# Integrating XP project management in development environments

M. Angioni[a], D. Carboni[a], S. Pinna[b], R. Sanna[a], N. Serra[b] and A. Soro[a]

[a]CRS4 - Center for Advanced Studies, Research and Development in Sardinia, Parco Scientifico e Tecnologico POLARIS, 09010 PULA (CA), Italy, {angioni, dcarboni, raffa, asoro}@crs4.it

[b]Dipartimento di Ingegneria Elettrica ed Elettronica, Università di Cagliari, Piazza D'Armi, 09123 Cagliari, Italy, {pinnasandro, nicola.serra}@diee.unica.it

Extreme Programming (XP) is an Agile Methodology (AM) which doesn't require any specific supporting tool for being successfully applied. Despite this starting observation, there are many reasons leading a XP team to adopt Web based tools to support XP practices. For example, such tools could be useful for process and product data collection and analysis or for supporting distributed development. In this article we describe XPSuite, a tool composed of two parts: XPSwiki, a tool for managing XP projects and XP4IDE, a plug-in for integrating XPSwiki with an Integrated Development Environment (IDE). Moreover, we will show how the full Object Oriented implementation provides a powerful support for extracting all data represented in the model that the system implements.

## 1. Introduction

Extreme Programming is a software development methodology which does not rely on any particular tool, but rather is based on the common understanding of fundamental values and on a disciplined application of best practices. In particular, projects developed with XP show that good results can be obtained using sheets of papers to collect user requirements, wall boards to show diagrams and other project-relevant information, and shared workspaces to maximize face-to-face communication. Nevertheless, there are a number of reasons that may lead XP teams to adopt software and Internet based tools to facilitate application of XP practices:

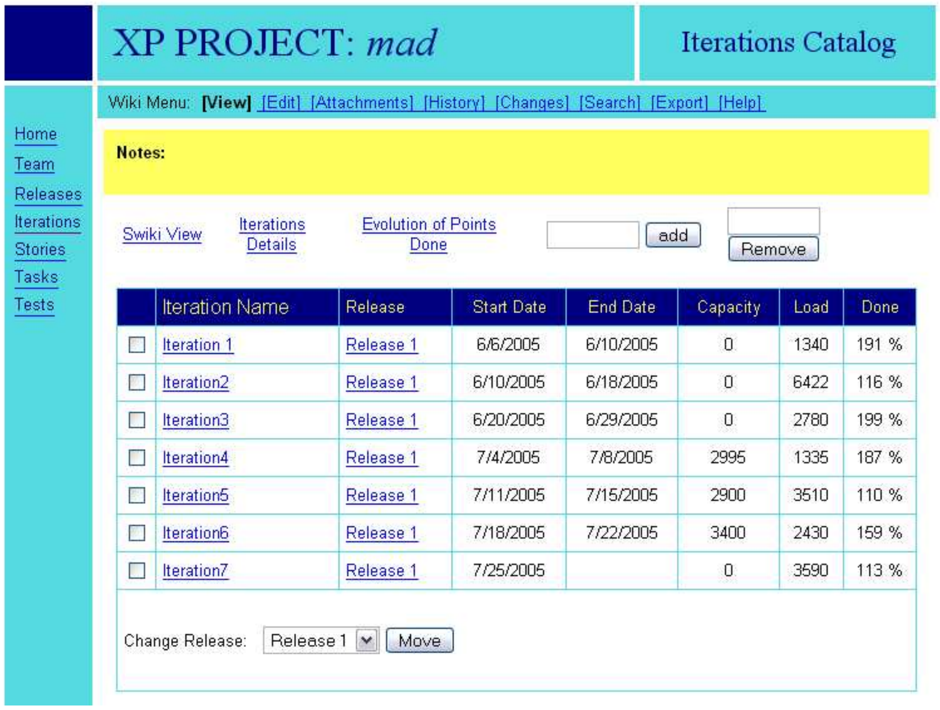
- process data can be automatically collected and analyzed afterwards. The process can be validated and team activities can be more effectively measured; the adoption level of agile practices can be quantified;
- managers and customers have the feeling that tools are something more "concrete" than pure methodologies;
- dispersed teams can work together with appropriate Internet connections. Team members can have a common and updated overview of the project status.

