Raising the Awareness of Development Progress in Distributed Agile Projects

Sultan Alyahya
King Saud University/College of Computer & Information Sciences, Riyadh, Saudi Arabia
Email: sualyahya@ksu.edu.sa

Wendy K. Ivins and W.A. Gray
Cardiff University/School of Computer Science & Informatics, Cardiff, UK
Email: { W.K.Ivins, W.A.Gray }@cs.cardiff.ac.uk

***Abstract*— This article discusses a computer-based holistic approach to raise the awareness of development progress in distributed agile projects. The approach analyses how the technical activities (i.e. source code versioning, unit testing, acceptance testing, integration and releasing) affects development progress and provides automatic mechanisms that help co-ordinate progress change with distributed team members. The approach has been evaluated through practical scenarios and has validated these through a research prototype. The result shows that the use of the holistic approach can provide better awareness of progress to team members than the traditional approach that relies on informal communications.**

***Index Terms*—Agile Development, Progress Tracking, Awareness, Co-ordination, Distributed Teams.**

I. Introduction

Awareness refers to the knowledge of the state or activities carried out on a team member's work which provides a context for the work of others [1] [2]. In agile development, team members may carry out several technical activities that may affect the progress state of others' work.

Progress state is judged in agile methods mainly based on the state of the essential output of the project which is basically the software that the customer will use. Several technical factors affect the state of the software. These factors include source code versioning, unit testing (UT), acceptance testing (AT), continuous integration (CI), and releasing. These factors play a role in determining whether software produced for a user story (i.e. feature or use case) is 'working software' (i.e. the user story is complete) or not. One of the principles introduced by the agile manifesto [8] is that *working software is the primary measure of progress*. We propose that each of the technical factors impacts the progress towards working software.

Therefore, it is required to monitor the status of the source code and manage the technical activities that affect it. The source code versioning, unit testing, acceptance testing, continuous integration and releasing imply technical activities have an effect on the software progress.

Agile teams can raise the awareness about development progress through direct interactions between team members. XP [3] (one of the most used agile methods) supports raising awareness of progress in co-located teams by two main practices: *Sit Together* and *Informative Workspace*. When team members sit together, they are expected to share information about factors that may affect the progress of the project through face-to-face communication. The ad-hoc co-ordination is likely to facilitate partial sharing of the progress information amongst team members. XP teams are also encouraged to surround the workspace with rich information about the state of tasks, stories and tests that are updated continuously. They often use big charts (e.g. burn charts) to visualise the development progress.

In distributed agile projects, team members lose the physical contact that helps them maintain awareness of the actual progress; hence, they replace it with informal methods such as video-conferencing meetings. Unfortunately, the informal methods are not sufficient. A significant limitation of the informal methods is that the impact of a progress change resulting from the technical activities may not be fully recognised by the team members. This is because of the difficulty of understanding how the work of one team member at one site influences the work of another team member at a different site.

Distributed agile projects use also formal methods such as agile project management tools to manage development progress. These tools facilitate sharing progress information about iterations' tasks and user stories and provide basic progress status notifications. For instance, if a task is delayed, a team leader can be notified. However, these tools fail in identifying the impact of technical activities on development progress. This is because they rely on changes in progress, caused by the technical activities, to be flagged in the system by team members.

Xu [4] observes that "teams have a difficult time keeping track of progress" in a distributed agile project. Teams may end an iteration having a large number of

doi:10.4304/jsw.8.12.3066-3081

failed acceptance tests, delivering progress information late and to the wrong team members. Jeff Patton, a team leader in several agile projects, states that he noticed many agile organisations struggling to keep track of the acceptance tests. He states [5]:

"*When an acceptance test fails, it's usually a long time after the offending code has been checked in. In fact, a lot of code may have been checked in. This makes finding the offending code difficult*".

In our research investigation, we attempt to support the effective management of progress by providing a computer-based holistic approach to managing development progress that aims to explicitly identify and co-ordinate the effects of the various technical factors on progress. The holistic approach will help raise awareness of distributed agile teams about change in progress as soon as it occurs. It supports the need for continuous feedback about project state which can potentially reduce the testing bottlenecks at the end of each iteration and release.

This article extends our work in [6] as follows: first, it discusses the concept of progress in agile methods and how it differs from the meaning of progress in the plan-driven approach. Second, the article surveys the popularity of the use of the technical factors among the agile methods. Third, the article also provides extensive literature review of the current approaches used to manage progress. This includes a study of the progress mechanisms in thirty agile project management tools. Finally, this article evaluates the proposed approach through two scenarios that have not been previously published.

II. The Concept of Progress in Agile Approach

Progress in the plan-driven methodologies is often based on the completion of deliverables such as the requirement specification document and analysis and design diagrams. It is difficult to judge progress based on these deliverables [7]. The progress reports may not reflect how healthy the project is. For instance, the progress report for a project that is at the end of the design phase may show that the project progresses well as all design diagrams are completed. However, team members may find many problems later in the integration phase or the testing phase. Software teams may struggle keeping all deliverables consistent when change occurs. Additional time is spent in developing these extra artefacts that are not software.

The view in agile methods is different from the view in the plan-driven approach. Progress status is judged in agile methods mainly based on the software that the customer will use. Principle 7 in the Agile Manifesto [8] states that: *Working software is the primary measure of progress*. Working software implies that the software is unit-tested, integrated and acceptance-tested by the customer. Thus, activities involved in the technical factors: *unit testing*, *acceptance testing*, *continuous integration*, *source code versioning* may affect working software that is delivered to the customer during the *releasing* process. Key technical activities affecting development progress are shown in Table I.

TABLE I.
KEY TECHNICAL ACTIVITIES AFFECTING AGILE DEVELOPMENT PROGRESS.

Unit-Testing (UT)	Acceptance-Testing (AT)	Continuous Integration (CI) & Releasing	Source Code Versioning
-Create a new UT -Update existing UT -Delete UT -Run UT	-Create a new AT -Update existing AT -Delete AT -Run AT	- Perform integration - Make a release	- Create an artefact. -Modify an artefact -Delete an artefact

III. The Technical Factors in Agile Methods

The popularity of the technical factors among the agile methods was surveyed. The results showed that most agile methods recommended using UT, AT, CI, releasing, and source code versioning (see Table II). Most methods have explicitly mentioned them in the formal methods description.

TABLE II.
THE TECHNICAL FACTORS IN AGILE METHODS.
(KEY: ● EXPLICITLY MENTIONED ○ IMPLICITLY MENTIONED)

Agile Method	UT	AT	CI	Releasing	Versioning
Extreme Programming	●	●	●	●	○
Scrum	○	○	○	●	○
Crystal Family of Methodologies	●	●	●	●	●
Dynamic Systems Development	●	●	●	●	●
Adaptive Software Development	○	○	○	●	●
Agile Modeling	●	●	●	●	●
Pragmatic Programming	●	●	●	●	●
Feature-Driven Development	●	●	●	●	●

However, not all the technical factors have been mentioned explicitly in some agile methods:

- Source code versioning is implicitly mentioned in XP
- UT, AT, CI and source code versioning are implicitly mentioned in Scrum
- UT, AT and CI are implicitly mentioned in Adaptive Software Development

XP did not explicitly emphasis the importance of using systems to manage source code versions in XP projects. Paulk [9] states that SCM is partially addressed in XP via collective ownership, continuous integration and small releases. However, the literature on XP emphasises clearly the need for versioning systems (e.g. [46] [10]). Furthermore, the focus in Scrum and ASD is not on the development techniques. Scrum has focused on providing a project management framework, while ASD's primary focus is on the problems of developing large and complex

systems. Scrum and ASD provide very few practices for day-to-day software development work [11]. These methods state that they welcome practices from other methodologies for use in the development.

