoing* and *incoming* messages, respectively, sent and received as HTTP requests with a JSON payload using specific URLs.
- Slash Commands API, allows the definition of custom commands (starting with a slash and followed by parameters, e.g., `/command params`), which trigger actions that enable users to interact with external services directly from Slack.
- Real Time Messaging API, a WebSocket-based API that allows receiving events from Slack in real time and send messages through it as well.
- Bot Users API, uses the Real Time Messaging API to build bots that are useful to monitor incoming messages and react to them.
- Web API, an HTTP RPC API to build applications that require interacting with Slack in more complex ways than those possible with the other APIs.

In particular, we intend to use the Slash Command API to implement custom ChatOps commands that would help, for example, the developer staff take care of the deployment (e.g., `/deploy project-name location`), force a new build (e.g., `/build project-name`) or re-execute test suites even without changes to the code repository (e.g., `/test project-name alltests`). In general, we foresee to add several commands to control the various phases of the development process.

[2] https://jenkins-ci.org
[3] https://www.jetbrains.com/teamcity
[4] http://www.ansible.com
[5] https://slack.com

Furthermore, Jenkins and Ansible are going to be integrated with Slack using the Webhooks API, thus defining a publish/subscribe communication model for dispatching notifications to and from the Communication hub. Regarding the Real Time Messaging API, it will be used to route messages that are relevant for the project coming from other communication sources (e.g., email clients). As for the Bot Users API, it will be used to implement automatic reaction to certain messages. For example, a bot can proactively respond to performance alerts of an application deployed to the cloud and automatically add new nodes to handle the increased traffic load. Besides, bots would enable the filtering and prioritization of notifications. In fact, Slack is intended to collect all the project- and team-related communications, it is soon going to be overloaded with them. In addition, Slack has a limited support for filtering, as it allows the definition of channels, similar to IRC chat rooms where information can be categorized, and silencing notifications unless one's recipient (in the form of `@username`) is specified in the message body. Thus, bots can implement more sophisticated recommendations based of the severity of the messages and the structure of the development team. For example, a team manager may not want to be notified of all new bug reports, but is definitely willing to receive a prompt alert in case of crash reports. Finally, we are going to use the Web API to create tool integrations other than those listed above as we see fit.

Finally, we note that the instantiated model that we just presented might be extended with other hubs that fulfill future needs of the company, as long as they can be connected to the communication hub.

## V. Conclusions & Future Work

In this paper, we have presented a model for a lightweight, loosely-coupled integration of software development tools. We argued that our hub-and-spoke model can help developers fight the channel-overload and information-fragmentation problems while also increasing at the same time the awareness of the elements in their working environment. As future work, first we intend to complete the implementation of the instantiated model and, then, we will conduct a case study to test whether and how the hub-and-spoke model work in large distributed software projects.

## Acknowledgments

This work fulfills the objectives of the PON&RC project "Digital Services Ecosystem (DSE)" and it is also partially funded by the project 'Investigating the Role of Emotions in Online Question & Answer Sites' under the program "Scientific Independence of young Researchers" (SIR). Both projects are funded by the Italian Ministry of University and Research (MIUR).

## References

[1] O. Baysal, R. Holmes, and M.W. Godfrey. "Situational awareness: personalizing issue tracking systems." In *Proc. of Int'l Conf. on Software Engineering (ICSE '13)*. 2013, pp. 1185-1188.

[2] B. Boehm. "A view of 20th and 21st century software engineering." In *Proc. of 28th ACM Int'l Conf. on Software* engineering *(ICSE '06)*, 2006, pp. 12-29, DOI=10.1145/1134285.1134288

[3] G. Booch and A.W. Brown. "Collaborative Development Environments." *Advances in Computers*, Vol. 59, 2003, pp. 1–27, DOI= 10.1016/S0065-2458(03)59001-5

[4] F. Calefato and F. Lanubile, "Augmenting Social Awareness in a Collaborative Development Environment." In *Proc. of 5th Int'l Workshop on Cooperative and Human Aspects of Software Engineering (CHASE '12)*, 2012, pp. 12-14, DOI=10.1109/CHASE.2012.6223009

[5] F. Calefato and F. Lanubile. "Using Frameworks to Develop a Distributed Conferencing System: An Experience Report." *Software: Practice & Experience*, Vol. 39, No. 15, 2009, pp. 1293–1311, DOI=10.1002/spe.937.

