Computer Science • Vol. 7 • 2005

Jacek Dajda*, Stanisław Ciszewski**

SUPPORT FOR DISTRIBUTED PROGRAMMING IN EXTREME STYLE

The basic limitation emerging from practising eXtreme Programming methodology is the constraint of close physical proximity between the members of the collaborating team including customer. This became the main idea behind research on XP supporting environment for geographically distributed teams. This work presents basic assumptions, elaborated architecture and selected implementation issues for the system of this type. Deliberations are supplied with the initial results of the verification of its usability based on the users tests.

Keywords: *eXtreme Programming support, Distributed eXtreme Programming, Virtual Pair Programming, Virtual Teaming*

WSPOMAGANIE ZDALNEGO PROGRAMOWANIA W STYLU EKSTREMALNYM

Podstawowym ograniczeniem, jakie nakłada na zespół metodologia programowania eXtremalnego, jest wymóg bezpośredniego kontaktu pomiędzy członkami grupy programistycznej w tym klientem. Z tego też względu podjęto prace nad środowiskiem wspierającymi ten styl programowania, przeznaczonym dla grup rozproszonych geograficznie. Praca prezentuje podstawowe założenia, wypracowaną architekturę i wybrane aspekty implementacyjne takiego systemu. Rozważania uzupełniają wstępne wyniki weryfikacji jego przydatności, uzyskane w ramach obserwacji grup jego użytkowników.

Słowa kluczowe: *wspomaganie programowania ekstremalnego, zdalne programowanie ekstremalne, zdalne programowanie w parach, zdalne zespoły*

1. Introduction

The eXtreme Programming (XP) provides a set of designing and programming methodologies which involves a close physical proximity of both team members and customer.

Nowadays, with an increasing number of projects conducted in a geographically distributed teams, this requirement can be treated as a considerable limitation. To overcome it, a new concept called DXP (Distributed XP) has been introduced. Its main goal is adopting core eXtreme Programming practices and principles into

*PhD Student EAIiE, AGH-UST, Kraków, Poland, jdajda@icslab.agh.edu.pl
**Institute of Computer Science, AGH-UST, Kraków, Poland

the distributed setting. This basically requires a support from a devoted tools and environments making the communication convenient and efficient regardless on the distance between cooperating people.

Several attempts have been made in order to fulfill these needs. Work [2] presents a solution called MILOS ASE which is meant to faciliate team communication, coordination and information routing by a story management system. For pair programming purposes Microsoft NetMeeting and video conferencing have been put into use. TUKAN environment presented in [3] is another approach to DXP support. It provides developers with share code repository, synchronous communication and collaboration as well as version management and distributed integration support. Practicing pair programming is feasible due to the video-voice link and other visual communication means (e.g. virtual mouse cursors). Video conferencing approach is a main idea behind DXP devoted environment summarized in [4, 5]. It uses large screens turning simple room into a virtual office space, similar to these ones described in [6] ("Office of the Future") and [7] ("Office of Real Soon Now"). Whiteboard application and a suitable plug-in for the Eclipse IDE are reported to be under development.

Interesting case has been reported in [8] which describes a development failure of requirement management tool called Storymanager. The idea behind it was to manage user stories and tasks by putting them into an electronic format available for all people involved. The StoryManager was developed as a plug-in to Eclipse framework.

In our case [1], supporting distributed pair programming and continuous integration was the primary goal. In addition we suggested simple on-line application synchronizing the work in distributed team. The final environment has been assembled from already existing tools and solutions like CVS and CruiseControl [9]. For pair programming we developed a devoted plug-in to Jext [10] with voice link and synchronized text editors.

The role of this article is to present achieved results with an emphasis on their efficiency in practical usage. Some useful tips for next experiments concerning creating a DXP support might be provided by describing occurred problems and applied solutions .

In Section 2 we give an overview on our approach to developed application accompanied by a list of requirements that need to be satisfied in the DXP supporting tool. Section 3 provides a description of the elaborated architecture. Occurred problems and interesting implementation details are the subject of Section 4. In Section 5 the final effect is presented with a report from the conducted testing experiment. We conclude with a summary and future work planning.