Regardless of the agile method applied, the literature and survey show that UT, AT, CI, releasing, and source code versioning have been widely adopted in agile projects (e.g. [12] [13]).

IV. Currents Methods to Managing Progress of Distributed Agile Teams

The primary methods used by distributed agile projects to manage development progress can be divided into two approaches: informal methods and formal methods. These approaches have been extensively reviewed for this section.

A. Informal Methods

Distributed agile teams use several informal methods to track progress information. The main informal methods are synchronous communication, asynchronous communication, daily tracker, information radiators and cross-location visits. These are discussed below.

1) Synchronous communication

In distributed teams, meetings can be held by synchronous tools such as audio and video-conferencing tools. Stand-up meetings may be held for about half an hour everyday using these tools. Other meetings can be scheduled weekly and monthly.

In addition, some teams may use instant messaging (IMs) for one-to-one communication between team members. These are likely to be used in situations where developers need to communicate personally about issues such as coding aspects or design aspects or any clarifications.

A large number of case studies about distributed agile projects reported difficulties in using synchronous communication (e.g. [14] [15] [16]). Some of these difficulties include: cultural and language differences, spending significant time in resolving technical issues, such as sound quality and, time-zone differences. These issues may cause synchronous meetings to be held less frequently than physical stand-up meetings in co-located projects.

2) Asynchronous communication

Distributed teams may prefer using synchronous tools only for the major progress update events and rely more on asynchronous communication tools, such as e-mail and community discussion boards. The asynchronous tools are cheap, popular, and have fewer technical issues. However, empirical evidence indicates that increasing reliance on asynchronous communication channels can result in higher software defect rates [17]. Kajko-Mattsson et al. [18] observed that the use of tools such as email proved to be insufficient for maintaining the daily communication as dictated by the agile values. Using these tools results in a slow turnaround in communication [19] and often causes misunderstandings, due to messages being composed quickly [20]. In addition, managing e-mails and filtering them may become more difficult and burdensome over time as the team and project knowledge grow in size and complexity [21]. It is also expected that some of the shared information will be misunderstood because of culture and language differences and because the body language, voice inflection and emotions are lost through this type of communication.

3) Daily tracker

The daily tracker's role has been used in many distributed agile projects. A couple of times a week, the tracker finds out where everyone is with the iteration [22]. He tracks the individual progress of the developers by asking them how many days they have worked on the tasks and how many more days are left to complete them.

The daily tracker's role helps reporting progress, but does not support the management of the daily dependencies among team members' work, which may affect development progress.

4) Information Radiators

Cockburn suggests having an 'information radiator' in the workspace [23]. An Information Radiator is a screen displaying information (e.g. progress information) in a place where passers-by can see it. It shows team members information they care about without having to ask anyone questions. Examples of the displayed information include burn charts, and state of acceptance tests. Similar to the daily tracker practice, the information radiator can support sharing the daily progress information but it cannot support identifying and managing changes in development progress.

5) Cross-location visits

Cross-location visits have been frequently recommended for distributed agile projects (e.g. [23] [24]). Team members are rotated across project locations often, to work within multiple teams. This helps in solving conflicts and misunderstandings among the distributed teams. In addition to the cost constraint of this practice, the visits do not serve the aim of sharing and managing the daily progress information but only help sharing of the overall progress information.

B. Formal Methods

Distributed agile teams use several formal methods to manage development progress. The key formal methods include Wikis and spreadsheets, traditional project management tools, and agile project management (APM) tools. This section will discuss these methods.

1) Wikis and Spreadsheets

The basic technologies used for managing development progress are Wikis and spreadsheets. These tools allow users to freely create and edit content. However, Wikis and spreadsheets have limitations. Dubakov and Stevens observe that these tools provide little support for working on distributed environments and limited support for progress reporting and progress visibility [25].

A case study applied to a geographically distributed team using a wiki-based system called MASE [26] revealed further problems. When many minds collaborate together in a Wiki repository, it becomes more difficult to search and maintain as users contribute more and more

content into the repository over time. In addition, content albeit useful may be put in the wrong place.

2) Traditional project management tools

Traditional project management tools such as MS Project [27] could be used with agile methods. These tools can show information in PERT charts, Gantt charts and work breakdown structure charts. Based on the surveys in [12] and [28], traditional project management tools have been widely utilised to manage development progress by many distributed agile projects. Most project managers are familiar with traditional tools and it is easier for them to manage iterations using well known tools. Unlike traditional software projects, only a part of the requirement is known when the project starts and new requirements will constantly emerge during development. This makes it unfeasible to follow the progress of the development work with these traditional tools [29]. Recreating the traditional charts whenever a new requirement emerges would take resources out of development work [29]. Furthermore, these tools are not designed for agile development and hence they do not include key progress tracking features, such as burn-down charts and story/task boards.

3) Agile Project Management (APM) Tools

Due to the limitations of the previous tools, a new generation of project management tools are being developed to satisfy the agile approach (e.g. Rally [30], Mingle [31], VersionOne [32], TargetProcess [33]). A review of thirty APM tools (Table III) revealed a number of different mechanisms available to assist in supporting the management of distributed agile development progress. The key progress tracking mechanisms in these tools are web-based task board, progress reporting, time tracking, acceptance testing (AT) tracking and progress notifications. These are discussed below.

Web-Based Task-Board. Task-boards are commonly used in co-located teams to visibly show the progress of tasks/user stories. They show all user stories with their tasks for current iteration. Usually, each user story and task is represented by cards stuck to a board. Distributed teams use a web-based version in imitation of the manual task-boards with easy drag-and-drop facilities. The task board usually has three main columns: Un-started (To Do), In Progress (In Process) and Done.

Nineteen of the tools reviewed have a graphical representation of task-boards while the rest allow merely for textual representation of a task's status. Tools such as On Time [34] and VersionOne [32] enable users to add extra columns such as 'To Be Verified' and 'Tested', which can show more detailed progress information.

Progress Reporting. During each iteration, while the team members are focused on creating the new user stories they have committed to deliver, the project manager is responsible for understanding the progress that the team is making and keeping the customer informed of any potential delays in the development. Most APM tools provide users with graphical reports that show key progress information about the project. These reports include:

- Iteration Burn-down Chart (the work that needs to be completed over an iteration).
- Release Burn-down Chart (the work that needs to be completed over a release).
- Velocity (number of units of work, i.e. user story points, completed over a period of time).

While velocity concerns the work done and how fast it is being done, the burn-down charts allow for forecasting. They allow "what if" analysis to be performed by adding and removing functionality from the release to get a more acceptable date or extend the date to include more functionality [35].

The review revealed that, of the 30 APM tools, 25 tools provide iteration burn-down charts, 17 tools provide release burn-down charts and 24 tools provide automatic calculations of the project's velocity. Some tools, such as eXplainPMT [36], has burn-down charts but it is based on the whole project, not for each iteration nor each release. Further progress charts called Cumulative Flow Diagrams (CFDs) [37] are offered by 11 APM tools. CFDs are constructed by counting the number of user stories that have reached a certain state of development at a given time. CFDs provide further detailed information about the 'Work In Progress' (WIP) state. Common progress points measured are: designing, coding and testing.

Time Tracking. Another progress tracking feature offered by the majority APM tools (21 tools) is time tracking, in which hours spent/hours remaining for each task/story/iteration are presented. It replaces the tracker role in the co-located teams mentioned in the informal

methods. Instead of having every team member entering their time, the time is calculated simply based on when a team member changes a task's status to 'In Progress', and when he sets the task to 'Done'.

