Towards an Infrastructure for Distributed e-Business Projects

Filippo Lanubile

Abstract--**Traditional software engineering can be helpful to avoid that e-Business projects run out of time, exceed budget, and deliver poor quality applications. However, the variety of technologies and the high pace of technological change make it difficult to find the knowledge and skills required for developing a large e-Business application. This problem can be solved by geographical distribution of development teams, analogously to Open Source Software (OSS) projects that have much contributed to the spreading of the Internet. We present process and product-oriented methods that supply an infrastructure for managing cooperative work in distributed development of e-Business applications.**

Index Terms—**distributed software development, Internet-based collaboration infrastructure, open source software projects, software inspections.**

I. Introduction

The goal of software engineering is to produce software that works reliably, it is easy to use and maintain, and arrives within budget and on time. To achieve this goal, the software engineering field has proposed process and product-oriented technologies that have much contributed to satisfy the increasing demand for big and complex software systems. Among the many application domains covered by software engineering, the development of e-Business software faces extreme challenges by heterogeneous and fast changing technologies. Today, a common e-Business application might employ most of the following technologies:

- HTML and CSS for web page rendering,
- DOM and JavaScript for enabling dynamic clients,
- ASP, JSP, or PHP for enabling dynamic generation of web pages,
- Java servlets, JavaBeans or ActiveX components for reusing functional building blocks,
- IDLs for exposing the services of wrapped legacy systems,
- SQL-based APIs for interfacing corporate relational databases,
- XML for data interchange, and XSLT for data transformation.

Building an e-Business application on the basis of the above technologies requires working experience with different programming paradigms (imperative, declarative, rule-based, component-based); confidence with markup languages, declarative languages, scripting languages, and object-oriented languages; knowledge of properties/methods/events of reused components. Even worse, all of these technologies are subject to frequent changes, leading to new language versions and upgraded component interfaces.

In order to access human resources and competencies not available at home, many e-companies today are assessing the option of distribute their development across multiple sites and even multiple countries. Global software development [7] is a phenomenon started in the early 1990s, mainly driven by telecommunication companies such as Motorola or Lucent Technologies, and motivated initially by the desire to cut personnel costs and access to new markets in other countries.

The main approach to distributed software development has been the subcontracting of chunks of software development projects (decomposed by functionality, customization for local market, development stage, or maintenance stage) to subsidiaries or third-party companies [11]. Early on, companies discovered that global software development incurred its own overhead costs and time delays, because of communication and coordination issues. Nevertheless, searching for skills across geographical boundaries, and integrating groups from mergers and acquisitions are the two main forces pushing to globally distribute software development (these forces also provide the reason for distributed development of e-Business software).

Software development is mainly a teamwork activity but distance affects the degree of interaction and this may result in reduced productivity and higher development time [8]. Participants at the different development sites often suffer inhibited communication and coordination because of the distance. Distances need not to be many kilometers to severely affect communication. Just being in another building or on a different floor of the same building drops the frequency of communication (especially informal communication) to nearly the same low level as people with offices spread over different countries.

Internet ubiquity makes it possible to reach skilled people everywhere but it does not explain how a large group of geographically dispersed people can effectively interact to produce highly reliable software.

Open-source software (OSS) projects provide the evidence of successfully distributed projects that use standard Internet applications for sustaining teamwork. Section 2 of this paper summarizes and discusses the process and product-oriented infrastructure adopted in OSS distributed projects

Distributed software development also challenges

Filippo Lanubile is with the Dipartimento di Informatica, University of Bari, via Orabona 4, 70126 Bari, Italy (e-mail: lanubile@di.uniba.it).

traditional verification techniques of software engineering, such as software inspections, and asks for new solutions. Section 3 presents an ongoing project that aims to provide an infrastructure for distributed software inspections over the Internet.

II. OSS Project Infrastructure

According to its definition, open-source software (OSS) is software for which the source code is distributed or accessible via the Internet without charge or limitations on modifications and future distribution by third parties [13]. OSS development, characterized by availability of the source code and openness to contributions from the community, has its roots in the ARPANet, Unix software and Free Software Foundation's GNU. However, it was in the 1990s that OSS development became popular and synonymous with highly distributed development characterized by frequent iterations, thanks to projects such as Linux, Apache, Perl, PHP, and others. Today, there are hundreds, if not thousands, of open-source projects that are currently under development.