2. System features

The creation of new environment requires suitable preparation and planning. This includes specification of system requirements, selecting available resources that will be used as well as deciding on the most appropriate approach to the development process. In this section we will overview these issues with a regard to our project [1].

2.1. General touchstones

As repeating the work which is already done is considered useless and jeopardizing rapid development, we decided on reusing existing resources. This seemed most preferable approach due to our time limitations and extent of the DXP supporting solution. This allowed us to accelerate the development process. On the other hand, it put us at risk of integration and maintenance problems, which are likely to appear when a variety of different resources is being employed.

Before the particular components of the developed environment were chosen, we attempted to specify the list of basic touchstones, which helped us finding the most appropriate solutions for our purpose.

Among them, we emphasized the following issues:

- licence and distribution matters,
- portability and flexibility,
- easiness of use,
- development status.

Since that was an academic project and we could not afford to use any commercial solutions, open source software became of our choice. The portability of the used tools seemed obvious as the created system should be available for cooperating developers working in different environments. As for the development status, it is preferable to choose resources, which are still being developed rather than concentrate on tools that are no longer improved.

2.2. User requirements specification

To provide a considerable support for distributed development in extreme style, the proposed solution should enable the developers to practise the XP core rules, at least the most fundamental ones. This includes programming in pairs and coordination of their work as well as continuous integration and testing.

There were considered a base for forming the general system requirements for the created solution:

- Virtual Coding,
- Communication Links,
- Remote Code Sharing,
- Automated Builds,
- Test support,
- Work coordination.

Virtual Coding assumes providing collaborating developers with a way to mutually operate on a created code. It involves synchronization of the current document state and the general view of the used editor. In addition, we decided that suitable mechanism concerning the change of keyboard ownership state and the possibility of marking specific code fragments in the remote document should be provided.

Communication Links requires the created system to offer cooperating developers a voice connection. In case of connection problems, a simple text chat should be available as well. Moreover, we emphasized that all the connection procedures should be simplified and automated, not to concern the developers with the configuration details.

The goal for defining *Remote Code Sharing* feature was pointing out the need for a convenient access to the code repository for all the members of the distributed team. Due to a possibility of parallel code modifications, a versioning support should be included as well.

Automated Builds cover the call for an automated compilation of the created code stored in the code repository. The integration build should be performed automatically after each modification to the source code. Obviously, build results should be easily accessible for all team members.

Test support requirement has been added in order to provide automated test runs performed during every build. In order to facilitate test implementation, test coverage report should be included as well. Clearly, all the results must be available for every developer.

Finally, we specified *Work coordination* feature to allow distributed developers for making pair programming sessions appointments. We agreed that it should supply users with a list of all scheduled sessions with information on the reserved tasks and source code fragments. These aspects are undisputable when it comes to synchronizing work inside a virtual team.

We decided that these six collected aspects cover the general needs of a DXP supporting solution and meet the most significant expectations towards it.

2.3. Components selection

Having specified requirements concerning the reused solutions and key features of the developed environment, we performed a selection of the resources and technologies, which were used to build the final result.

To enable virtual pair programming (which includes *Virtual Coding* and *Communication Links* features) we created our own tool, which consisted of:

- Text editor – for this purpose we created a DXP plugin to Jext Editor [10]. It has become of our choice due to its support for extendable plugins and different programming languages. In addition, it has been created in Java that answers the portability and licencing needs. Finally, it offers a simplicity and low system requirements, which are not to be neglected when a flexible and configurable environment is created.
- Communication server – we used object oriented communication mechanism offered by Java Remote Method Invocation (RMI) system.
- Voice communicator – it has been created using the Java Media Framework (JMF) API and integrated with the Jext DXP plugin.

As for the selection of text editor, it should be added that we did not take under consideration IDE kind of environments (such as JCreator or Eclipse) as we found the light-weight, flexibility and openness to a variety of programming languages of simple editor more preferable.

To provide the *Remote Code Sharing* functionality, we could employ CVS or ClearCase solutions. Since the second one involves purchasing commercial and quite expensive licence, we decided on the open source solution provided by CVS.