The online derivation of time tracking data supports distributed agile projects as team members are scattered over different sites. It also eliminates erroneous data and time wastage problems existing in the manual calculation method.

AT Progress Tracking. Eleven of the APM tools reviewed allow scheduling and tracking acceptance tests' progress during the different iterations. Feedback on tests' progress are provided by a built-in electronic testing board and through various types of AT graphical reports such as the test run progress rate graph which shows number of acceptance tests passed, failed and ready to test.

Acceptance tests are linked with their corresponding user stories. In addition, some APM tools such as VersionOne maintain a full history of each acceptance test which can be used for traceability purposes.

Progress Notifications. An effective progress notification system is an important requirement for managing development progress in distributed agile teams. Team members have to be made aware of the changes in development progress that affect them. Many of the APM tools reviewed provide some support for

progress notifications when there is a change in progress status.

TABLE III.
A REVIEW OF PROGRESS TRACKING MECHANISMS IN APM TOOLS.

Agile Tool	Web-based Task-board	Time Tracking	AT Progress Tracking	Notifications about Task	Notifications about Story	Notifications about AT	Progress Reporting			
							Iteration Burn-down	Release Burn-down	Velocity	CFD
Rally [30]		●	●	●	●	●	●	●	●	●
Mingle [31]	●	●	●	●	●	●	●	●	●	●
VersionOne [32]	●	●	●	○	○	○	●	●	●	●
ScrumWorks [38]	●	●		●	●		●	●	●	
ExtremePlanner [39]	●	●	●	○	○		●	●	●	
XPlanner [40]		●		●			●		●	
TargetProcess [33]	●	●	●	○	○	○	●	●	●	●
Pivotal Tracker [41]	●	●			○			●	●	
Scrum VSTS [42]	●		●				●	●	●	●
Agilefant [43]		●					●		●	
IceScrum [44]	●		●		●		●	●	●	●
Planbox [45]		●			○		●		●	
XP StoryStudio [46]							●		●	
XPWeb [47]									●	
AgileWrap [48]	●	●		●	●		●	●	●	●
ScrumDesk [49]	●	●					●	●	●	
SpiraTeam [50]		●	●	●	●	●	●	●	●	
Leankit [51]	●			○	○					●
DevSuite [52]	●	●	●	●	●	●	●		●	
TinyPM [53]	●	●		●	●		●	●	●	●
Planigle [54]	●				●		●	●	●	●
Acunote [55]		●		●			●			
On Time [34]	●	●		●	●		●			
AgileZen [56]	●									●
ScrumPad [57]	●	●	●	○	○	○	●	●	●	
eXPlainPMT [36]			●						●	
AgileBuddy [58]		●					●	●	●	
Daily Scrum [59]	●	●					●	●	●	
Express [60]	●	●					●		●	
Agile Tracking [61]							●		●	

Fourteen of them provide notifications when the progress status field of a task is changed by a team member, while 16 tools notify when the progress status field of a user story is changed. In addition, 6 tools only provide notifications if there are changes in the progress status field of an acceptance test.

Rally [30] allows team members to set up personalised notifications. They can select the types of event to be notified. Notifications offered by Mingle [31] are classified into three main types: user-generated messages between team members, system-generated alerts from subscriptions, and admin-level announcements to the whole project team. For instance, team members can send messages to raise awareness about new issues or provide immediate visibility of the status change of an asset (i.e. task, story or test). Team members can also set up subscriptions that will alert them if there are changes to a specific asset.

ScrumWorks [38] allows team members to select the period at which they wish to receive notification of changes. Notifications can be sent either immediately when each change occurs, or a once-daily listing of all accumulated changes.

Less robust notification systems are provided by VersionOne and TargetProcess. In VersionOne, team members cannot subscribe to events. Only the asset owner is notified when change occurs. The notification system in TargetProcess is role-based, that is, selecting any of the system roles will send the notification to all members of that particular role in the appropriate project. For example, selecting developer will notify all team members whose project role is developer.

The notification system in Planbox [45] is also limited. The scope of notifications is restricted to user stories only (called *items* in Planbox). Moreover, Planbox does not offer an event subscription service. A team member is notified either when the progress status field of the stories he works on has been changed, or when the progress status field of any story in the project has been changed. Notifications can be triggered by various activities including progress status changes. Conditions can be defined so that only business critical items result in an email notification being sent, for example when an item's status is changed to 'Done'.

Agilewrap [48] sends notifications if a task or story is overdue. In addition, if somebody accepts (story passed

testing) or rejects (story did not pass testing) a user story, the story owner is notified.

ScrumDesk [49] does not provide notifications about specific assets. When the system starts, it displays all changes since the last time the user logged out.

In ScrumPad [57], the asset creator can designate the team members who will receive notifications about change in the asset's progress. This could be a disadvantage as the creator may not know who is affected by his work.

V. A Computer-Based Holistic Approach

In the informal approach, managing progress of distributed agile teams is conducted in an *ad hoc* manner by the individual team members. If a change in progress has been introduced, the originator of the change has to co-ordinate the introduction of the change with other team members affected. A significant limitation of the informal methods is that the impact of the change may not be fully recognised by the team members. This is because of the difficulty of understanding how the work of one team member at one site influences the work of another team member at a different site. Members may not recognise that there is an effect on progress or may not know who is affected. In addition, they may decide not to contact other team members, because of the time it takes to locate and notify the affected people.

The formal approach uses several mechanisms, incorporated into computer systems such as APM tools, that can be used to facilitate managing development progress. The distributed team members use these mechanisms to register, share and report the progress information. However, the main limitation of the formal approach is that the computer systems are static and rely completely on team members to report changes in progress. A team member performs a task and then registers the task's progress status in the system. Changes in progress caused by technical factors, e.g. modifying source code, are not logged by the formal approach; hence, if these changes affect development progress, this will not be discovered.

From the analysis of the informal methods and the formal methods, it is clear that these approaches are insufficient to fully identify and co-ordinate changes in progress caused by the technical activities. It is realised that better progress management support can be achieved by providing a computer-based holistic approach to developing a progress tracking system. The progress tracking system has to have a holistic view from the perspective that it needs to manage the effects of changes not only from the users (team members), but also from the various technical systems that cause changes in progress.

This will first require analysis of the various events that cause change in progress. This includes identifying the co-ordination support necessary for managing these events (Section VI).

The holistic approach will also require designing computer-based mechanisms that take into consideration the impact of technical activities on progress (Section VII). This means that the tracking system must link to the technical systems.

VI. Analysis of Co-ordination Requirements for the Technical Activities

Every technical activity may be carried out in a way that causes a change in development progress. Progress change events need to be recognised as well as the provision of the necessary co-ordination activities (i.e. derived from the co-ordination types discussed in the previous section) that can help managing these events.

This section analyses the progress change events caused by each technical activity and identifies explicitly the co-ordination support required to manage them.

A. Source Code Versioning

Activities involved in source code versioning (create an artefact, update an artefact and delete an artefact) may cause several progress change events that need co-ordination support.

In the case of creating a source code artefact, if the state of corresponding task/story is 'un-started' or 'complete', creating the new artefact for the task/story implies that its state is changed to 'active'. The developer who tries to create the new artefact has to be informed that the task/story is inactive. The state of the task/story has to be changed to 'active' in the progress tracking system. If other team members are affected by the recent task's/story's state, they must be notified.