The main reason for this success has been the growth of the Internet, which made cooperation between distant programmers feasible on a scale much larger than was possible before. With the ubiquitous availability of the Internet, a huge base of potential developers and testers became available to create "virtual software projects". However, to deal with the issue of coordination among their many and widespread contributors, OSS projects evolved their own infrastructure.

One of the most important characteristics of distributed software development is that developers cannot any more rely on face-to-face meetings, but have to make use of technology to allow them to communicate over distance. There is a great number of Internet-based groupware tools offering both synchronous (chat, instant messaging, audio/video conferencing) and asynchronous tools for communication, and workflow management systems for coordination. However, OSS projects primarily rely on asynchronous tools for almost all communication and coordination services:

- mailing lists (sometimes archived to web sites or gatewayed to web-based discussion forums or newsgroups), to keep everyone informed as to the project status, discuss about the direction of the project, and even report bugs and contribute with new or changed code;
- version and configuration management systems (typically CVS to manage the code repository), to control remote access to source code and prevent change collisions (the 'lost update problem');
- bug tracking systems (like Apache's GNATS, Linux kernel's Jitterbug, or Mozilla's Bugzilla), to track problems and report bugs;
- web portals, offering web-based access to all services (including general information, FAQs, news, to-do lists, and native services listed above).

There are some reasons for choosing general-purpose media such as email readers and web browsers as the prevalent client-side communication and coordination tools:

1. Email and web are the most popular Internet applications; using the lowest common denominator for Internet-based communication significantly increases the chances for potential contributors to start participating, or even just tracking progresses and following project's development discussions.
2. The geographic dispersion of OSS projects with widely dispersed time zones (developers can be located in different countries and across continents) practically prevents the usage of synchronous communication.
3. Developers' contributions to OSS projects is not uniform and changes over time, depending on their interest and other commitments. Giving this situation, it would be arduous to enforce a prescriptive coordination technology, such as workflow management systems. Thus, most OSS projects rather continue using email for coordination, even after they have got bigger than the original small group of core developers.

Since most everything happens using email, OSS communities have developed development processes, which work as "social contracts" among participants to an OSS project. For example, in the Apache HTTP server project [1], there are guidelines for programming style, accessing the source repository via CVS, editing problem reports, and managing the project itself. There is a layered organization based on meritocracy [5] (the more contribution you have provided, the more you are allowed to do):

1. A group of core people, called the "The Apache Group", focuses on process rules and strategic issues. The group was formed from the project founders but it has changed over time to include volunteers who have been long-time contributors to the project. Active members have the right to cast a binding vote on a issue (such as code changes) proposed on the mailing lists.
2. A group of contributors, called "The Apache Developers", with "commit" access to the CVS repositories of source code. They receive contributions, review them, and integrate the accepted ones into the code repository. Over time, developers who have distinguished themselves by the quality and frequency of their work may be invited to join the Apache group and gain more responsibility in the project.
3. Everyone else, who can access to the CVS snapshots, and participate in the mailing lists for reporting problems and proposing changes. Volunteers who want to become Apache developers need to ask for it on the mailing list and, if approved, get a user account and write access to the source base.

An average of about 40 messages a day flow over mailing lists, with discussions on new features, bug fixes, user problems, community news, release dates, etc. The actual code

development takes place on the volunteers' local machines, with proposed changes communicated using a "patch" (the set of differences between the current and proposed versions of the code), and committed to the source repository by one of the Apache developers using remote CVS. Doubtful changes, new features, and large-scale renovations need to be discussed on the mailing list before being committed to a repository. Consensus approval needs a majority with a minimum quorum of three positive votes by active Apache group members.

There are some individual differences among OSS projects: in Linux, for example, the last word on what goes into the kernel rests with Linus Torvalds, although the responsibility over subsystems has been delegated to his lieutenants, called "module maintainers" (who have actually developed most the code they manage). However, whatever development process has been established, read-only access to documentation and source code is granted to anyone on the Internet, while only the core developers are responsible for evaluating and approving changes submitted by the community and integrating them into the source code base.

