# Stabilizing the XP Process Using Specialized Tools

**Martin Lippert, Stefan Roock, Robert Tunkel, Henning Wolf**
University of Hamburg
Computer Science Department, SE group
& APCON Workplace Solutions GmbH
Vogt-Kölln-Straße 30
22527 Hamburg, Germany
{lippert, roock, tunkel, wolf}@jwam.org

**ABSTRACT**
One problem with the XP development process is its fragility. If developers use the XP techniques in an unintended way or not at all, the XP process is likely to break down: The misused techniques affect the other XP techniques in a negative way, breaking the whole process.

We believe that it is possible to stabilize the XP process using specialized artifacts to reify the XP techniques. We discuss the reification of the XP technique *Continuous Integration* using the JWAM IntegrationServer as an example. We present our experience with this tool and analyze its effects on the other XP techniques.

**Keywords**

eXtreme Programming, process, artifacts, continuous integration, team development, tool support.

## 1 MOTIVATION

Extreme Programming is a combination of a number of different techniques for developing software. These techniques are not independent, but rather influence and complement each other. Kent Beck shows their relationships in [1].

Over the past two years, we have used XP techniques in various development projects. Here, we observed that the XP process is fragile: if one XP technique is used in an unintended way or not at all, dependent XP techniques may be affected. The problem is that XP is based on discipline and experience. If a team lacks the necessary discipline or experience, the XP process is likely to break down. The role of a XP coach is therefore suggested by Beck. The XP coach has the required experience and tries to establish the XP values within the XP team. He/she is the conscience of the team.

However, this situation is unsatisfactory. Many small teams do not have the financial resources to pay for an XP coach, and even if they do, it is very hard to a get a good one. So farm there have not been many successful XP projects that have produced skilled coaches.

Fortunately, there are other ways of transferring experience and knowledge than by coaches. One is to reify successful routines and behavior in artifacts (cf. [2]). This is what cultures do: if a routine is executed over and over again in the same or a similar way, the culture will create an artifact that reifies this routine. A carpenter hammers nails into wood. The reification of this routine is the hammer. The hammer does not force the carpenter to use it as intended, or even to use it at all, but it helps the carpenter to do his work effectively. Thus, the carpenter will, of course, use the hammer. At the same time, the hammer "helps" the carpenter to remember how to hammer. This facility is more important for an inexperienced carpenter than for an experienced one. The hammer helps to transfer the knowledge about how to hammer from the experienced carpenter to the inexperienced one. The hammer stabilizes the routine of hammering.

The idea of using artifacts to reify proven routines and practices is not only useful for XP projects without a coach. Even with an XP coach, the artifacts help to stabilize the good practices and techniques. It is crucial to use artifacts as part of the game, but the artifacts and routines have to be complemented by a value system. Artifacts, routines and value systems stabilize each other.

In this paper, we focus on the *Continuous Integration* technique of XP. We present the JWAM IntegrationServer as one possible artifact reifying this technique and therefore stabilizing the XP process.

## 2 Reification of the integration process in AN ARTIFACT

The dependency diagram in Figure 6 focuses on Continuous Integration, which is shown to be directly related to Collective Ownership, Coding Standards, Testing, Refactoring, Pair-Programming and Short Releases. If a team has problems with Continuous Integration, this is likely to cause problems with other XP techniques. But if Continuous Integration works well, it should support many other XP techniques.

When we started using XP techniques, Continuous Integration was one of the first we adopted. We tried to establish the technique as suggested in [1]. We used one physical integration machine for the whole team (about 8 developers) which always keeps a consistent and running version of the system under development. However, this did not work as expected. We achieved only a few integrations per week and the team's motivation was