Updating a source code artefact normally requires that the developer checks out the source code artefact, makes the modification, and then checks it in again. Progress changes resulting from the check-out process are similar to the case of creating a source code artefact.

In the case of having a source code artefact shared between two tasks/stories or more, where the state of one of them is 'complete', making a check-in to the artefact may change the progress state of that task/story. In this case, which tasks/stories have been affected must be identified. The affected team member must also be found and notified.

If an integrated artefact is modified, it will need to be re-integrated. The recent changes may affect progress of other tasks/stories that share the same artefact. Developers who share a previously integrated artefact to complete their tasks/stories should be made aware that it has new version and, therefore, the artefact needs to be re-integrated.

Deleting an integrated artefact may break the build leading to a negative impact on the progress state of a large number of tasks/stories. Affected user stories may need to undergo AT again. If deleting an artefact breaks the build, this needs to be clarified to the developer and deletion may be delayed until the developer discusses the activity with the other affected developers. It is important to identify the impact of deleting the artefact on progress and reflect it in the tracking system. Finding and notifying affected team members are also required.

B. Continuous Integration (CI) and Releasing

Integration and releasing activities can lead to positive/negative progress change. If an integration

process has been performed that failed, team members may not realise which user stories have been negatively affected. The 'failed' result should not affect those stories that do not have new versions entered in the build.

An integration 'pass' result should contribute to making progress on the affected stories. When a successful integration is made, story owners and testers may not know exactly which stories are ready for the acceptance testing stage. If all the functionalities for a story have been completed and integrated, the tester responsible for the story has to be located and notified that the story is now ready for acceptance testing.

Another potential progress change event may result from the releasing process. A set of user stories may be released while some of them have not been fully tested. In this case, the release process should be prevented. The person making the release has to be made aware that releasing should be for complete stories only.

C. Unit Testing (UT)

Activities involved in unit testing include adding, modifying, deleting and running a unit test. These activities may cause several progress change events that need co-ordination support.

Adding or modifying a unit test without re-testing it or with a 'fail' result can affect the corresponding task, if it was complete. It is important to clarify to the developer that state of completed tasks may change due to his activity. It would be safe to prevent the addition or the modification until the unit test passes.

Deleting the only unit test for an artefact of a completed task affects the task's progress. If it is the only unit test for the corresponding source code artefact, and if the corresponding task is complete, the task state may be affected. It is required to prevent the deletion.

Furthermore, a unit test may not have passed when its corresponding source code version is checked in. In this case, it may affect development progress because getting the unit tests to pass is a necessary part of completing the source code artefacts that are developed to fulfil requirements of a task. Sharing new source code versions should be prevented if the corresponding unit tests have failed.

Similarly, a unit test may not have passed when a developer wants to set its corresponding task to 'complete'. It is required then to prevent setting the task to 'complete' until all its source code artefacts are successfully unit-tested.

D. Acceptance Testing (AT)

Activities resulting from manual and automated acceptance testing can cause several progress change events that need co-ordination support. These progress change events are discussed below.

Adding or modifying an acceptance test without testing it, or with a 'fail' result, can affect the corresponding story if it was complete. States of the corresponding completed stories may need to be changed. Finding and notifying the owner of the completed story may be required.

Deleting the only acceptance test for a complete user story affects the story's progress. If it is the only acceptance test for the user story, and if the story is complete, the story's state may need to be changed. The story owner and the affected tester must also be found and notified.

Running automated AT may result in two progress change events. First, a complete user story will be affected if one of its associated automated acceptance tests has failed. If an acceptance test fails, the corresponding user story must not be set as 'complete'. Team members affected must also be found and notified.

Second, a user story may be affected if one of its associated automated acceptance tests has passed. This happens if all the functionalities required for the story have been completed and integrated and the other acceptance tests for the same story have already passed. The corresponding user story must be set as 'complete' and team members affected must be found and notified.

Similar to running automated AT, updating a manual acceptance test to 'fail' causes a complete story to become incomplete. In this event, the user story must not be set as 'complete'. Finding and notifying the story owner and the affected tester may be required. Likewise, updating a manual acceptance test to 'pass' may cause the story to become complete The corresponding user story must be set as 'complete' and team members who are affected must be found and notified.

Finally, an acceptance test may not have passed when a team member wants to set its corresponding story to 'complete'. It is required then to prevent setting the story to 'complete' until all its corresponding acceptance tests pass.

VII. Design of a Progress Tracking System

This section discusses the design of a computer-based progress tracking system capable of providing the necessary co-ordination requirements identified in the previous section.

A. System Architecture

There are several technical activities affecting development progress as discussed in section II. These activities are currently carried out by technical systems:

- Source Code Versioning: carried out by version control systems
- Unit Testing (UT) activities: carried out by UT tools
- Acceptance Testing (AT) activities: partially carried out by AT tools
- Continuous Integration (CI) and Releasing activities: carried out by CI and releasing tools.

The problem with these systems is that they work separately from the progress tracking system. Hence, if a progress change event is caused by a technical activity, the progress tracking system cannot identify it.

The progress tracking system proposed here enables the tracking system to keep track of the impact of the technical activities by placing them under control of the tracking system (Figure 1). This can be achieved by:

integrating the versioning functionalities into the progress tracking system and linking the UT tool, AT tool and CI tool with the progress tracking system.

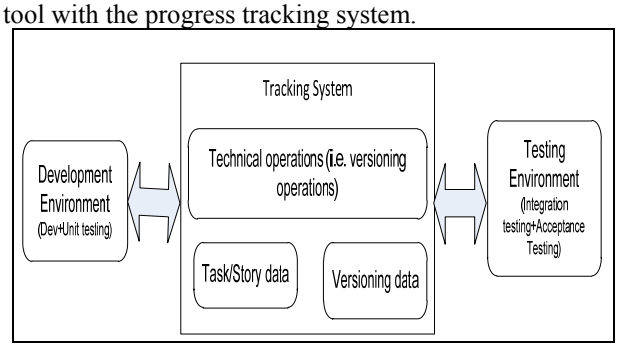

Figure 1. A high level architecture for the progress tracking system.

- Integrating Versioning Functionalities into the Progress System

Current versioning systems provide technical mechanisms to store and control source code artefacts. However, these systems provide no support for identifying and co-ordinating changes affecting development progress.

Because development progress in an agile development is directly based on the maturity of the source code artefacts, tasks/stories should not be tracked separately from the source code artefacts that determine their functionalities. There should be a consistency between the progress data and the actual work performed by developers. This has been seen as a worthy reason to fully integrate versioning functionalities into the progress tracking system.

- Linking the UT, AT and CI Tools with the Tracking System

The progress tracking system has to offer interfaces to the UT tool, AT tool and CI tool, so the tracking system can capture the point where a potential progress change takes place.

B. Version Model

1) Version States and Operations

Version states are used to indicate the level of maturity of different versions of source code artefact. Version state is taken into account when determining progress. Based on the fact that source code artefacts pass several stages before they are released (unit testing, integration, releasing), a four-stage hierarchical promotion model that shows this evolution is proposed which incorporates the following versions:

- Transient Version (TV): the artefact version is not shared with other team members.
- Unit-Tested Version (UTV): the artefact version is unit-tested and available to be shared with other team members.
- Integrated Version (IV): the artefact version is unit-tested and has passed the build.
- Releasable Version (RV): The user stories for which the artefact version provides functionality have passed AT and are ready for releasing.

New operations are required to fulfil the requirements of providing a better description of artefact progress states. Extended versioning operations are described in Table IV.

TABLE IV.
EXTENDED VERSIONING OPERATIONS.