Having all the project code stored in a CVS repository, we extended the environment with automatic integration and testing, which have been defined as the *Automated Builds* and *Test support* requirements for the created system. To achieve this we integrated two existing tools:

1) CruiseControl – it offers an automated build process relying on the Ant environment. Hence, the performed build is enabled to include testing. Build and test results are available via suitable web page with additional information concerning time and produced artifacts.
2) JCoverage – it provides test coverage reports which we included as a part of CruiseControl build results.

For *Work coordination* purposes we came up with web oriented system as it offers unconstrained and comfortable access for a variety of environments the cooperating developers may work in. To achieve it, we employed Java JSP environment for user interface and PostgreSQL as a database server for storing session and user data.

To sum up, from existing solutions we employed Jext Editor, Cruise Control, JCoverage and CVS. To provide all lacking functionalities we created: Extreme and Voice Chat plugins for Jext Editor (RMI, JMF) as well as Session Management System (JSP, PostgreSQL).

3. Architecture

Since the proposed environment covers several different aspects of the distributed development process, it can be approached as a combination of corresponding elements, which significantly facilitates its understanding as well as further development.

We distinguished the following three separate functionalities:

1) Work coordination,
2) Virtual collaboration,
3) Integration.

3.1. Used symbols

In order to systemize the architecture description, a set of specific symbols need to be introduced (Fig. 1).

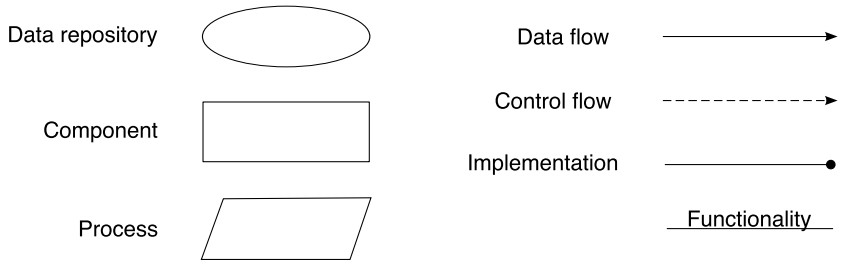

Fig. 1. Used symbols

3.2. Reference model

The role of Reference Model is to present the distinguished system's functionalities and the data flow, which takes place between them. It has been depicted in Figure 2.

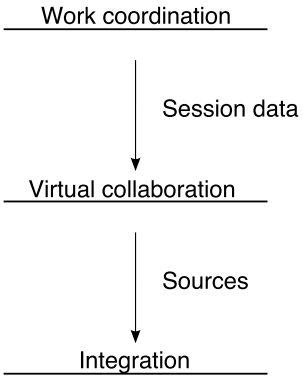

Fig. 2. Reference model of the developed environment

3.3. Architecture model

Virtual collaboration (Fig. 3) represents the user environment in the created solution. It generally consists of the Jext editor supplied with suitable plugins:

- Extreme Plugin,
- Voice Chat Plugin,
- CVS Plugin.

They are used to access CVS repository with shared code and enable user communication using either direct connection (voice data) or *Communication Manager* messaging server (text chat and session data).

Work coordination (Fig. 4) is responsible for both the communication and control issues regarding the distributed pair programming session.