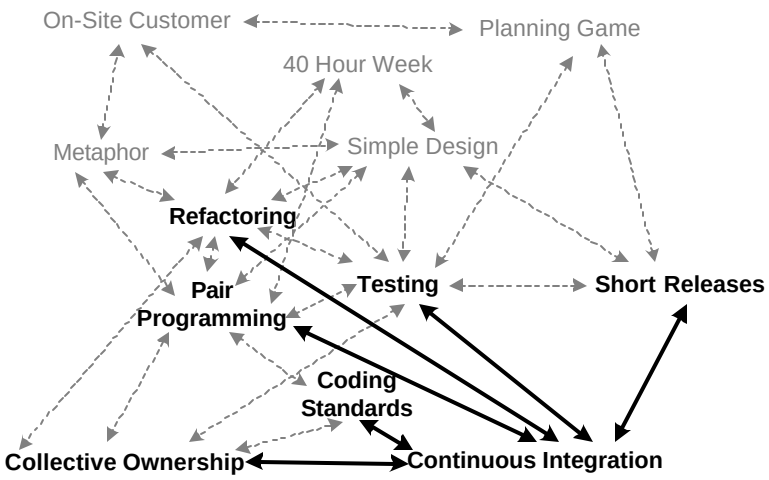


**Figure 6: Dependencies between XP techniques, with highlighted dependencies between Continuous Integration and the related XP**

adversely affected.

The reason for the failure to establish Continuous Integration was easily found: each integration took too much time. There were too many things to, such as:

- Find out what changed on the client and on the server since the last synchronization of server and client.
- Merge the modifications done on the client.
- Compile server sources.
- Test server sources.

Since we had no tools to support the integration process, only some conventions and to-do lists were available to guide the developer through the integration process. Especially occurring problems were a pain. Sometimes, after a merge, the test cases would not run any more. In that case, we had two possibilities: we could remove the problem on the integration server machine, blocking the server for further integrations during this period, or we could restore the previous state of the integration machine and remove the problem on the client.

A single integration, for example, often took more than 2-3 hours. The integration process, then, was laborious and nobody wanted to do it. Another negative aspect of the integration process was the fact that the integration machine was located in a different room from where most of the development was done. These two facts, the long and complicated integration process on a totally different machine and the spatially separated integration machine were a considerable handicap for the people doing the integration. This had a further impact on other XP techniques, especially refactoring. Since integration was done infrequently, the refactorings were very large. We were therefore very often without running system versions, which hindered Short Releases. During the large refactorings, the test cases often broke down and had to be more or less rewritten after refactoring. This hampered the XP technique of Testing.

We started to look for a more suitable integration method to restabilize our XP process.

We liked the idea of using an artifact to support continuous integration within our development team. But what sort of artifact would be suitable? Using one integration machine was unsuccessful, as were to-do lists.

A tool might be the right answer to this question. CVS, Envy or other source-code-management systems could be used to reify the continuous integration technique. The important thing is that the tool is accepted by the team members and that it or its use supports the principles of XP Continuous Integration. CVS or other source-code- or version-management systems can be used in combination with conventions specifying how to use them. One convention might be that every developer has to download the current version before the next integration, update the changed source codes and conduct all the tests to ensure that all test cases run with the integrated version. This is one way of using a tool in combination with conventions to stabilize continuous integration within the team. We identified the following points as important for the Continuous Integration technique:

- Very short integration cycles. We consider a few minutes, no more to be optimal. This allows us to realize smooth team development without adversely affecting other members of the team.
- Unit testing is essential. Ideal would be a tool that takes care of the correctness of all sources at every integration.
- And last but not least: the tool and the conventions should be as easy to use as possible.

We decided to develop a specialized tool, the JWAM IntegrationServer as an extension to CVS, to provide our developers with a smooth way of dealing with continuous integration. Based on our positive experience with the tool, we present its basic functionality in the following section. We would, however, like to emphasize that any other source-code-management system may be useful for reifying continuous integration with the right set of conventions and values.

