

Software Production Infrastructure to Support Agile Methodologies

Alberto Sillitti
Free University of Bozen
Bolzano, Italy
asillitti@unibz.it

Giancarlo Succi
Free University of Bozen
Bolzano, Italy
gsucci@unibz.it

Abstract – In the last few years, software production has dramatically changed: the Internet, globalization, alternative development practices like open source have been important drivers and there are also pressures from ever more educated and informed customers for ever more stringent needs and ever more rapid delivery of solutions, etc. Recently, a set of new software development processes has emerged, addressing typical problems of traditional development techniques, such as keeping the project within the planned budget and deadlines, increasing the quality, satisfying customers, coping with dynamic business needs, etc. These methodologies are called “Agile Methodologies” (AMs) as they are adaptive and not predictive, people oriented and not process oriented, and quality focused (Beck *et al.*, 2001). Organizations that have adopted them report they were able to deliver quality software which is easy to maintain (Beck, 1999) in shorter time. All these methodologies use an object-oriented approach, which is well suited to support an iterative incremental process.

Keywords: integrated environments, agile methodologies

I. INTRODUCTION

All AMs are based on similar principles, which Alistair Cockburn defines as “Minimal” (1998). They suggest small, agile teams based on direct interpersonal communications, using state of the art technology with the lowest possible overhead from tasks that are not directly productive.

There is increasing evidence that Agile Methodologies (AMs) have the potential to improve significantly the way software is produced (Marchesi and Succi, 2002).

The Research Roadmap on the Agile Methodologies developed by the NAME project (Network for Agile Methodologies Experience) (NAME, 2003) identified a set of research needs in the area of AMs. Topics with the highest priority are (a) specific tool support for agile methodologies – non-invasive project management, unit and functional testing, requirements negotiation, experience exchange; and (b) tool integration to reduce the inefficiency caused by the use of different formalisms and operative procedures – integration of existing tools with new agile ones, integration of customized tools, multiple views of the development process according to the role of the stakeholders, e.g. developer, customer, manager, etc.).

A set of stand-alone tools is hardly enough. An integrated environment is really what should be sought, still maintaining a very flexible and agile infrastructure: a lightweight, multi-stakeholder system appears to be the most suitable solution. In a multi-stakeholder system, every key stakeholder (developer, manager, customer, etc.) can have a consistent view of the system being developed in her or his own formalism of choice – from customer requirements down to code. Nearly always, individual tools use different repositories to store information making

synchronization a complex, time expensive, error-prone task, hardly ever accomplished properly. Different tools require different competencies from the people using them, thus making the integration of the work done using different tools even harder.

The paper is organized as follows: section 2 describes the integrated architecture; section 3 presents a possible solution; finally, section 4 draws the conclusions.

II. AN INTEGRATED ARCHITECTURE

The integration framework of the multi-stakeholder system needs to be open source. This is by far the most effective way to achieve a successful system to support Agile Methodologies, promote interoperability and make the system extendable. An open integration framework provides incentives for small, medium, and large tool developers and even universities and research centers to have their own tools compatible and interoperable with the framework, as the experience of Emacs evidences (Raymond, 2003). The Agile Software Engineering community is particularly sensitive to the issue of open source tools, as is evident by the tools agile people prefer – JUnit, Cruise Control, Eclipse just to cite a few, as has been presented in the NAME research roadmap (NAME, 2003) and in the discussion at the last XP200x conferences (Marchesi and Succi, 2003).

The choice of open source software is hardly a philosophical choice; it is a necessity due to the basic principles of the AMs. In addition, open source tools are more adaptable to specific environments, even if, sometimes, corporate tools are better engineered; this double edge sword forces the inclusion in the proposal of both open source and proprietary tools. Integration is a key step in reduction of waste; tool integration can only be effective if the integration protocol is fully open.