In this article we will describe XPSuite, a two-parts tool incorporating XPSwiki, a tool for managing XP projects, and XP4IDE, a plug-in for integrating XPSwiki into IDEs. The system has a full Object Oriented implementation so it is possible to extract all data represented in the developed model.

In section §2 we briefly introduce the XP methodology and process. Section §3 contains a description of possible metrics for a XP process. In section §4 we present XPSuite, the suite of Internet tools we have developed for eliciting requirements, collection of user stories, process tracking and process data collection, while in section §5, we describe the underlying conceptual model. Finally, section §6 describes the functionalities of this suite of tools while section §7 briefly discusses privacy issues.

## 2. An Agile Methodology: XP

Agile Methodologies (AM) are interesting new methodologies for software development proposed at the end of the 90's. They are particularly suited when it's difficult to understand the sys-



| | Iteration Name | Release | Start Date | End Date | Capacity | Load | Done |
|---|---|---|---|---|---|---|---|
| ☐ | Iteration 1 | Release 1 | 6/6/2005 | 6/10/2005 | 0 | 1340 | 191 % |
| ☐ | Iteration2 | Release 1 | 6/10/2005 | 6/18/2005 | 0 | 6422 | 116 % |
| ☐ | Iteration3 | Release 1 | 6/20/2005 | 6/29/2005 | 0 | 2780 | 199 % |
| ☐ | Iteration4 | Release 1 | 7/4/2005 | 7/8/2005 | 2995 | 1335 | 187 % |
| ☐ | Iteration5 | Release 1 | 7/11/2005 | 7/15/2005 | 2900 | 3510 | 110 % |
| ☐ | Iteration6 | Release 1 | 7/18/2005 | 7/22/2005 | 3400 | 2430 | 159 % |
| ☐ | Iteration7 | Release 1 | 7/25/2005 | | 0 | 3590 | 113 % |

Figure 1. The GUI of XPSwiki.

tem functionalities during the early phase of the process, due to continuous requirements changing, mutable environmental factors or mutable market conditions. So, agile methodologies are goal-oriented: they allow to adapt the process to all these changes, reaching a goal at a time, with frequent release cycles. They are in opposition to "heavy" methodologies like the well-known Waterfall.

The most famous and common AM is eXtreme Programming (XP). Simplicity, communication, feedback and courage are the founding values of XP and all the XP best practices are coherent with these values (for a complete description of XP see [2]). XP is "extreme" because it carries to an extreme degree some of the best practices already used in traditional software development. For instance, if unit testing produces high quality code, then it must be applied to its extremes, writing unit tests before the code itself. Similarly, if refactoring produces a better code, then "refactor mercilessly". Also communication is "extreme": to share a common vision of project's requirements and project's status, to avoid any misunderstanding and to develop exactly what is required, the customer (or an effective representative) is "embedded" as a member of the team. In addition, daily meetings are held to check the progress of the team, to identify and manage problems and to propose corrective actions.

The first step of a XP project is the planning phase called *Planning Game*, where developers and customers elicit requirements and define the functionalities to be developed during the project. Customers and developers identify and isolate specific functionalities and describe them in pieces of papers called *User Stories*, with a business value and a priority. Effort required to implement a story is estimated by developers, who divide stories into micro-activities, called *Tasks*, to be performed in 2-3 ideal working days ("ideal" stands for 8 hours of effective work). Developers choose voluntarily which task to develop and take responsibility for their task being completed on time.

Each story generally requires an effort of 1-2 weeks. A story is considered complete only if all its Acceptance Tests pass. Acceptance Tests are functional tests that guarantee the correctness of the functionalities specified with *User Stories*. The customer is responsible for specifying and verifying the validity of the tests, that should be automated in order to be run often. Stories are grouped into *Iterations*, each one with a duration of 2-3 weeks. After a cycle of 2-3 iterations, the product is released to the customer.

The customer is aware that the first release will provide only some functionalities and that many releases could be necessary before all functionalities are implemented. These periods of time, indicated by [2], can vary depending on the specific project, but the idea is always the same: release cycles must be very frequent.

In XP, one of the team members plays the role of *Tracker*: tracking is the critical activity of measuring the team progress. The essence of tracking is the face-to-face contact with the team, to check and track what is done, what is in progress, what is in stand-by and why. In fact, the tracker asks developers questions like:

- which is the status of your task?
- how many (ideal) days have you worked on this?
- how many more (ideal) days do you need before you've done?
- what obstructs you from completing the task?