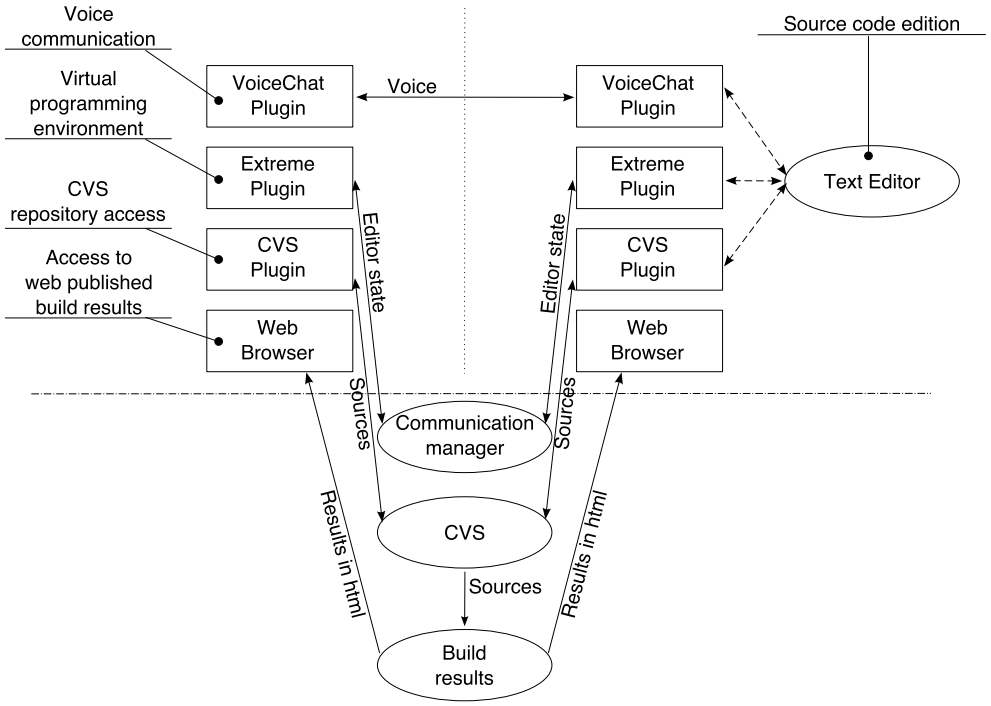

Fig. 3. Virtual collaboration functionality

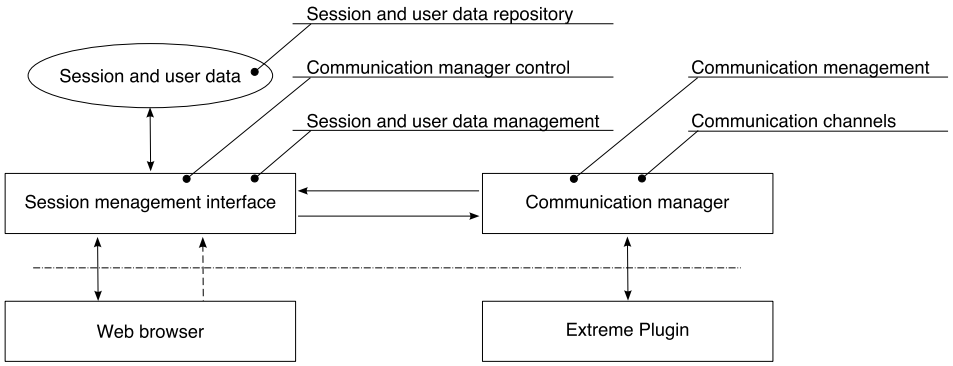

Fig. 4. Work coordination functionality

It consists of a web oriented Session Management Interface (SMI) system and RMI based Communication Manager (called Communication Server as well), which provides a set of functionalities, among which the role of a messaging server between the session participants is of the highest importance.

The Integration functionality (Fig. 5) allows for passing developed code from CVS repository to CruiseControl and JCoverage tools to perform automated builds and integration testing. The results are available through a web page. Therefore, it can be stated that integration domain is accessible from the user environment on two possible ways:

1) CVS repository,
2) web repository of build and test results.