In its seminal paper [15], Eric Raymond discusses some theories implied by the history of OSS projects. Among these, there is so-called "Linus's Law": "Given enough eyeballs, all bugs are shallow". The law is based on the observation that with a large enough base of beta-testers and co-developers almost every problem shows up, and the fix appears clear to someone. The law can be restated as "Debugging is parallelizable", where the implication is that the process is scalable because it does not require coordination among contributors acting as debuggers, but it only needs one-to-one communication between each debugger and a coordinator, responsible for the fix. Thus, even if the use of the mailing list as a primary communication channel among the developers can eventually overwhelm the resources of their contributors (especially would-be contributors), OSS projects have been successful to keep pace with commercial rivals.

Even if the bazaar style of OSS projects might not be considered appropriate for industrial organizations, there is enough that can be learnt to inspire new collaboration infrastructure to assist E-distributed projects.

III. The Internet-Based Inspection System

We propose the Internet-Based Inspection System (IBIS) to support scalable and distributed software inspections. The worldwide distribution of the Internet provides the necessary transport and application-level infrastructure. However, the process used for classical software inspections does not scale up to large inspections. The IBIS adopts a reengineered inspection process to minimize coordination problems and a lightweight architecture to achieve the maximum of simplicity of use and deployment.

A. Software Inspection

Software inspections are a software engineering "best practice" for detecting and correcting defects in software artifacts, including requirements, design, code and test cases. Because defects are detected early in the software development process, software inspection improves software quality and reduces rework, thus saving time and costs.

Inspections are carried out by a small group that provides a feedback for authors. Software inspections are distinguished from other types of peer reviews in that they rigorously define:

- a phased process to follow (Fig. 1);
- roles performed by peers during review (e.g., moderator, author, recorder, reader, and inspector);
- a reading toolset to guide the review activity (e.g., defect taxonomies, product checklists, and scenario-based reading techniques);
- forms and report templates to collect product and process data.

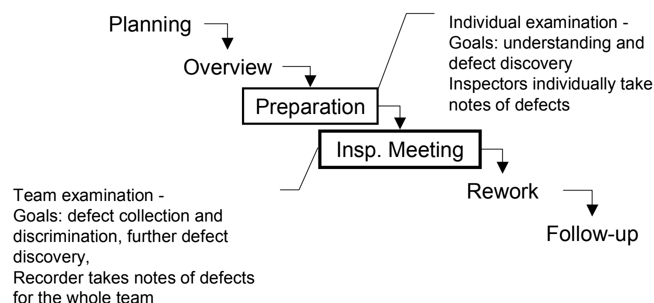

Fig. 1. Classical inspection process

Software inspections were developed first by Michael Fagan at IBM [4] and then adopted in many variants [10] by organizations such as Ericsson, Hewlett-Packard, Lucent Technologies, Motorola, and NASA. However, software inspection has never become a mainstreaming process among industry practitioners, mainly because of clerical overhead and time bottlenecks. Furthermore, traditional inspection processes cannot be simply adapted for being included into "offshore development", where large, permanent companies are replaced by temporary groups of developers collaborating on projects over the Internet. Because of its roots on manual activities and face-to-face team interactions the inspection process must be first reengineered and then supported by some Internet-mediated environment. The challenge is to scale up the process and make it distributed, without loosing the advantages of inspections (e.g., phased process, reading toolkits, roles, measurement) over less formal review processes.

B. *Reengineering Inspection*

In the attempt to shorten the overall cost and total time of the inspection process, the need for a meeting has been empirically investigated by many independent studies together with other factors affecting inspection effectiveness [2, 14]. In a recent article [16], Sauer et al. consider a reorganization of the inspection process on the basis of behavioral theory of

group performance.

The alternative design for software inspections mainly consists of replacing the preparation and meeting phases of the classical inspection process with three new sequential phases: defect discovery, defect collection and defect discrimination (Fig. 2).