[6] D. Cubranic, G.C. Murphy, J. Singer, and K.S. Booth. "Hipikat: a project memory for software development." *IEEE Transactions on Software Engineering*, Vol. 31, No. 6, 2005, pp. 446-465, DOI=10.1109/TSE.2005.71

[7] C.R.B. de Souza and D.F. Redmiles "The Awareness Network, To Whom Should I Display My Actions? And Whose Actions Should I Monitor?" *IEEE Transactions on Software Engineering*, Vol. 37, No. 3, 2011, pp. 325-340, DOI=10.1109/TSE.2011.19

[8] P. Dourish and V. Bellotti. "Awareness and coordination in shared workspaces." In *Proc. of ACM Int'l Conf on Computer-supported cooperative work (CSCW '92)*. 1992, pp. 107-114. DOI=http://dx.doi.org/10.1145/143457.143468

[9] P.M. Duvall, S. Matyas, and A. Glover "*Continuous Integration: Improving Software Quality and Reducing Risk*.". Pearson Education, 2007

[10] C. Gutwin, S. Greenberg, and M Roseman. "Workspace Awareness in Real-Time Distributed Groupware: Framework, Widgets, and Evaluation." In *Proc. of HCI '96: People and Computers XI*, 1996, pp 281-298, DOI= 10.1007/978-1-4471-3588-3_18

[11] G.V. Hulme. "ChatOps: Communicating at the speed of DevOps." Last accessed: Feb. 3, 2016, http://devops.com/2014/07/16/chatops-communicating-speed-devops/

[12] J. Humble and D. Farley. "*Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation.*" Addison-Wesley Professional, 2010.

[13] F. Lanubile, F. Calefato, and C. Ebert, "Group Awareness in Global Software Engineering." *IEEE Software*, Vol. 30, Issue 2, 2013, pp. 18-23, DOI=10.1109/MS.2013.30

[14] M. Mattsson, J. Bosch, and M.E. Fayad. "Framework integration problems, causes, solutions." *Communications of the ACM*, Vol. 42, No. 10, 1999, pp. 80–87, DOI=10.1145/317665.317679

[15] M.P. Robillard, R.J. Walker, and T. Zimmermann. "Recommendation systems for software engineering." *IEEE Software,* Vol. 27, No. 4, 2010, pp. 80-86, DOI=10.1109/MS.2009.161

[16] G. Murphy, "Attacking Information Overload in Software Development". *In Proc. of IEEE Symposium on Visual Languages and Human-Centric Computing. (VL/HCC '09)*, 2009, DOI=10.1109/VLHCC.2009.5295312

[17] A. Sarma, D.F. Redmiles, and A. van der Hoek, "Palantir: Early Detection of Development Conflicts Arising from Parallel Code Changes." *IEEE Transactions on Software Engineering*, Vol. 38, No. 4, 2012, pp. 889-908, DOI=10.1109/TSE.2011.64

[18] J. Sonmez. "Why The IDE Has Failed Us." Last accessed: Feb. 5, 2016, http://simpleprogrammer.com/2010/08/03/why-the-ide-has-failed-us/

[19] M.A. Storey, L. Singer, F. Figueira Filho, A. Zagalsky, and D.M. German, "How Social and Communication Channels Shape and Challenge a Participatory Culture in Software Development" *IEEE Transactions on Software Engineering* (to appear).

[20] A.I. Wasserman. "Tool integration in software engineering environments" In: Lecture Notes in Computer Science, Vol. 647, 1989, pp. 137–149, DOI= 10.1007/3-540-53452-0_38

[21] M.N. Wicks and R.G. Dewar. "A new research agenda for tool integration." *Journal of Systems and Software*, Vol. 80, No. 9, 2007, pp. 1569-1585, DOI=10.1016/j.jss.2007.03.089

[22] R. Wilsenach. "*DevOps Culture.*" Last accessed: Jan. 30, 2016, http://martinfowler.com/bliki/DevOpsCulture.html