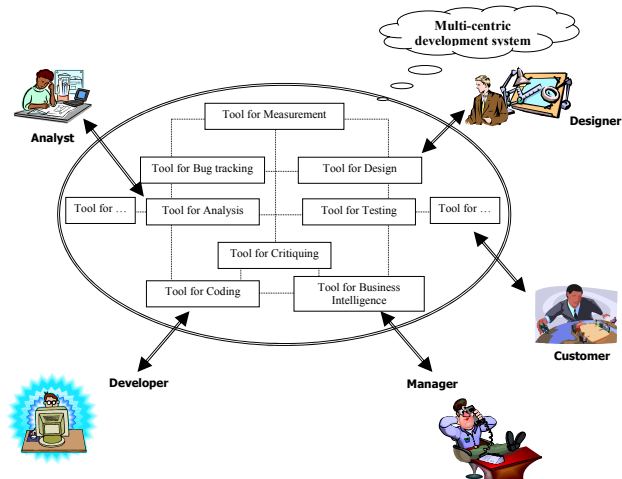


Fig. 1. Overview of the multi-stakeholder development system

The proposed architecture includes the development of a lightweight integration protocol, interface specifications, an integration engine, and a set of basic and customizable views focused on specific classes of users.

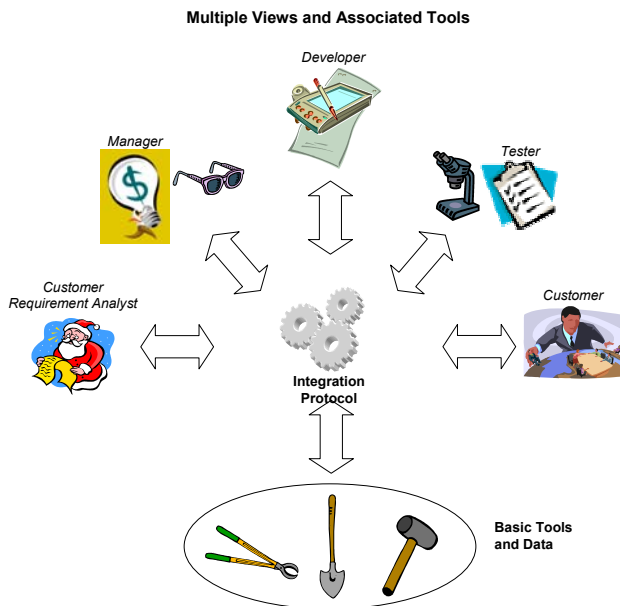


Fig. 2. Architecture of the solution

The architecture includes two levels of complexity providing two levels of integration:

1. Passive integration
2. Active integration

The former level, passive integration, mainly deals with a common set of guidelines to store information in files and to manage such shared data repository.

In this architecture, information exchanges include only data, no notification or control commands are exchanged.

The main components of the architecture are:

1. *Data Repository*: it collects data using a specific data format based on the XML technology
2. *Client Applications*: they are applications sharing the same roles to collect and store data into the data repository. There are three kinds of clients:
 - a. *Integration-aware Applications*: these applications know how the integration system works and are developed according to these set of standards
 - b. *Customizable Applications*: these applications do not provide support for the integration architecture but they are extensible through plug-ins that can add such functionalities
 - c. *Non-customizable Applications*: these applications are neither integration-aware nor extensible through plug-ins. To allow an interaction with the integration architecture a wrapper to convert input and output data is required.

The latter level, active integration, includes the previous one but requires a more complex architecture and a local service to perform data integration. Moreover, the integration is not limited to a common data repository; a dataflow management system is required to perform complex integration and message exchanges among all applications.

The proposed system will contribute to extend the state of the art of the software production infrastructure, especially focusing on tools for agile software development.

At present, there are only a few tools focusing on agile software development and most of them lack key functionalities. Moreover, they do not sufficiently support the whole development process. The system will contribute in this area through an extensive support to agile software development.

Tool integration and the ability to provide customized views on the same set of engineering data is a key feature to allow customers, managers, and developers to reduce misunderstandings. This feature is one of the main drawbacks of modern development tools. The system will address this issue by developing an open and lightweight integration framework and a basic infrastructure implementing the framework.

III. THE IMPLEMENTATION

According to the architecture presented in section 2, the implementation of the system has two levels of complexity.

The first step (Fig. 3) deals with the passive integration of information flows generated inside applications. These information are mainly file formats and configuration preferences that should be shared among different applications.

At this level, all the three main kinds of applications can interact because they all use files or databases to store application-specific information. In the hardest case (non-customizable applications) a simple data wrapper is required in order to create a bidirectional communication with the integrated system.