Versioning Operation	Description
Create a new artefact	A new artefact is created as transient version (TV) in a developer's workspace.
Check-out artefact version	A new TV can be created from a version of an existing artefact. The version is created as part of specific task duty.
Check-in artefact version	If TV is stable and unit-tested, it can be promoted to UTV.
Perform integration	If integration is successful, all UTVs included in the integration are promoted to IV.
Release artefact version	If acceptance testing is successful for all affected stories, their associated versions can be released to the customer.
Delete an artefact	An artefact is deleted.

The UML Statechart Diagram in Figure 2 shows how the new versioning operations can change an artefact's state.

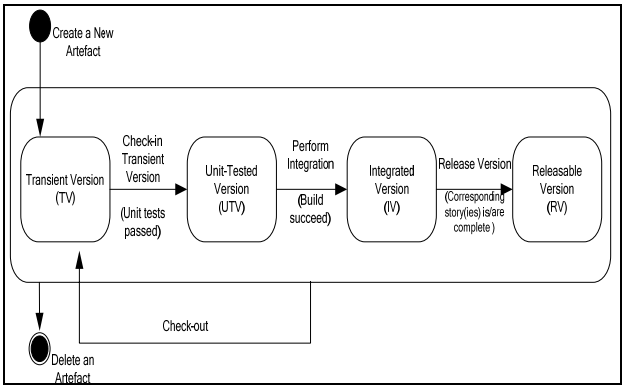

Figure 2. Source code version states.

2) Version Tracking

Managers need to know which tasks are working on each source code artefact. This information can help to recognise which tasks are affected by progress change. In order to obtain this information, every time a developer tries to change an artefact, the task he is working on has to be identified.

A linkage for each source code artefact can be created that describes the tasks that are using versions of the artefact and at which state they are. Table V shows an example of an artefact with different versions for different tasks.

TABLE V.
EACH SOURCE CODE ARTEFACT IS LINKED TO THE TASKS WORKING ON IT.

Version	Task1	Task2	Task3
TV			V3
UTV		V2	
IV	V1		
RV			

The linkage allows the two important versions for each source code artefact to be kept track of. They are the last stable version (last IV), and the last recent

version (last UTV). When a developer asks to read or create a new version, he can choose either one. This provides the awareness to the developer about the status of the version he is using.

The promotion to a higher state affects all the tasks /stories that have the same or lower state. The tasks that have TVs are not usually affected by new updates as the TVs represent unstable copies that are isolated from the other developers. Table VI illustrates when a new version can affect current versions. The main driver of producing the table is to identify clearly the situations in which editing or promoting a source code artefact by one task can have an effect on other tasks' progress.

TABLE VI.
NEW VERSION AFFECTS CURRENT VERSIONS.

	Current TV	Current UTV	Current IV	Current RV
New TV	No	No	No	No
New UTV	No	Yes (developers who use old version should be informed)	No	No
New IV	No	Yes (The task that is linked with the UTV should be updated with the new version)	Yes (developers who use old version should be informed)	No
New RV	No	Yes	Yes	Yes

If a version is promoted to UTV, all the tasks that have UTV versions for the same artefact can be affected. The new modifications for the artefact may influence the work recently completed by other tasks, which are not integrated.

However, if a new UTV is produced, the tasks that have IVs are not affected because the UTV version has not been integrated yet.

Furthermore, it is important to prevent the new build result from influencing complete stories. Let's assume that a story US1 is complete and new versions of the artefact that US1 used are produced by another story and have undergone a build. If the build failed, it should not affect the stories that are already complete. The progress impact should apply only to those stories that made recent changes.

C. User Story Progress Model

Status of the agile projects is expressed through determining the state of the user stories, where each story should produce a block of working software. In order for the story to produce a working software, the individual source code versions contributing to fulfilling the requirements of a story has to be integrated successfully (i.e. reach IV stage) and the set of versions together has to be acceptance-tested in a testing environment to ensure the customer is satisfied with the story. The version model presented in the previous section helps determine the state of the software (source code) and hence can help describe the state of user stories.

The user story's state may change, even after performing acceptance testing (AT). Source code artefact related to a completed story may be versioned and need to be re-integrated and re-tested. To satisfy this, we have identified a new model for the story progress states that takes into account the different progress stages that a user story can assume. User stories may assume one of the following states: 'Not started', 'Active', 'Waiting for integration', 'Waiting for AT', or 'Complete'.

The user story progress model supports providing a more realistic view of the actual state of the software project and also helps reflect the impact of the technical activities on development progress.

Figure 3 is a UML state diagram showing how the technical activities may move a user story from one state to another.

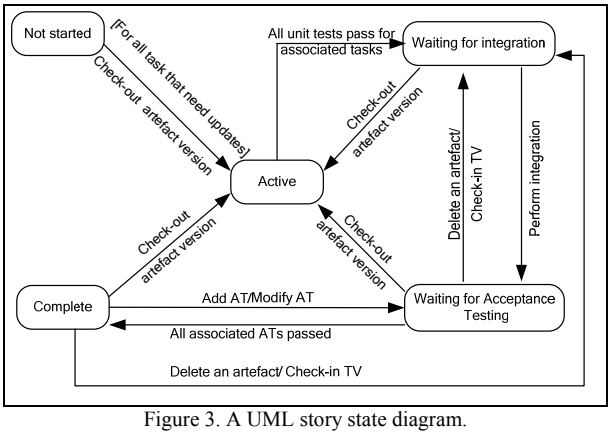

Figure 3. A UML story state diagram.

A user story becomes 'Active' once a developer works on one of its corresponding tasks. After implementing all its functionalities, it moves to the 'Waiting for Integration' state. Once the integration is passed, it moves to the 'Waiting for acceptance testing' state. Finally, it can only become complete if all the associated acceptance tests pass. Team members will have better awareness of the progress state of user stories if they can obtain detailed information about these midpoints.

At the beginning of an iteration, all user stories are in the 'Not started' state. However, the technical activities described in Table I may change stories' states and move them from one state to another.

Checking-out a source code artefact version, as part of working on a user story, causes the story to become 'Active'. The completion of implementing all the tasks corresponding to a user story makes the story move to the 'Waiting for integration' state (this implies that source code artefacts associated with each task have been unit-tested). If integration is passed, the story becomes ready for acceptance testing (moves to the 'Waiting for AT' state). If all acceptance tests associated with the user story have passed, the story then becomes 'Complete'.

The check-out process means that there is still work needed to fulfil requirements of the story, i.e. story is still 'Active'. Editing or deleting a source code artefact that belongs to a 'Waiting for AT' story or a 'Complete' story may require that the story undergoes integration once again. Moreover, adding or modifying an acceptance test for a 'Complete' story moves the story to the 'Waiting for AT' state.

D. Process Model

A set of process models have been developed to illustrate how each technical activity affects development progress. These models provide a visual representation of how the co-ordination activities discussed in section VI can be implemented in a computer-based system. This includes showing explicit support for checking progress constraints, finding and notifying team members affected by progress change, identifying potential sources of progress change, and reflecting progress change in the tracking system.

As an example, the integration process model is provided in Figure 4. It shows the sequence of activities involved in performing integration.