This information is critical to assess whether or not the effort estimated during the *Planning Game* is realistic and reflected.

## 3. Measuring the XP Process

Measurement is an inherent and fundamental activity of all engineering disciplines. Measurement provides the mechanism to create the aid in answering a variety of questions associated with the enactment of any software process. It helps to support project planning, to determine the strengths and weaknesses of the current processes and products, to evaluate the quality of specific processes and products. Measurement also helps, during the course of a project, to assess its progress, to take corrective action based on this assessment, and to evaluate the impact of such action.

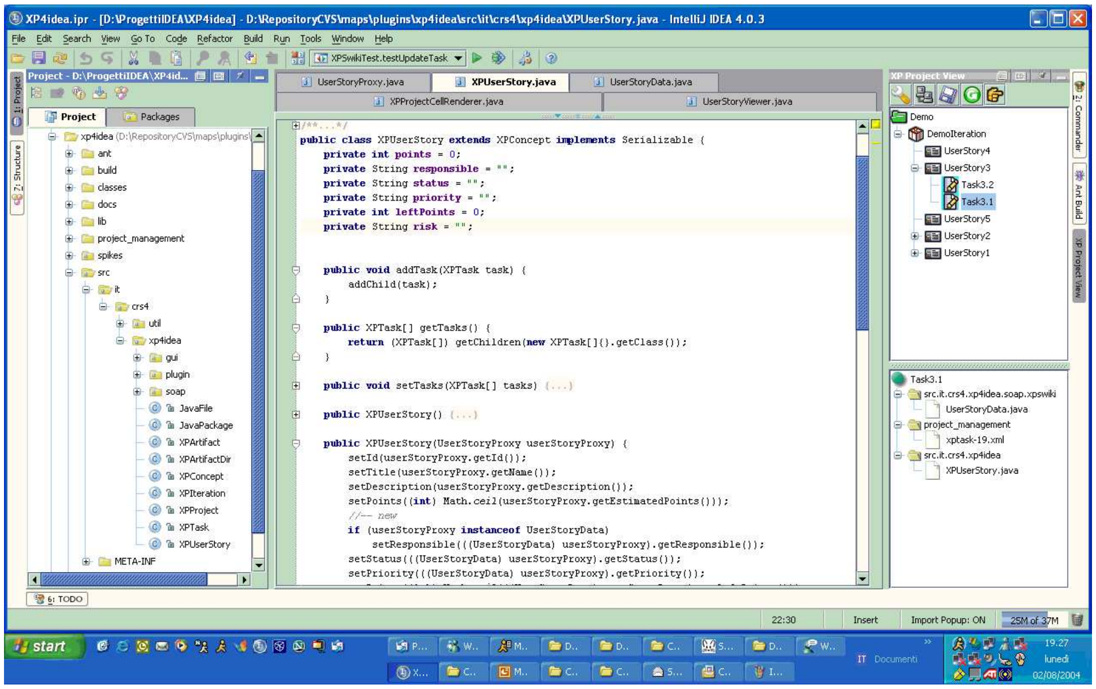
What exactly is being measured can change depending on the point of view. For instance, industry managers are mainly interested in maximizing the value of products minimizing costs and time to market, while engineers may be more interested in measuring product quality to decide what has to be changed in the process to improve it. While Agile Methodologies and Extreme Programming are not metric centric processes, measurement still plays a fundamental role as a steering tool for controlling the development process. One of the principles in the Agile Manifesto [1] states:

> *At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.*

Kent Beck, explaining the eXtreme Programming methodology [2], points out that the measurement represents the basic management tool to get control on the project evolution. For example, he suggests the ratio between estimated and actual development time as a basic measure for playing the planning phase and to set the project velocity. Such metrics once collected can be presented in the form of tables and diagrams as a visual representation of project evolution. For example, figures 3 and 4 show mean and standard deviation of the task estimation error for each iteration.

Measurement and metrics can also be used to quantify the process agility. For leading this purpose, Ron Jeffries argues that a very powerful metric is the Running Tested Features in time (RTF, see [4]).

