Analysis of the Benefits the Peer-to-Peer Paradigm brings to Distributed Agile Software Development

Patrick Mukherjee[1], Aleksandra Kovacevic[2], Andy Schürr[1]

[1]Real-Time Systems Lab, Technische Universität Darmstadt
Merckstr. 25, 64289 Darmstadt, Germany
mukherjee, schuerr@ES.tu-darmstadt.de

[2]Multimedia Communications Lab, Technische Universität Darmstadt
Merckstr. 25, 64289 Darmstadt, Germany
sandra@KOM.tu-darmstadt.de

Abstract: In this work, we analyze the potential benefits of taking a peer-to-peer approach to supporting tools for distributed agile software development. Current tools that support software development are client/server based and designed for on-site development. We discuss their drawbacks and show how a peer-to-peer based approach can overcome them. Additionally, we present a peer-to-peer based tool, ASKME, that supports distributed agile software development and focuses on its crucial needs, namely communication and awareness among distributed developer teams.

1   Introduction

The success of agile methods in software engineering made them attractive to a wide range of developers, all focused on getting fast results from the reduced overhead of the classical software development process. Most software projects are developed by distributed teams, and taking into account that agile methods like pair programming and frequent scrum meetings are meant for collocated developer teams, the following question arises: can agile methods be applied in distributed software development? An online survey [ISHGH07] showed that 33% of the observed distributed development projects consisted of teams with less than ten members. Having small-sized teams, like having direct communication between team members, plays a crucial role in the application of agile methods. There is an obvious need to compensate for the inability to meet face-to-face; coming from the field of software engineering, we try to confront this challenge with a tool-oriented approach.

Distributed software development is currently supported by traditional tools, which are created for on-site development. Such tools are based on the client-server communication paradigm. However, the centralized architecture of client/server systems is orthogonal to the natural structure of distributed software development. Developer teams in the same location normally communicate with one another and manipulate the same artifacts. This communication (including message exchange and file transfer) obviously does not need a remote server and should be processed directly between developers.

In this paper we discuss the benefits of using a peer-to-peer communication paradigm for agile software development methods. Therefore, we first identify some key concepts of agile development methods in Section 2 and then, in Section 3, we discuss the problems that the client/server approach brings. In Section 4, we show why peer-to-peer based tools would overcome the previously discussed limitations and in Section 5, we present a peer-to-peer based tool called ASKME that supports agile software development. This tool focuses on the need for developer awareness in distributed teams and for communication that does not use a central server. Finally, we conclude our work in section 6.

2   Properties and Needs of Distributed Agile Software Development

Agile development methods aim to get fast results. To achieve this, the actual implementation phase is emphasized and all other phases (e.g. requirements analysis, design, documentation) are reduced to a minimum time consumption. In contrast to classical development methods like the Rational Unified Process [RUP], agile methods are people-centric rather than process oriented. In fact, the design of the software develops not only during the actual coding, but also during the requirements analysis with the customer. Existing tools that support agile development focus on reducing the duration of non-coding phases. Automatic documentation generators (e.g. JavaDoc) or automatic regression tests (e.g. JUnit) reduce documentation and testing phases, respectively. However, the actual need for documentation is weaker than the need for communication because teams in agile development consist of a few highly competent developers who share their project knowledge directly, i.e., face-to-face [ea01]. Most modern software development is distributed, often even over continents. As such makes the typical agile principle of face-to-face knowledge exchange impossible, a tool to support direct *communication* is needed. Besides basic communication, enabling *awareness* is crucial - because the participants are not able to physically see each other, it is important that their presence in the development environment is visible. As soon as a participant appears online, similar to hallway conversations, other participants are reminded about a deferred conversation, etc.

3   Drawbacks of Client/Server Based Solutions

In the client/server communication paradigm, all data exchanges occur between one dedicated machine (the server) and the machines of the team members (the clients). The server must always be available to offer services while the clients only consume the services. Client/server based support for distributed agile development has numerous drawbacks. First of all, it obviously creates a *communication bottleneck* due to the fact that all data exchanges have to be transmitted through a server, often making communication slow, with poor scalability. If two clients, which might be geographically close to each other, want to exchange information, they cope with *additional latency* introduced by a distant server. Furthermore, *suboptimal resource consumption resources* result from exploiting a

server for all services where client machine resources (such as computation power, memory, bandwidth, and storage space) typically remain unused. If the server fails, the whole system stops working and data could be lost, making the server a *single point of failure*. Finally, the client/server approach has *high set-up and maintenance cost*, which is a particular issue for non-profit projects (e.g. open source projects) as they are normally funded by unreliable donations.

4   Motivation for a Peer-to-Peer Approach

In a peer-to-peer paradigm, each communication participant (peer) offers and consumes services. All communication and data exchange is done directly between intended parties. The *significantly lower costs* of peer-to-peer solutions are the result of *self-organization* and *optimal resource usage* from all participants. Because all users maintain their own machines, there are no additional maintenance costs. If a peer fails, it will be replaced automatically; thus good maintenance is not needed. When a peer comes online he has to be able, in a short time, to overtake services offered by his neighbors; thus the applications must be designed to have a short start-up time. Because each peer is only responsible for a part of a service, the start-up time is a part of the time the combined service would take to start on a single server. [SE05]

In contrast to the client/server approach, messages in peer-to-peer communication have *decreased delay*. Two peers can communicate directly with each other. In the case of client/server communication, if two geographically close clients (e.g. in the same room) want to exchange data, they have to send the data to a distant server (e.g. in another city) and experience unnecessary delay. Only the initialization of the communication requires peer-to-peer routing, which is likely to introduce some delay due to the stretch factor [RS07]. But subsequent messages can be exchanged directly, which is shorter than any three-folded way using the client/server approach[1].