The specification of the data repository is rather than simple. It can include a huge amount of different information making the system too complex and hard to understand and use.

A feasible approach, in this case, is the identification of a small set of information required to satisfy common requests. These data include: source code (including tests), requirements specifications (e.g., user stories), etc.

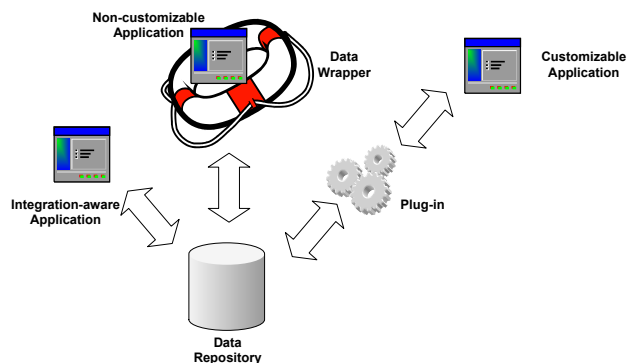


Fig. 3. Passive integration architecture implementation

This first level of integration is able to address only the problem of the data exchange, but application interaction is more than that.

An higher level of integration includes the ability of synchronous and asynchronous communication among

applications, in order to react when some events are generated (Fig. 4). At this level, applications share data formats and events in order to achieve an higher integration. Applications register themselves in order to receive events they are interested in and an integration server is able to route messages to the specific application.

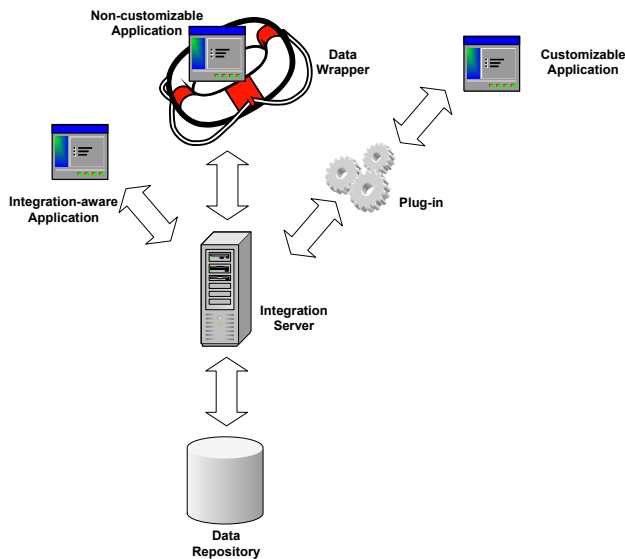


Fig. 4. Active integration architecture implementation

Message exchange can be very simple notifying applications that some data have changed, or very complex if data are exchanged directly. This second approach is more complex and involve on-the-fly data conversion among applications and a certain degree of knowledge of the applications connected to the system.

IV. CONCLUSIONS

This paper presented an architecture for application integration focusing on the development of an integrated development platform able to support agile methodologies through the customization of the basic components.

Many application integration frameworks have failed due to their broad objective: integrating all possible applications. On the contrary, the aim of this framework is the integration only of a specific subset: development tools. This focus reduces the complexity of the architecture required and the code required to support the framework that developers have to include in their products.

V. REFERENCES

(Beck *et al.*, 2001) Beck, K., M. Beedle, A. Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries, J. Kern, B. Marick, R. Martin, S. Mellor, K. Schwaber, J. Sutherland, D. Thomas (2001) Manifesto for Agile Software Development, available at <http://agilemanifesto.org/>

(Cockburn, 1998) Cockburn, A. (1998) Surviving Object-Oriented Projects, Addison-Wesley

(Marchesi and Succi, 2002) Marchesi, M., G. Succi (eds.) (2002) Proceeding of the 3rd International Conference on eXtreme Programming and Flexible Processes in Software Engineering (XP2002)

(Marchesi and Succi, 2003) Marchesi, M., G. Succi (eds.)

(2003) Proceeding of the 4th International Conference on eXtreme Programming and Flexible Processes in Software Engineering (XP2003)

(NAME, 2003) NAME Consortium Research Roadmap (2003), available at <http://name.case.unibz.it>

(Raymond, 2003) Raymond, E.S. (2003) The Art of Unix Programming, available at <http://www.catb.org/~esr/writings/>