The suite of tools we have developed has a pri-



```
/**...*/
public class XPUserStory extends XPConcept implements Serializable {
    private int points = 0;
    private String responsible = "";
    private String status = "";
    private String priority = "";
    private int leftPoints = 0;
    private String risk = "";


    public void addTask(XPTask task) {
        addChild(task);
    }

    public XPTask[] getTasks() {
        return (XPTask[]) getChildren(new XPTask[]{}.getClass());
    }

    public void setTasks(XPTask[] tasks) {...}

    public XPUserStory() {...}

    public XPUserStory(UserStoryProxy userStoryProxy) {
        setId(userStoryProxy.getId());
        setTitle(userStoryProxy.getName());
        setDescription(userStoryProxy.getDescription());
        setPoints((int) Math.ceil(userStoryProxy.getEstimatedPoints()));
        //-- new
        if (userStoryProxy instanceof UserStoryData)
            setResponsible(((UserStoryData) userStoryProxy).getResponsible());
        setStatus(((UserStoryData) userStoryProxy).getStatus());
        setPriority(((UserStoryData) userStoryProxy).getPriority());
```

Figure 2. The GUI of XP4IDE embedded in IntelliJ IDEA.

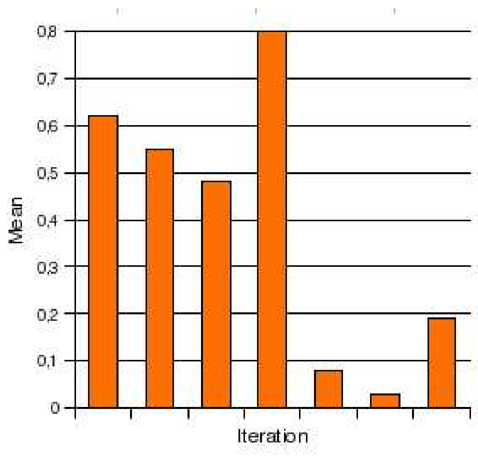


Figure 3. Mean Estimation Error for each iteration.

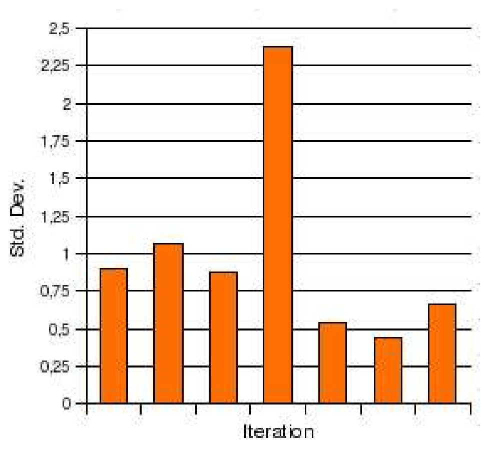


Figure 4. Standard Deviation Estimation Error for each iteration.

mary role as it is able to gather process data that could prove useful for calculating the desired process metrics.

## 4. XPSuite: Tools for the XP Process

In this section we provide an overview of two Internet based tools we have developed to support planning and tracking in a XP project: XPSwiki and XP4IDE.

### 4.1. XPSwiki

XPSwiki [5,6,8] (figure 1) was developed to meet real needs for software businesses to collect requirements and schedules in an electronic format. Managers, customers and developers can access XPSwiki by means of a Web browser. XPSwiki is built on top of a Wiki [3] engine written in Smalltalk called Swiki [7].

Wiki is a very simple and effective content management system. Users can browse Wiki pages and, if they have the appropriate privileges, can also edit the pages directly from the Web, using a small set of simple formatting rules.

Swiki also enables to add structure to pages, defining input forms for them. In this way, in a Swiki one can have pages constrained by a given structure, that are used to hold structured information, as well as free-format pages, freely added to existing pages like Post-it notes.

The direct advantages of a Web based tool for the management of XP process data can be summed up as follows:

- the *Tracker* and the developers can enter effort data directly from the Web, without handling paper documents, minimizing risk and errors;
- dispersed teams can cooperate and view the overall project status;
- everyone can have a common and updated vision of the project tracking;
- *User Stories*, *Tasks* assignments and any relevant effort data are immediately available on the Web, and accessible from programmers pairs workstations.

Although a Web based tool is effective during the *Planning Game*, it does lack one major feature: it is not integrated into the user's development environment. So, the synchronization between developers progresses and tracking data collected in the XPSwiki is not automatic, but left to the common sense and the firmness of the developers. Team members must edit and submit information on the XPSwiki, and this is possible only via Web.