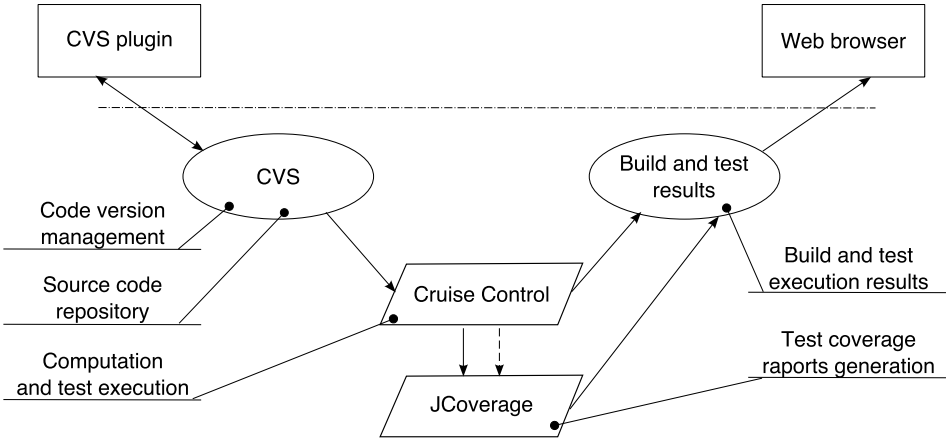

Fig. 5. Integration functionality

4. Implementation overview

With the beginning of system implementation the RMI technology has been chosen as a base communication medium between collaborating text editors. Taking into consideration the ease of sending message objects as well as the ease of communication channel implementation it seemed to be the best solution. However, after the first prototype of the system has been implemented, it turned out that direct RMI calls are not efficient enough if the network connection is quite poor. The application tended to block for the time the message object has been send or received, what was really undesirable. Moreover, as the editor during source code modification generates a number of message objects, the network bandwidth required to comfortable work was really high.

To answer the problem each client part of the communication channel has been equipped with two threads dedicated to send and receive message objects. This en-

sures that the information is sent/received as quickly as possible, without blocking the application for the time the actual transmission is performed. Moreover, if only possible, the message objects are assembled into clusters and sent/received within one RMI call. This results in decreasing the required network capacity and much more comfortable work with the editor.

Another interesting issue was establishing bidirectional voice communication channel, which relied on RTP data transmission. In this case, each side needed to know the IP address as well as the UDP port number of the cooperating editor. The responsibility for providing required information could not be delegated to the editor user due to the inconvenience of this solution. On the other hand, JMF required to use different ports for data input and output. Moreover, it was not possible to fix the port numbers as their availability on different machines could not be guaranteed. Therefore, an automatic support was highly needed.

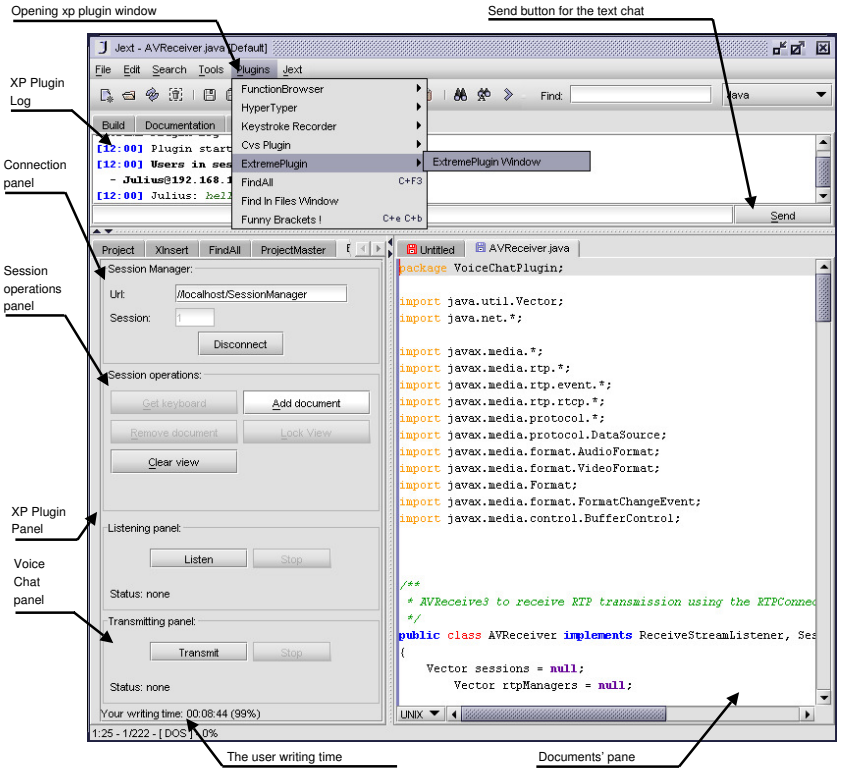

Fig. 6. XP Plugin for Jext Editor

The suggested solution (Fig. 6) consisted of automated port assignment procedure as well as protocol, which was used to exchange the information between two editor entities.