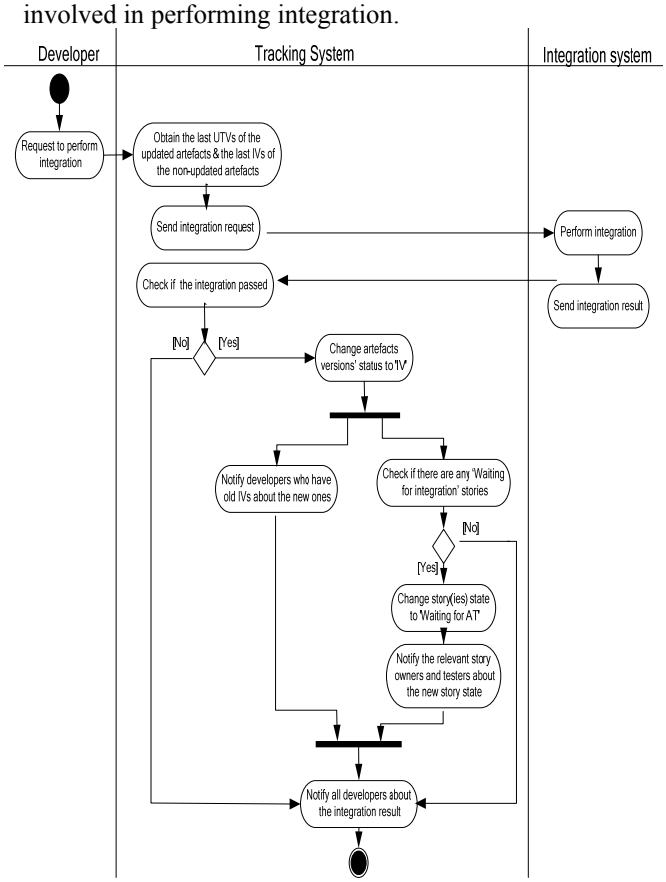

Figure 4. The 'Perform Integration' process model.

VIII. EVALUATION

A scenario-based methodology is used to evaluate the holistic approach. The evaluation process consists of three main parts: **selection of scenarios, analysis of scenarios and validation of scenarios.**

To achieve the first, a systematic method was used in our previous work [6] to identify suitable scenarios. This ensured that the progress change events involved in the scenarios were significant and had reasonable coverage for evaluating the holistic approach by considering the complexity degree of the progress event, frequency of the event occurrence and influence of the progress event on the development progress. In addition, scenarios were selected that included potential progress events from each technical factor. The selection process resulted in choosing three representative scenarios: Check-in Source Code Version, Performing Successful Integration and Running Automated Acceptance Testing.

In the analysis of scenarios, a comparison is made between the classical XP version of each scenario, where the holistic approach is not considered, and the new version after introducing the holistic approach. In addition, a multi-perspective view is achieved by providing a role-oriented analysis of each scenario version.

The validation of scenarios is achieved through developing a prototype system to validate the holistic scenario.

Our previous work [6] has discussed the 'Check-in Source Code Version' scenario only. In this paper we provide analysis of two additional scenarios: 'Performing Successful Integration' and 'Running Automated Acceptance Testing'. Each of them has two versions: the classical XP version and the holistic approach version. The classical XP version of the scenarios are based on the best practices used for performing integration [62] and running automated acceptance test [63].

Before describing the scenario details of each version, a general description of the scenario is given through showing pre-conditions (the state before the scenario starts), trigger (what initiates the scenario) and post-conditions (the state after completing the scenario).

A. Scenario: Performing Successful Integration

Pre-conditions: Ten new versions have been developed since last integration. The functionalities required for user story US1 have been implemented and the story is waiting for its new versions, A5.1 and A6.1, to be integrated. In addition, the functionalities required for user story US2 have been implemented and the story is waiting for its new versions, A8.1 and A9.1, to be integrated. The user stories US3 and US4 are still active (Figure 5).

Trigger: Additionally to the nightly build practice that the team follows, Sally would like to initiate an integration process to ensure that source code does not include any integration problems at the moment.

Post-conditions: The integration is successful and the user stories US1 and US2 become ready for acceptance testing.

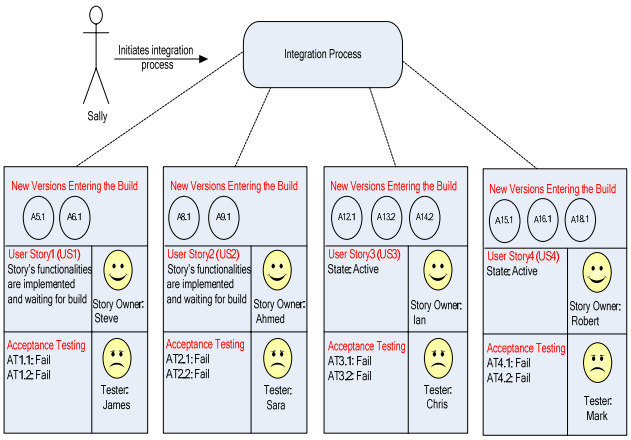

Figure 5. The state before performing the integration.

Classical XP Scenario

1. Sally clicks on 'Perform Integration' in the continuous integration (CI) system.
2. The integration system retrieves the new artefact versions from the versioning system and performs the integration.
3. The integration system returns the result to Sally and sends generic notifications of the integration result to all.
4. Team members need to figure out which story functionalities are completely implemented and integrated in the current build and then need to be acceptance-tested.

The key scenario steps are summarised in Figure 6.

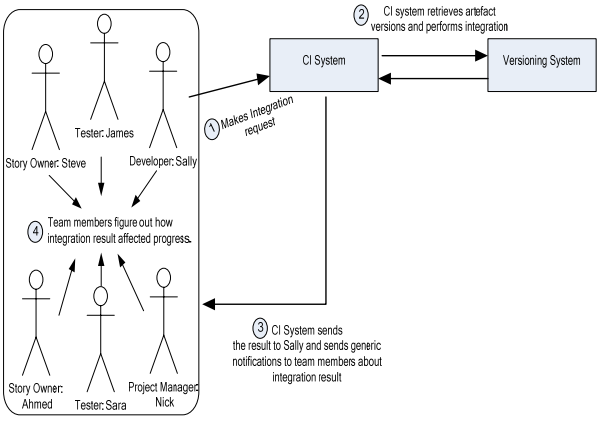

Figure 6. The Classical XP version of the integration scenario.

The Holistic Approach Scenario

1. Sally clicks on 'Perform Integration' in the tracking system.
2. System retrieves the last UTVs of the recently updated artefacts and the last IVs of the non-recently updated artefacts and sends an integration request to the continuous integration (CI) system.
3. System receives 'Successful' result from the CI system and updates the UTV versions to 'IV'.
4. System checks if there are any 'Waiting for Integration' user stories. It moves the stories US1 and US2 to 'Waiting for AT'.
5. Generic notifications are sent to all team members to raise awareness of the integration result. In addition, specialised notifications, clarifying the new state of US1 and US2, are sent to those responsible for US1 and US2, story owners (Steve and Ahmed) and testers (James and Sara) as well as the project manager.

The key scenario steps are summarised in Figure 7.

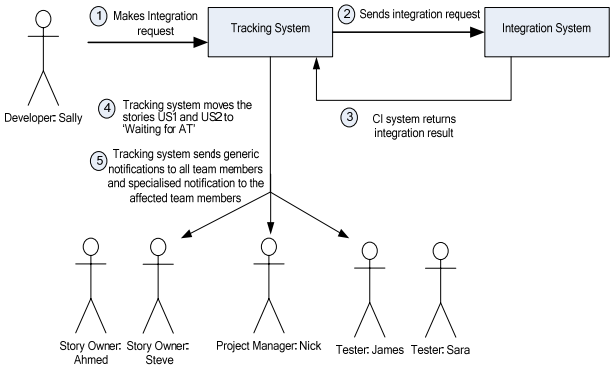

Figure 7. The holistic approach version of the integration scenario.