In the next section we will describe XP4IDE, which integrates some of the XPSwiki functionalities directly into the IDE, providing some additional features:

- effort can be automatically recorded by the IDE;
- developers can view the XP process integrated with their development tools;
- the choice of what task to deal with is in real-time automatically communicated to the project management tool;
- time spent tracking the project sensitively decreases;
- developers stop to hate the *Tracker*!

### 4.2. XP4IDE

XP4IDE is an Internet based tool which connects to a XPSwiki server and provides a view of the XP process data embedded in the GUI of the IDE (figure 2).

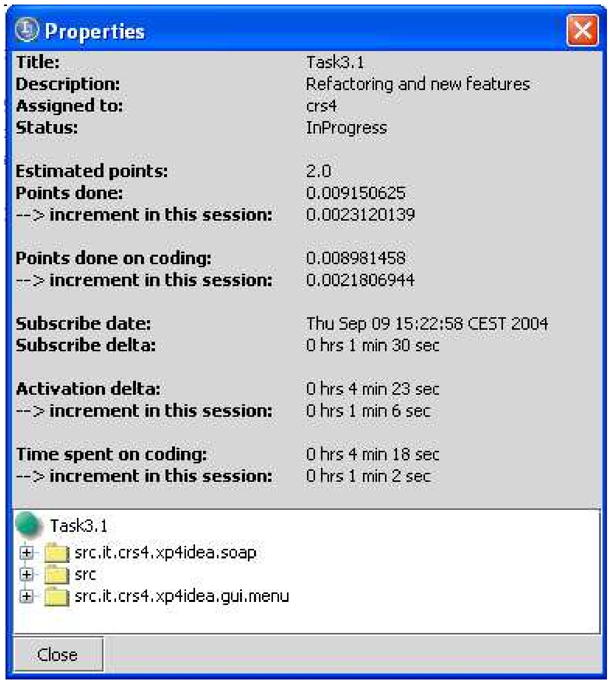
The role of XP4IDE is twofold: on the one hand it enables the developer to view, directly integrated into the IDE, process data, *User Stories*, *Tasks*, Acceptance Tests and *Artifacts* (classes, documentation files, Make files or Ant files, test cases, etc.), all managed and saved on the XPSwiki server. On the other hand, it measures the specific time spent on *User Stories*, *Tasks* and *Artifacts* and sends this information to the XPSwiki server, automating the tracking activity. The overall system architecture is quite simple: XPSwiki acts as backend in a client-server architecture where the application protocol is built on top of SOAP/TCP/IP. The integration between XPSwiki and XP4IDE provides the following advantages:

- each actor in the XP process accesses to process data with the right user interface for the right phase. Developers can use the IDE during development, while customers and managers can use XPSwiki by means of a simple Web browser. Both of them can use the Web interface during the *Planning Game*;

- it is possible to associate the *Artifacts* to the *Tasks* in which they are created and developed. This way, the tool collects data about the time spent on every *Artifact*, every *Task* and every *User Story*;

- measuring effectively the effort spent and presenting this information to developers (figure 5), is one way of building a knowledge base useful for enhancing their ability to estimate stories in future planning activities (see section §3);

- beginners can immediately have a view of the XP process. Concepts, roles and process data are presented in a structured way, minimizing the learning curve.



| | |
|---|---|
| **Title:** | Task3.1 |
| **Description:** | Refactoring and new features |
| **Assigned to:** | crs4 |
| **Status:** | InProgress |
| **Estimated points:** | 2.0 |
| **Points done:** | 0.009150625 |
| **--> increment in this session:** | 0.0023120139 |
| **Points done on coding:** | 0.008981458 |
| **--> increment in this session:** | 0.0021806944 |
| **Subscribe date:** | Thu Sep 09 15:22:58 CEST 2004 |
| **Subscribe delta:** | 0 hrs 1 min 30 sec |
| **Activation delta:** | 0 hrs 4 min 23 sec |
| **--> increment in this session:** | 0 hrs 1 min 6 sec |
| **Time spent on coding:** | 0 hrs 4 min 18 sec |
| **--> increment in this session:** | 0 hrs 1 min 2 sec |

```
Task3.1
⊞ src.it.crs4.xp4idea.soap
⊞ src
⊞ src.it.crs4.xp4idea.gui.menu
```

Figure 5. Effort data and properties of a Task in the XP4IDE GUI.