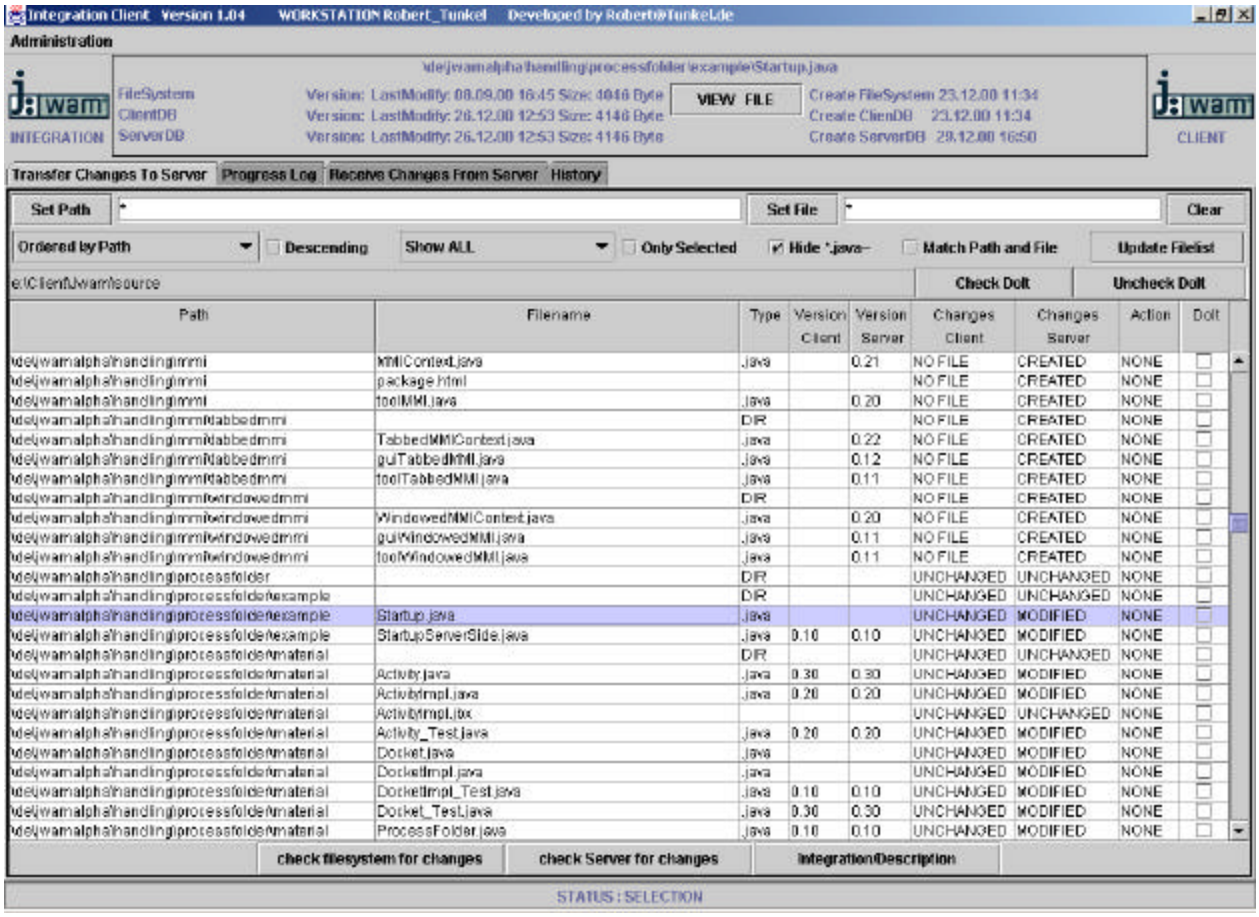


**Figure 7: The *IntegrationClient* user interface**

## 3 JWAM IntegrationServer

Given the requirements and observations mentioned above, we developed a specialized tool to support the continuous integration process. This tool can be thought of as a unit-testing addition to a normal version-management system. It is based on two components:

- The *IntegrationServer* is a server process that runs on a server machine that is accessible to all developers. This server machine defines the reference machine and controls and manages the complete source code using a version-management system.
- The *IntegrationClient* is a client-side tool that allows the user to update source code on the reference machine as well as obtaining changed source code from it. The client tool works as the artifact for the developer and offers the user an easy-to-use interface.

The typical use case for this tool is: the developer has changed a number of pieces of source code and would like to integrate them to provide the other developers inside the team with the changes. This task is supported by the *IntegrationClient*, where the developer can list the files he/she has changed. The developer can then integrate them by simply pushing a button. This single action guarantees that:

1. the changed pieces of source code are transferred to the server reference machine
2. the complete source code is compiled
3. all test cases run on the reference machine

Only if all three actions are successful and no failure occurs during unit testing are the updated pieces of source code accepted on the reference machine and the integration successful. If one unit test fails or an error occurs during one of the actions, the integration is cancelled. As a result of a failed integration, the changed versions of the pieces of source code are rejected and none of the code base is changed on the reference machine. Feedback about a successful or failed integration is, of course, reported to the developer sitting in front of the *IntegrationClient* tool. The main user interface of the *IntegrationClient* tool is shown in Figure 2.