Scenario Analysis

The 'Perform Successful Integration' scenario requires performing three co-ordination activities:

- **Identifying potential sources of progress change:** An integration 'pass' result should contribute to making progress on the stories that are completely implemented and have associated versions entering the build.
- **Reflecting progress change in the tracking system:** the progress state of the affected stories has to be changed.
- **Finding and notifying team members affected by a potential progress change:** Story owner and tester have to be located and notified that the story is now ready for acceptance testing.

Table VII compares the holistic approach with the classical XP approach, based on the three co-ordination activities needed. The holistic approach provides better awareness of the actual work completed by team members. It automatically identifies the affected stories and hence team members will not need to spend time recognising how the integration result affects their work progress. When integration passes, relevant story owners and testers become aware immediately that their stories have become ready for acceptance testing. This provides the opportunity for testers to carry out the set of acceptance tests for each story as early as possible in the iteration.

TABLE VII.
ANALYSIS OF THE CO-ORDINATION SUPPORT IN THE INTEGRATION SCENARIO.

Co-ordination Activity	The Classical XP Approach	The Holistic Approach
Identifying potential sources of progress change	• Team members need to figure out how the integration result affects progress. • It can be difficult for members working in the distributed project to realise which stories have been affected.	• Potentially affected stories are automatically identified once the integration is performed. • Provides better visibility of the actual progress. Immediately identifies the potential change in progress resulting from the integration process.
Reflecting	• Team members	• The integration effect

Co-ordination Activity	The Classical XP Approach	The Holistic Approach
progress change in the tracking system	usually share the new progress state informally, not in the tracking system.	is automatically reflected in the tracking system. • It increases the entire team's awareness about the project state.
Finding and notifying team members affected by potential progress change	• It is done in an ad-hoc manner. • The affected story owners and testers may be in different sites. This may make it difficult to identify who should be notified. • A delay in making the acceptance testing may take place because affected team members do not know that the story is ready for acceptance testing.	• The affected story owners and testers are notified automatically by the system once the integration process is completed. • The automatic notification raises awareness that the stories are available for acceptance testing, thus increasing the opportunity to run the acceptance tests earlier in the development cycle.

This scenario involves participation from a developer, story owners, testers and project manager. A role-oriented analysis is provided in Table VIII.

TABLE VIII.
A ROLE-BASED ANALYSIS OF THE INTEGRATION SCENARIO.

Role of the Team Member	The Classical XP Approach	The Holistic Approach
Developer	• performs integration	• performs integration
Tester	• needs to figure out which story functionalities are completely programmed and integrated in the current build.	• is informed immediately about which stories have become ready for AT.
Story Owner	• may not recognise how the integration result affects his story progress.	• is informed immediately about change in his story progress.
Project Manager	• will know the integration result but will not know how the result affects the development progress.	• The approach allows him to realise the effect of the integration on the development progress.

B. Running Automated Acceptance Testing

Pre-conditions: The acceptance tests AT1.1, AT2.1 and AT3.1 are passed. These tests belong to the completed stories US1, US2 and US3 respectively (Figure 8).

Trigger: As part of a regression testing, the test leader, Sam, runs automated acceptance testing with the three acceptance tests (AT1.1, AT2.1, AT3.1).

Post-conditions: The acceptance tests AT1.1 and AT2.1 failed due to recent modifications to shared code belonging to US1 and US2.

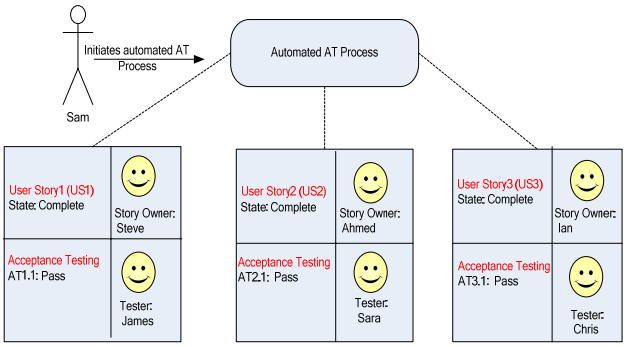

Figure 8. The state before the testing process.

Classical XP Scenario

1. Sam initiates automated acceptance testing using the AT tool.
2. The AT tool performs the tests and returns the results to Sam.
3. Sam finds and then notifies the affected team members.

The key scenario steps are summarised in the following diagram (Figure 9):

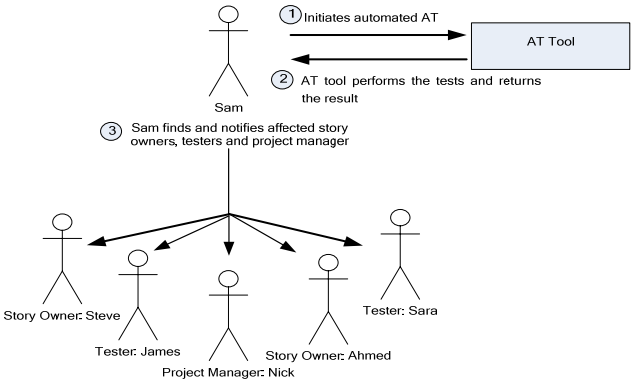

Figure 9. The Classical XP version of the automated AT scenario.

The Holistic Approach Scenario

1. Sam initiates automated acceptance testing using the tracking system.
2. The tracking system sends request to the AT tool and then receives the test result.
3. Because the result shows that the acceptance tests AT1 and AT2 failed, the tracking system changes the state of the user stories US1 and US2 to 'Waiting for AT'.
4. The tracking system provides the result to Sam and automatically sends notifications to the affected team members (story owners: Steve and Ahmed, testers: James and Sara, and the project manager, Nick).

The key scenario steps are summarised in Figure 10:

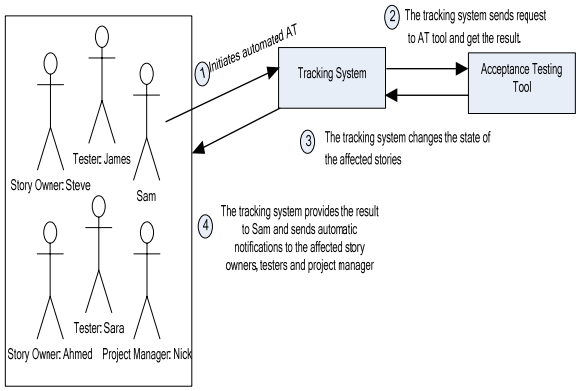

Figure 10. The holistic approach version of the automated AT scenario.

Scenario Analysis:

The 'Running Automated AT' scenario requires performing three of the co-ordination activities:

- **Identifying potential sources of progress change:** If running automated acceptance testing has led to failed acceptance tests and if these acceptance tests belong to completed stories, these stories become incomplete.
- **Reflecting progress change in the tracking system:** the progress state of the affected stories has to be changed.
- **Finding and notifying team members affected by a potential progress change:** Story owners and testers have to be located and notified that the affected stories have failed acceptance tests.

Table IX compares the holistic approach with the classical XP approach, based on the three co-ordination activities needed. This scenario shows that the holistic approach allows the tracking system to identify the influence of failed acceptance tests on development progress. If an acceptance test fails, the system automatically changes the state of the corresponding user story and notifies the story users.

By replacing the manual method, the holistic approach can provide better awareness to team members about the real progress of development. The impact of failed acceptance tests is formally reflected in the tracking system. In addition, the affected story owners and testers, as well as project manager, are automatically found and notified.