## 5. Process Data

This section describes the conceptual model underlying the XPSwiki and XP4IDE suite, showing the entities and their relations by means of UML class diagrams. This Object Oriented structure can be navigated to extract process data for the computation of useful metrics.

As previously mentioned in section §4, XPSwiki extends a Wiki structure, inheriting in this way all its attractive features like the versioning capability. This feature enables XPSwiki to keep track of all the versions of each entity (*Task*, *User Story*, *Iteration*,...).

Now we give a description of the UML model of the XPSwiki shown in figure 6. Since this model represents the basis for managing a XP process as described in section §2, only a brief description of the model is shown here.

A XP *Project* is performed by a *Team*. A *Team* is composed of one or more *TeamMembers*. The *Project* is then divided into *Releases* and *Iterations* of fixed length through the timeboxed approach of XP. A number of instances of *UserStory* must be implemented within an *Iteration*. Each *UserStory* is divided into one or more instances of *Task* and has one or more *Acceptance Tests*. The successful completion of all its *Acceptance Tests* proves the successful implementation of a *UserStory*.

To fully integrate the model with the development data collected by XP4IDE, we have extended the XPSwiki model to include other entities, such as *Artifacts*. As shown in figure 7, *Tasks* and *Artifacts* are associated with a many-to-many relation (an *Artifact* could be created in a *Task* and modified in another one). An *Artifact* may be of different types such as *SourceCode* or a generic *Document*. A *Task* has its own *TrackTable* which stores information concerning the effort expended by developers for each spe-

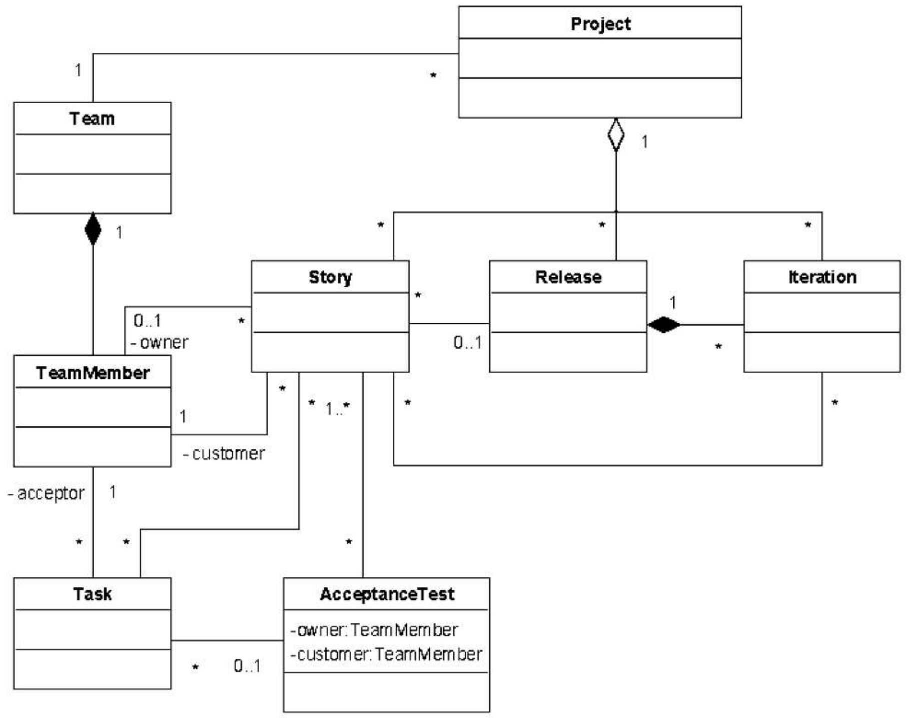


Figure 6. UML model of the XP process as supported by XPSwiki.

cific working *Session*. In more detail, each *Session* contains the following fields:

- **subscriber**: the *TeamMember* who subscribed to the *Session* on that *Task*;
- **pairProgrammer**: the *TeamMember* who was programming with the *subscriber* in that *Session*;
- **startTime**: date and time *subscriber* started working on that *Session*;
- **subscribedDelta**: duration of the *Session*.