Test cases are handled as normal source code by the IntegrationClient. The developer can add, modify or delete test-case classes using the same tool as for every other piece of source code.

The other use case that often occurs is where another

developer has changed some source code on the reference machine. In this case, the *IntegrationClient* shows all changed pieces of source code and the developer has the opportunity to download all new or changed pieces of source code to his/her development machine.

Another interesting case is the occurrence of conflicts, e.g. when two developers on the team have changed the same source code. In this case, the *IntegrationClient* indicates the conflict and does not allow the changed source code to be uploaded to the server. First, the developer has to download the changed version from the server, transfer his/her changes into this source code and then integrate the new merged version. This can be supported by a Diff- or Merge-like tool.

## 4 Experience

We have been using the tool since April 2000 and our experience during more than 2,000 integrations has been extremely positive. The tool is easy to use and enables changed source code to be easily integrated. An integration is done in less than 10 minutes for large projects (thousands of source files) and in a few seconds for small projects (some hundred source files), the XP idea of Continuous Integration being optimally supported. This makes the tool very attractive for developers. The programming pair can take a break while the integration process is running. Results and possible errors are reported in a progress log.

Our use of the tool suggests that small changes and frequent integrations are best. This is because all developers like the tool and know how to use it. And the tool lists all differences between the source code on the reference machine and the individual developer's source base. This makes it very easy to see whether someone else has changed a piece of source code which could have serious effects on my changes.

Since the IntegrationServer incorporates an optimistic locking strategy, the first developer to integrate modified sources wins. If the integration is refused by the IntegrationServer, the developer who tried to integrate has to remove the problem. This leads the developers to integrate as fast as possible to avoid potential conflicts. Thus, the reification of the Continuous Integration technique in a tool as an artifact supports not only this technique but the refactoring technique as well. Developers are "forced" by the IntegrationServer to make small refactorings rather than large ones.

The fact that the IntegrationServer ensures a running version on the server supports the Small Releases technique, too. In principle, it is possible to deliver a new version every day. The technique of Collective Code Ownership is also supported by the IntegrationServer because the developer that caused a conflict has to remove it – no matter where code he/she has to modify. Testing is supported by the IntegrationServer's testing facility. The developers know that the IntegrationServer will execute their tests over and over again and will avoid breaking their code. Developers thus experience the benefits of test cases and are willing to write test cases for their code.

In our experience, the IntegrationServer does not specifically support the XP techniques Pair Programming and Coding Standards. These techniques are supported by Continuous Integration as a technique, and not by the IntegrationServer.

## 5 Related work

There are a number of different tools that can be used as an artifact for the Continuous Integration process. As mentioned before, CVS, Envy or TeamStreams by Object Technoloy International (see [3]) may be called to mind. They are all useful for developing with XP. The IntegrationServer only adds a special, extremely easy-to-use and smooth interface and automated testing of the integrated version, which makes it easier to use for our XP projects.

There are other artifacts that may be useful for reifying other XP techniques. The Refactoring Browser, for example, might be used for the Refactoring technique[1].

## 6 Conclusion and OUTLOOK

The reification of the Continuous Integration process using a specialized tool as an artifact works well. In particular the shift from non-XP development to XP was stabilized by this tool.

The IntegrationServer is only a first step toward a set of artifacts stabilizing the XP process. In addition to presenting the IntegrationServer, this paper is intended to provoke a discussion on other artifacts suitable for stabilizing the XP process. These we are still looking for.

## 7 References


1. Kent Beck: *eXtreme Programming Explained – Embrace Change*, Addison-Wesley, Reading, Massachusetts, 1999.
2. Keld Bødker, Jesper Strandgaard Pedersen: Workplace Cultures: Looking at Artifacts, Symbols and Practices. In: Joan Greenbaum, Morten Kyng (Eds.): *Design at Work*. Lawrence Erlbaum Associates, Publishers, Hillsdale, New Jersey, pp. 121-138. 1991.
3. Jim des Rivières, Erich Gamma, Kai-Uwe Mätzel, Ivan Moore, André Weinand, John Wiegand: Team Streams – Extreme Team Support, in *Proceedings of eXtreme Programming and Flexible Processes in Software Engineering - XP 2000*, Cagliari, Sardinia, Italy, June 2000.
4. JWAM website: http://www.jwam.org


[1] Thanks to the reviewer who suggested this idea.