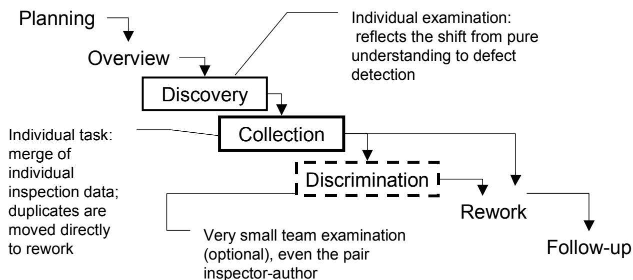

FIG. 2. THE REENGINEERED INSPECTION PROCESS

The first, defect discovery, reflects the historical shift of goal for the preparation phase, that has changed from pure understanding to defect detection, and so inspectors have to individually take notes of defects. The other two inspection phases are the result of having removed the goal for team activities of finding further defects and then separating the activities of defect collection (putting together defects found by individual reviewers) from defect discrimination (removing false positives). Collection independent of defect discrimination only requires either the moderator or even the author himself, with the addition of an automatic support for merging individual inspection data, thus eliminating the phenomenon of collection losses. Defect collection could also include identifying and marking for rework without further discussion any duplicate defects found, thus saving time. Defect discrimination may either be skipped, passing the collected defects directly to the author for rework, or the number of reviewers may be reduced to those that can be recognized as experts, on the basis of the analysis of individual findings. The behavioral theory and empirical findings suggest that a single expert reviewer paired with the author may be as effective in the discrimination task as larger groups.

The reengineered inspection process makes it possible to employ a large number of parallel inspectors focusing just on defect discovery, thus increasing the probability of detecting "hard to find defects". A high number of inspectors has an effect on cost but not on interval time, and will not pose coordination problems: defects found by more than one inspector are moved directly to the author and defects found by just one inspector are discussed by smaller groups, even the pair inspector-author.

C. IBIS Architecture

In order to achieve the maximum of simplicity of use and deployment, we have chosen a lightweight approach for the architecture of the IBIS:

- On the client side,
  the system uses common Internet-based application clients such as browsers, email readers, and (optionally) an audio-video plug-in. It does not require any additional installation, neither manual nor automatic.
- On the server side,
  the system uses only technologies based on Internet standards or web standards (including the XML family of technologies), as specified respectively by the Internet Engineering Task Force (IETF) and the World Wide Web Consortium (W3C).
  It does not require any DBMS for the data repository. All structured and persistent data are stored as XML files, programmatically accessed via the DOM API, and automatically manipulated by XSL transformations.
  It does not require any workflow management system or groupware platform. All groupware functionalities are developed from dynamic web pages on the basis of scripts and server-side components, when available; event notification is achieved by using the email service of a server-side component, which transfers generated messages to an outcoming SMTP server.

Fig. 3 shows the UML deployment diagram of the IBIS. The current configuration of the IBIS is the following:

- Modern browsers to render HTML 4.0 web pages with presentation effects specified by CSS1 and CSS2 properties (currently Microsoft Internet Explorer 5, Netscape 6, and Opera 5).
- Web and application servers based on Microsoft (MS) Internet Information Server (IIS), including
  - Active Server Pages (ASP) components with server-side scripts to implement the business logic and dynamically generate web pages;
  - the dynamic link library (DLL) MSXML3 to parse XML data, manage the tree representation via DOM, and execute XSL transformations;
  - the DLL CDONTS (Collaboration Data Objects for NT Server) to generate notification mail messages;
  - the DLL CPSHOST to upload new or changed documents through the HTPP POST method;
- Mail server based on MS SMTP Mail Service (an IIS feature for a virtual SMTP server).

As an optional adjunct for managing multimedia information in the overview stage, the current configuration of the IBIS is the following:

- RealNetworks's RealPlayer G2 as a browser plug-in to interpret streaming information
- MS PowerPoint and RealNetworks's RealPresenter G2 to produce and synchronize streaming information
- RealNetworks's RealServer G2 as a streaming server

At the project startup, the functional requirements of the IBIS were based heavily upon a classical inspection process as defined by a NASA guidebook [12]. Other than providing an

automated support for clerical activities, the major departure from the NASA guidebook was that inspections should not be co-located, and then face-to-face meetings could be replaced by various forms of distance meetings (synchronous or asynchronous inspection) or removed at all, depending on circumstances.