Moreover, each *Session* has its own *ArtifactsTrackTable* which keeps track of the effort spent on each *Artifact* modified or created during that *Session* (*ArtifactSession*). More in depth, each *ArtifactSession* stores the following data:

- **artifact**: *Artifact* involved in the *Session*;
- **startTime**: date and time *subscriber* started working on that *Artifact* in that *Session*;
- **activationDelta**: the period of time the *Artifact* window was active during that *Session*;
- **codingDelta**: the time spent on coding (typing) that *Artifact* during that *Session*.

## 6. Working with the XPSuite

In this section we describe the functionalities of the XPSuite from the developer point of view during a development session. To better harmonize with XP, the functionalities are described in the form of *User Stories*.

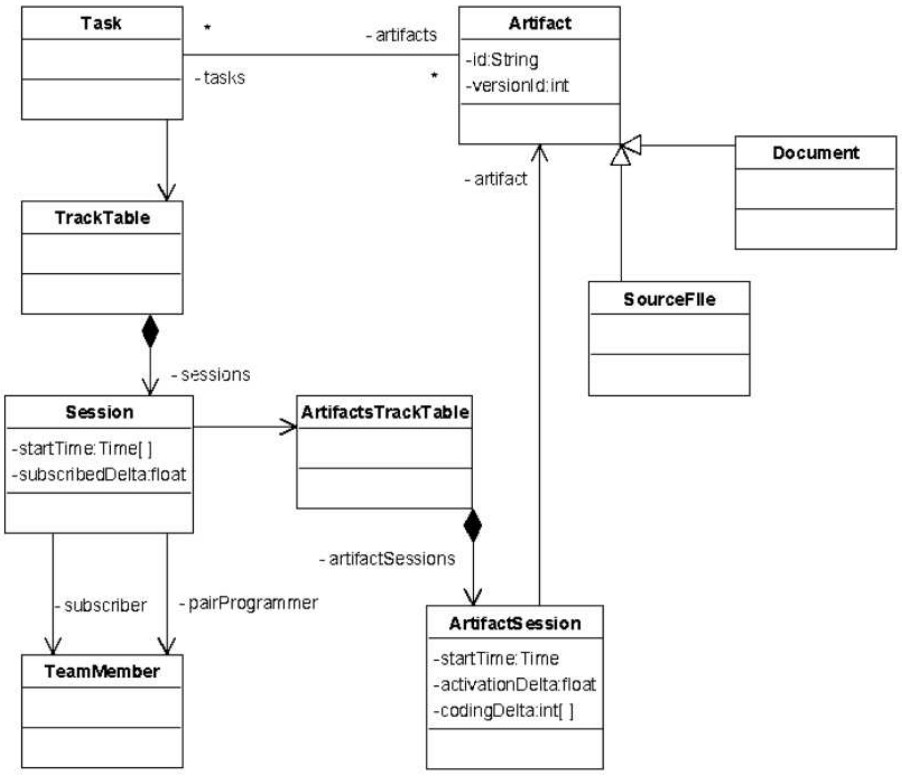


Figure 7. Extension of the XPSwiki model in order to support data collected by XP4IDE.

## 6.1. Story: Login and Connection to a Project

To start a working session, the developer opens the IDE, selects the docked GUI called XPView, the view of the XP project, which prompts for login and password to connect to the XPSwiki server. Then the developer selects the pair programmer with whom he is going to work with and the project among those available on the XPSwiki. Project data are loaded in the IDE and presented in the XPView in the form of a tree. The tree is composed of *Releases, Iterations, User Stories, Tasks* and *Artifacts*. *Tasks* already assigned to the developer are highlighted and sorted by priority. Every tree element has a property window which presents the attributes for that element (for instance, a story has a title, a description, a priority, a risk and an estimated effort).

## 6.2. Story: Subscribe to a Task

Once the developer has logged in, he can select a *Task* and subscribe to it. The status of the *Task* changes from “inactive” to “in progress”. Developers subscribed to a *Task* can create and associate new *Artifacts* to that *Task* or modify existing ones even if associated with other *Tasks*.

## 6.3. Story: Automatic Measure of Effort Spent

During the working session, the XP4IDE measures:

- the *Artifact* activation delta (the period of time the *Artifact* window was active);
- the time spent on coding an *Artifact*;

- the *Task* activation delta (computed as the sum of the activation delta of all *Artifacts* developed in the *Task*);
- the time spent on coding within a *Task* (computed as the sum of the time spent on coding all *Artifacts* developed in the *Task*);
- the subscribe delta (total amount of time the developer is subscribed to a *Task*).