One of the key characteristics of peer-to-peer systems is *robustness*, meaning that a peer failure is considered normal and does not cause a system break. The system would only be harmed if a peer and all its replicating peers (usually five peers altogether) crashed simultaneously, before one were able to replicate its state to a new peer. However, even in this unlikely scenario, only a part of the system would be harmed; the system as a whole would remain functional.

---

[1]Only in the case that the server were on the routing path would the delay be the same.

5 A Peer-to-Peer based Support Tool for Agile Development: ASKME

As an example how peer-to-peer based applications can improve agile distributed software development, we implemented a tool that aims to improve *awareness* and communication among distributed developer teams; we call this tool **ASKME**, 'Awareness Support Keeping Messages Environment'.

5.1 Functionality

The implemented application is designed as a plug-in for the widely used development environment Eclipse. In a side-panel, a user can add his team mates as contacts by entering their nicknames. The tool allows users to track the status of their colleagues, e.g. wether one is absent or free to respond.

The best source of information while having problems with collaborative coding is to exchange information directly, according to the agile principles. That means contacting the developer who modified the code most recently. Exactly that developer gets automatically highlighted once a document is opened, which is stored by an integrated version control system[2]. The user will be made aware of the contact and reminded that he can ask him questions. If the contact is offline a users can still send messages, which will be automatically delivered by other peers as soon as the destination appears online. In existing instant messengers, such a message would arrive only when both communicating parties are online, meaning that the message could be delayed as long as the sender is offline.

5.2 Architecture

To gather context information about participants' presence, our instant messenger application is built as a plug-in for Eclipse. Using methods to access the version control repository from Eclipse, version information can be accessed (e.g. the last author of a document). It is also possible to access other kind of information in the development environment like contextual (e.g. chatting with bob, working on document X) or user-specified (e.g. busy).

For message exchange, ASKME uses the structured, DHT-based peer-to-peer overlay network *pastry* [RD01], namely its implementation *freePastry* [FP]. When a message is sent it gets routed to the contact nearest the destination the sending peer knows, as described in [RD01]. This peer will automatically forward the message until the destination peer is reached. If the destination peer is offline, our algorithm handles the delivery as follows: due to pastry's routing, the last receiver is nearest to the destination; this receiver will replicate the message to his neighbors, which are the next nearest peers to the intended destination. If peers are changing, the message will be replicated via the standard methods in structured peer-to-peer systems, thus ensuring that it will be stored by the peers surrounding the offline destination. If the destination peer finally comes online, the message gets replicated to it. Upon recognizing that the message was addressed to him, the destination peer will remove it from all replicating peers. We have chosen to replicate

---

[2]We are currently working on a peer-to-peer based version control solution.

offline messages on five neighbors. The message could only get lost if all five were to fail at the same time, which is very unlikely ([RD01]). Fail means that a peer leaves without being able to clean up its connections, that is, to handover stored offline messages to its neighbors and to announce to all contacts that it is going offline. In the case that both communication partners are online, the first message will reach its destination and establish a direct connection by informing both peers about the others IP address. Thus all following messages in a conversation can be exchanged without peer-to-peer-layer routing.

The implemented application could have been built using client/server based communication, but that would have introduced the drawbacks described in section 3, namely the dependency on a single machine, which would make the system less robust, and which would represent a bottleneck, which would reduce the overall communication bandwidth and increase the delay.

6 Conclusion

When applying agile methods to distributed software development using a client/server based approach, numerous issues arise, the main being bottlenecked communication channels, time consuming setup, and maintenance work. In this paper we discussed how peer-to-peer based tools address nearly all of these problems and how they meet the main needs of agile software development - communication and awareness. As an example, we implemented a peer-to-peer based instant messenger as an Eclipse plug-in, with additional features that support agile teams in their communication and aid in building awareness.

References

[ea01]   Beck et al.   Manifesto for Agile Software Development.   `http://agilemanifesto.org/`, 2001.

[FP]   FreePastry. `http://www.freepastry.org/FreePastry/`.

[ISHGH07] T. Illes-Seifert, A. Herrmann, M. Geisser, and T. Hildenbrand. The Challenges of Distributed Software Engineering and Requirements Engineering: Results of an Online Survey. In *Proc. of GREW*, pages 55–66, 2007.

[RD01]   A. Rowstron and P. Druschel. Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems. In *Lecture Notes in Computer Science*, pages 329–350, 2001.

[RS07]   A. Richa and C. Scheideler. Overlay Networks for Peer-to-Peer Networks. In *Handbook of Approximation Algorithms and Metaheuristics*, chapter 72. May 2007.

[RUP]   Rational Unified Process.   `http://www-306.ibm.com/software/awdtools/rup/`.

[SE05]   Ralf Steinmetz and Klaus Wehrle (Eds.). *Peer-to-Peer Systems and Applications*. Springer, Sep 2005.