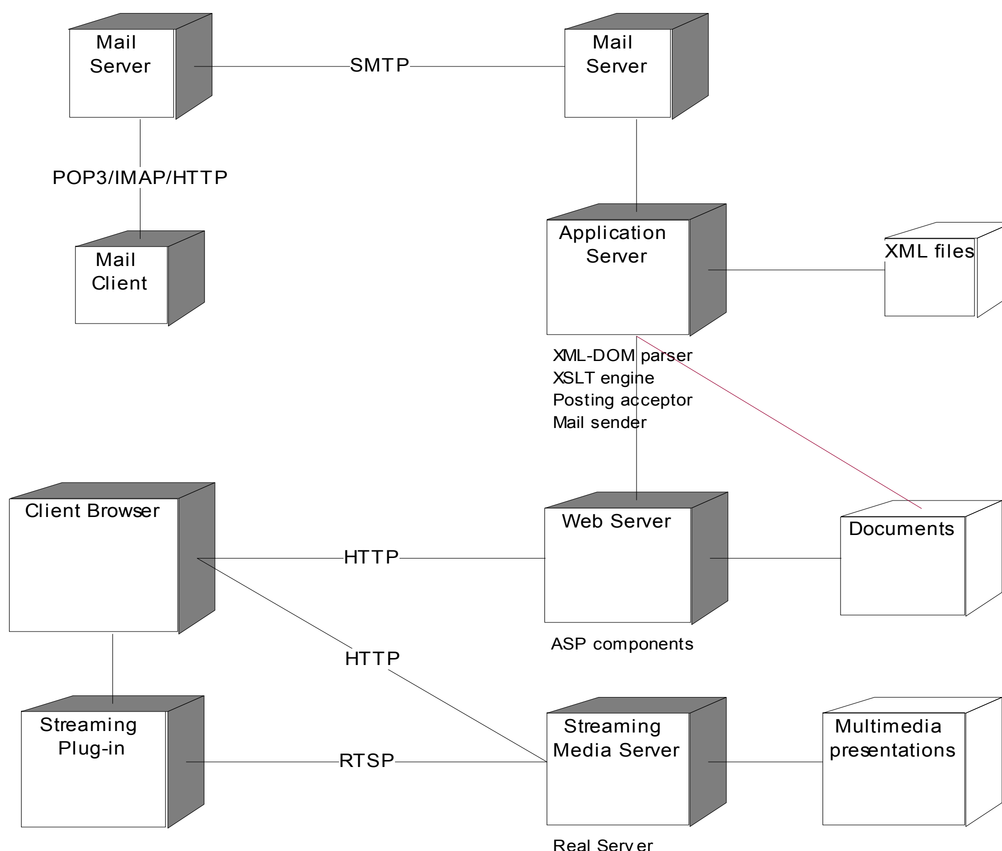

FIG. 3. IBIS DEPLOYMENT DIAGRAM

In the meantime, we have embraced the idea that Internet ubiquity offers the opportunity of breaking the size limit of inspection teams but this can be achieved only by reengineering the inspection process [9].

We have adopted an alternative design of the inspection process, which is based on behavioral theory of group performance and empirical findings of research on inspection variations. Furthermore, lessons learnt from OSS projects have made us confident that defect discovery can really scale up if we remove the constraint that all issues have to be discussed with the mandatory participation of the entire group of inspectors. As a consequence, we have redesigned the IBIS according to the reengineered inspection process.

The conformance to a redesigned inspection process and the lightweight architectural approach makes the IBIS different from other web-mediated inspection systems, including both prototypes such as WiT [6], developed at University of Oulu, and commercial applications such as ReviewPro [17] developed on the basis of Radnet's WebShare Server, a groupware platform.

IV. CONCLUSIONS

The development of large e-Business applications requires highly-skilled engineers who are not easily available in a same location. The labor-shortage problem for e-Business projects can be solved by geographical distribution of development teams. With its globally distributed developer population and frequent cycle time, OSS projects represents the extreme in "virtual software projects", thus dealing with the issue of communication and coordination among their many and widespread contributors.