This measurement should be sent to the XPSwiki, choosing one or more of the following options:

- periodically;
- when the developer unsubscribes to a *Task*;
- when the *Task* is completed;
- when the IDE is closed.

### 6.4. Story: Open a Thread on a Story

A developer can post from the IDE user interface an issue or a question opening a new thread. The other members of the team can follow up the discussion and provide solutions or suggestions.

### 6.5. Story: Managing a Task

A developer should be able to see the status of a *Task*, to launch its tests, to work on its *Artifacts*, to close the *Task* or to suspend the *Task* defining a prerequisite (for instance Task1 depends on Task2).

## 7. Privacy Issues

Concern may arise over privacy issues, since software tools such as XPSuite could clearly be used to spy the behavior of developers, instead of supporting them in their day to day activities, as they are supposed to do. Such objections, perfectly justified, should be seriously taken into account when developing and deploying systems like XPSuite. Even in those cases where it is possible to oblige developers to use automatic tracking tools like XPSuite, this should be seriously evaluated. In fact, Extreme Programming is claimed to be a people centered methodology, hinged on trust and reciprocal esteem between team members, and on making full use of human resources, instead of imposing control and authoritarian management.

In our opinion, XPSuite (and other similar tools) should be adopted, upon mutual agreement with team members, if necessary in a smooth, gradual way. On the other hand, the use of XPSuite has proven valuable in the experimentations conducted so far, both for the advantages that it offers when coordinating dispersed teams and for its ability to gather process data contextualized against the specific task and process phase. Being able to measure the exact time spent on any given artifact or task, and not rely simply on the estimates and reports provided by the developers, that may contain errors, is a clear advantage for the whole team.

These data can of course be of interest even to researchers, if they help to expose mechanisms and behaviors underlying the development process, spot critical phases and activities, prevent errors.

## 8. Related Works

There are a number of nascent or on-going projects similar to XPSuite and suggesting that the integration of IDEs with server side XP management tools is recognized as a valid option. Among others we mention:

- **xplanner-idea**: a plug-in for IntelliJ IDEA which connects it to the XPlanner tool (http://sourceforge.net/projects/xplanner), using a SOAP API;
- **ecliXPweb**: a plug-in for the Eclipse IDE which allows to connect to a XPWeb database.

In our opinion, it may be of some advantage to define a common protocol for the communication between IDEs and server side project management tools, in order to facilitate interoperability between clients and third-party servers.

## 9. Conclusion

In this article we have outlined the nuts and bolts of XPSuite, a set of tools aimed to provide support and assist metrics gathering in a

XP project. The tools allow multi-channel interaction with process data, both via Web pages and by means of rich user interfaces, directly embedded in the IDE. Although the tools have been used for internal development and have proven effective for supporting activities of dispersed teams, they are in an early-stage of development and we would like to have other teams use these tools to obtain valuable feedback.

## 10. Acknowledgements

This work was supported by MAPS (Agile Methodologies for Software Production) research project, contract/grant sponsor: FIRB research fund of MIUR, contract/grant number: RBNE01JRK8.

## REFERENCES

1. Manifesto for agile software development. http://agilemanifesto.org.
2. Kent Beck. *Extreme Programming Explained: Embracing Change*. Addison-Wesley, 1999.
3. Ward Cunningham and Bo Leuf. *The Wiki Way*. Addison-Wesley, 2001.
4. Ron Jeffries. A metric leading to agility, June 2004. http://www.xprogramming.com/xpmag/jatRtsMetric.htm.
5. Sandro Pinna and al. Developing a Tool Supporting XP Process. In *Extreme Programming and Agile Methods, Lecture Notes in Computer Science*, pages 151–160. Springer, 2003.
6. Sandro Pinna and al. XPSwiki: an Agile Tool Supporting the Planning Game. In *Proceedings of the 4th international conference XP2003*, pages 104–113. Springer, 2003.
7. Swiki: the wiki engine written in Smalltalk. http://minnow.cc.gatech.edu/swiki.
8. XPSwiki: an open source web tool for eXtreme Programming teams. http://agile.diee.unica.it/xpswiki.