The main goal was to hide all negotiation details for the working user. As a result, the user was only supposed to press suitable button in order to establish voice connection.

Integration of JCoverage and Cruise Control tools (Fig. 7) can be approached as one of the implementation challenges as well. The intention was to provide automated source code compilation, tests execution as well as generation of coverage reports for performed test cases. The main hurdle was to achieve the execution of JCoverage tasks from the Ant build file driven by Cruise Control as well as to adjust the appearance of the JCoverage to the build results generated by Cruise Control. As a solution to these issues a set of configuration files was prepared. It included Cruise Control build loop configuration file and Ant build script, which provided the way to initialize test coverage report from the build process. Unfortunately, in order to spawn JCoverage tasks the full path to its libraries had to be hardcoded in the ant build configuration file.

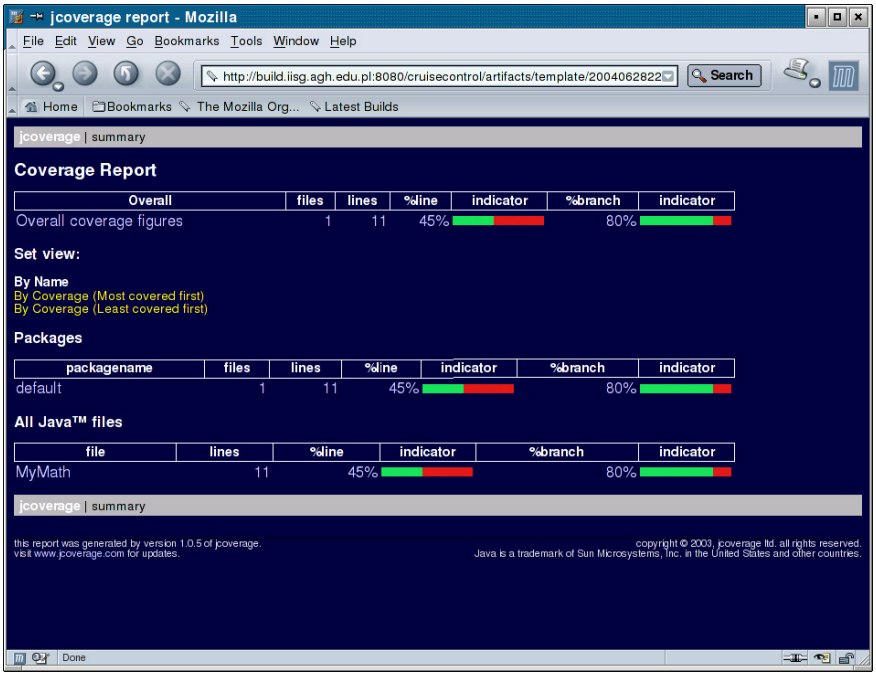

Fig. 7. JCoverage integrated with Cruise Control

5. System presentation

To validate our assumptions regarding the set of implemented functionalities and user-friendliness of interface design, we called for the support of our fellow students. Due to the continuous development of the proposed environment, it was not possible to verify its all included aspects at the same time.

Therefore, the presentation was to be divided into two following stages:

1) Extreme Experiment (EE),
2) Integration test bed.

The goal of EE was to inspect the efficiency, reliability and convenience of the provided distributed pair programming support. With a regard to the second stage, it included code management, integration and testing issues, which built the Integration domain of the given solution.

The invitation to EE was answered by 10 students divided into 5 independently working pairs. Each one received identical task as well as necessary support. The task required information flow between cooperating developers and was successfully completed by all the groups, which proved the usability of provided solution. Generally, judging by the collected comments (see Tab. 1), it can be stated that the presented tool and organized experiment have been approached positively.

Table 1
Users' comments and opinions

Question	Answers	
	Yes	No
Did your team manage to accomplish the task?	10	0
Did the task realization required permanent communication with your partner?	10	0
Did the tested tool disturb task realization in any way?	1	9
Do you think that presented idea of Distributed Team Programming is worth to be further developed?	9	1
Would you like to use such tool in your professional work?	8	2

The received feedback allowed us for the evaluation of certain aspects of the presented solution which we considered most significant and valuable. The average marks for every issue are shown in Table. 2.