In this paper, we have discussed the current state of OSS project infrastructure. Although each OSS project developed some unique practices for communication and coordination, there are also significant commonalities, including both process and product technologies: widespread asynchronous communication tools (email readers and web browsers), mailing lists and web portals to support the community of developers and deploy services (change and version control, bug tracking, task assignments) which do not require long connections, process guidelines working as social contracts

among participants (coding styles, access to source base, decision-making).

Even if peer reviewing is part of OSS processes (although limited to source code), its informal practice does not provide product and process measures, that can be tracked over time for improvement purposes. In OSS projects, code review is conducted though mailing lists, and then defect data are buried among hundreds of other messages in the mailing list archives. Rather, software inspections (a rigorous form of peer review) have been adopted from many years by large industrial organizations because of their impact on product quality and cost of non-quality. However, software inspections have been limited in their adoption by a lack of automated support, the prevalence of synchronous communication between participants, and the overload of coordination on the moderator's shoulders.

We have described an ongoing project that aims to provide a system, called the IBIS, to support scalable software inspections over the Internet. Based on lessons learned from OSS projects' infrastructure, the IBIS adopts a reengineered inspection process to minimize coordination problems and a lightweight architecture to achieve the maximum of simplicity of use and deployment.

At this stage of development, the IBIS prototype implements much functionality of the reengineered inspection process. We are also extending the system to include variations with respect to the products and related defect classifications. Finally, we intend to include support for various reading techniques such as checklists and scenarios [3]. As the system is incrementally delivered, we intend to empirically evaluate it within academic software engineering courses and pilot projects of supporting companies.

References

[1] Apache HTTP Server Project, Developer Resources. http://dev.apache.org/

[2] V. Basili, S. Green, O. Laitenberger, F. Lanubile, F. Shull, S. Sorumgard, and M. Zelkowitz, "The Empirical Investigation of Perspective-based Reading", Empirical Software Engineering, 1, 133–164, 1996.

[3] V. R. Basili, "Evolving and packaging reading technologies", Journal of Systems and Software, 38 (1): 3-12, July 1997.

[4] M. E. Fagan, "Design and Code Inspections to Reduce Errors in Program Development", IBM Systems Journal, 15(3):182–211, 1976.

[5] R. T. Fielding and G. Kaiser, "The Apache HTTP server project", IEEE Internet Computing, 1(4): 88--90, July/ Aug. 1997.

[6] L. Harjumaa, I. Tervonen, "Virtual Software Inspections over the Internet", Proc. of the 3rd ICSE Workshop on Software Engineering over the Internet, January 2000.

[7] J. D. Herbsleb, A. Mockus, T. A. Finholt, and R. E. Grinter, "An Empirical Study of Global Software Development: Distance and Speed", Proc. of Int. Conf. on Software Engineering, Toronto, Canada, 2001.

[8] J. D. Herbsleb, and D. Moitra, "Global Software Development", IEEE Software, 18(2): 16-20, 2001.

[9] P.M. Johnson, "Reengineering Inspection", Comm. of the ACM, 41(2): 49-52, 1998.

[10] O. Laitenberger, J.M. DeBaud, "An encompassing life cycle centric survey of software inspection", The Journal of Systems and Software, 50: 5-31, 2000.

[11] A. Mockus, D. W. Weiss, "Globalization by Chunking: A quantitative approach", IEEE Software, 18(2): 30-37, 2001.

[12] NASA Goddard Space Flight Center, Software Formal Inspections Guidebook, NASA-GB-A302, 1993, http://satc.gsfc.nasa.gov/fi/fipage.html

[13] O. S. Initiative. The Open Source definition, 1997. http://www.opensource.org/osd.html

[14] A. Porter, H. Siy, A. Mockus, and L. Votta, "Understanding the Sources of Variation in Software Inspections", ACM Transactions on Software Engineering and Methodology, 7(1): 41-79, January 1998.

[15] E. Raymond, "The Cathedral and the Bazaar", http://www.earthspace.net/~esr/writings/cathedral-bazaar/

[16] C. Sauer, D. R Jeffery, L. Land, and P. Yetton, "The Effectiveness of Software Development Technical Reviews: A Behaviorally Motivated Program of Research", IEEE Transactions on Software Engineering, 26(1):1–14, January 2000.

[17] Software Development Technology (SDT), ReviewPro, http://www.sdtcorp.com/reviewpr.htm