Table 2
Evaluation of the presented plugin on the 10-point scale

Aspect	Average rate
Importance of voice communication	8.9
Usability of tested tool according to XP needs	7.66
GUI intuitiveness	7.4
Comfort of the system utilization	7.0
Visual aspect of tested tool	7.1
Keyboard passing method	6.7

Finally, we received a list of suggestions on possible enhancements of the presented plugin:

- Shortcuts – for faster accessing the editor's functionalities.
- Syntax helper – including auto-completion feature, however this was rather a matter of reused editor, not the provided plug-in.
- Automatic keyboard requesting when keyboard pressed – meant for keyboard changing more intuitive and comfortable.
- Bell when keyboard has been lost – a sound indicating the change of keyboard ownership.
- Automatic CRC cards generation – this would facilitate the creation of CRC cards based on the source code.

The second stage of the presentation included verification of the elaborated support for the code maintenance, integration and testing practices of the DXP model. To achieve it, we prepared a devoted environment, which consisted of Cruise Control, CVS and JCoverage solutions. The assigned task did not specify any concrete goals the invited developers were supposed to achieve. Generally, we asked them to familiarize with the configured environment and its utilization, by finalizing a default test package we created for this very occasion.

As a result, a number of useful comments regarding the tested integration environment were produced. The presented combination of CVS, Cruise Control and JCoverage tools occurred suitable for remote automatic builds and tests. A few shortcoming were reported:

- long period between the complication runs,
- unlimited access to all CVS projects,
- Java oriented environment.

6. Conclusion

The principal goal of the carried work was delivering an environment for virtual collaboration of geographically distributed developers. In addition, we attempted to explore the range of available solutions that may be considered a support for the distributed development. Finally, we aimed at verifying the accepted assumptions and producing constitutive suggestions concerning possible enhancements of the followed approach.

The scope of the final solution covered four main XP aspects: Pair Programming, Continuous Integration, Testing and Work Coordination. We managed to validate the provided Pair Programming support during the organized experiment as well as initialize a test installation for automated tests and builds. The achieved effect confirms the feasibility of virtual collaboration in XP style as well as manifests the potential and capabilities of the new software development discipline, which DXP is gradually becoming.

Future work should include further investigation of the key functionalities of DXP supporting environment as well as the improvement of the current ideas and provided solutions. This should be achieved by applying a variety of experiments and test installations, which is a natural step towards validation of the theoretical work.

References

[1] Dajda J., Sztuka T.: *Support for distributed programming in eXtreme style.* Master thesis, Cracow, 2004

[2] Maurer F., Martel S.: *Process Support for Distributed Extreme Programming Teams.* University of Calgary, Department of Computer Science

[3] Succi G., Marchesi M.: *eXtreme Programming Examined.* Addison-Wesley, 2001

[4] Stotts D., Smith J., Williams L.: *eXtreme Hypervideo Support for Distributed Extreme Programming.* Dept of Computer Science, Univ. of North Carolina at Chapel Hill, Technical Report TR02-009

[5] Stotts D., Williams L.: *A Video-enhanced Environment for Distributed Extreme Programming.* Dept. of Computer Science at Univ. of North Carolina at Chapel Hill & Dept. of Computer Science at North Carolina State University, 2002

[6] Fuchs H.: *The Office of the Future.* `http://www.cs.unc.edu/~raskar/Office/`

[7] Bishop G., Welch G.: *Working in the Office of 'Real Soon Now'.* IEEE Computer Graphics and Applications, July/August 2000, pp. 76–78

[8] Kaariainen J., Koskela J., Abrahamsson P., Takalo J.: *Improving Requirements Management in Extreme Programming with Tool Support – an Improvement Attempt that Failed.* Copyright 2004 IEEE

[9] Cruise Control Home Page, `http://cruisecontrol.sourceforge.net/`

[10] Jext Editor Home Page, `http://www.jext.org`