TABLE IX.
ANALYSIS OF THE CO-ORDINATION SUPPORT IN THE AUTOMATED AT SCENARIO.

Co-ordination Activity	The Classical XP Approach	The Holistic Approach
Identifying potential sources of progress change	● The person who made the automated AT may not know which stories have been affected. This is likely if the failed acceptance tests belong to stories created at different sites.	● Identified automatically by the tracking system.
Reflecting progress change in the tracking system	● Progress change is not reflected in the tracking system.	● The effect of the automated AT is automatically reflected in the tracking system. ● It increases the entire team's awareness of the project state.
Finding and notifying team members affected by potential progress change	● It is done in an ad-hoc manner. ● The affected story owners and testers may be at different sites. This may make it difficult to identify who should be notified. ● A delay in resolving the defects may take place because affected team members may not be notified about the failed tests.	● The affected story owners and testers are notified automatically by the tracking system once the AT process is completed. ● The automatic notification helps resolving the defects early in the development cycle.

This scenario involves participation of the test leader, testers, story owners and project manager. An evaluation of the benefits that each of them can achieve is assessed in the role-oriented analysis presented in Table X.

TABLE X.
A ROLE-BASED ANALYSIS OF THE AUTOMATED AT SCENARIO.

Role of the Team Member	The Classical XP Approach	The Holistic Approach
Test Leader	● performs the automated AT. ● has to find and notify the affected team members.	● performs the automated AT.
Tester	● will not be able to investigate the source of the problem until he is notified by the test leader.	● is informed immediately about failed test.
Story Owner	● will not be able to investigate the source of the problem until he is notified by the test leader.	● is informed immediately about change in his story progress.
Project Manager	● will not know the actual progress until he is notified by the test leader.	● will be notified automatically about the progress change.

C. Validation of the Holistic Approach

A research prototype system has been developed to demonstrate the feasibility of the proposed holistic approach to developing a progress tracking system. It validates the holistic approach version of the scenarios provided in the previous section and ensures that a computer-based system is capable of demonstrating them.

The NetBeans IDE [64] has been used to develop the application, while MySQL [65] is used as the backend database. Both are popular tools used for creating computer applications.

System Database. The proposed progress tracking system requires storing and accessing different types of data entity. These data entities have dependencies among them. There is a large number of dependencies among tasks, stories, releases, unit tests, acceptance tests and integration tests. These dependencies in the tracking

system need to be carefully represented in a database. The database in Figure 12 shows the minimum data requirements needed to design a simple progress tracking system using the holistic approach.

Implementation of 'Performing Successful Integration' Scenario. The implementation for each scenario consists of a sequence of screens, where each screen represents one step of the scenario described earlier. Each screen displays the output that results from moving from one step to another. A timer is used to move the screens forward.

Due to the page limit of this article, we explain in detail the various steps involved in the holistic approach version of 'Running Automated Acceptance Testing' scenario only.

The scenario starts with the following initial data set:

- The acceptance tests AT1.1, AT2.1 and AT3.1 have passed.
- These tests belong to the completed stories US1, US2 and US3, respectively.

Implementation of the four steps involved in the holistic version of the acceptance testing scenario is discussed below.

1- Sam initiates automated acceptance testing using the tracking system.

The current states of AT1.1, AT2.1 and AT3.1 are displayed in Figure 11.

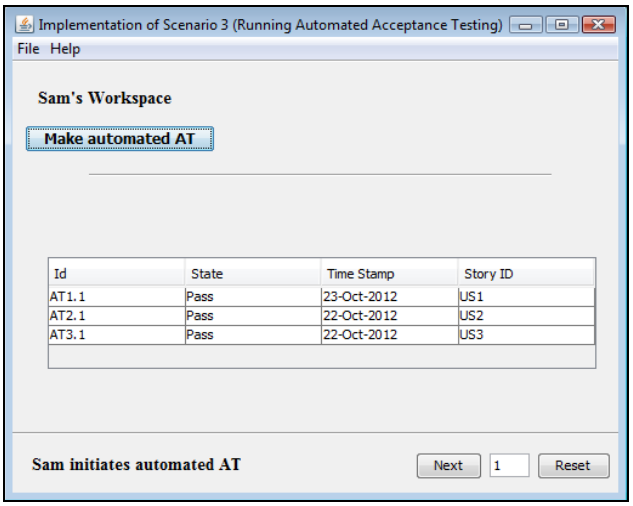

Figure 11. Implementation's step 1.

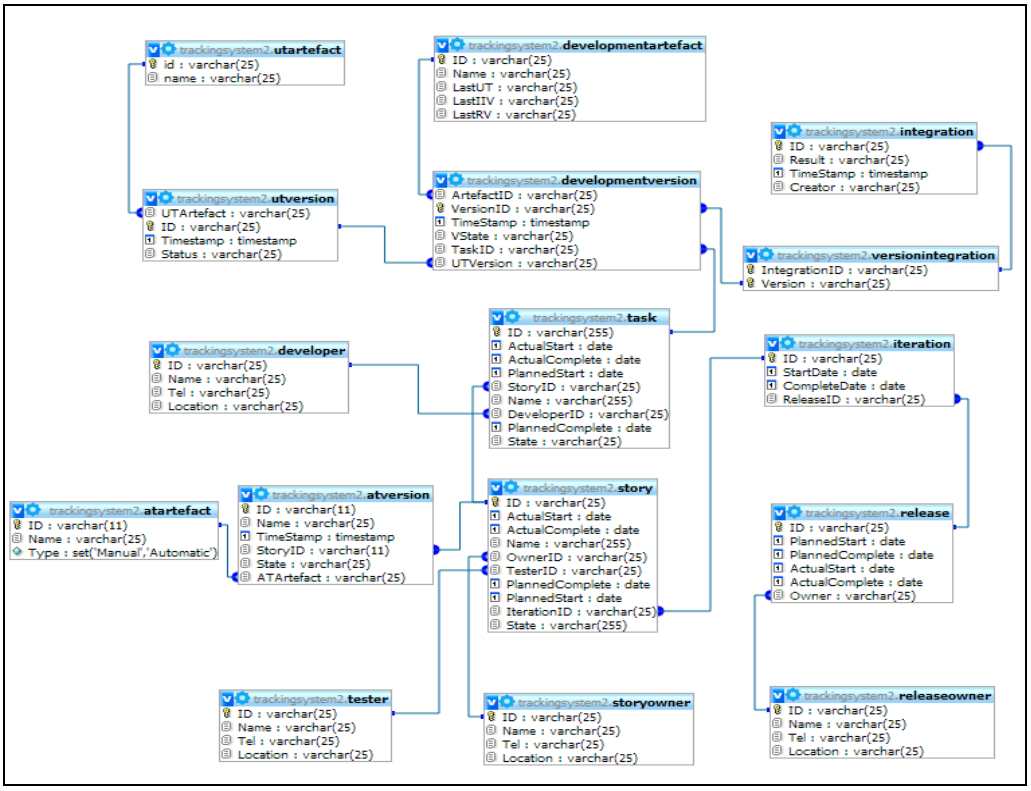

Figure 12. System database.

2- The tracking system sends a request to the AT tool and then receives the test result.

The test results show that acceptance tests AT1.1 and AT2.1 have failed. Hence, the state of each test needs to be updated in the database. This can be achieved through the following SQL query:

```
"Update ATversion
Set state='Fail'
Where id='"+ s[i]+ "'"
```

The ‘s[i]’ symbol shown in italics in the previous query refers to an array in the source code that stores the id value of the failed acceptance tests.

3-4. The tracking system changes the state of user stories US1 and US2 to ‘Waiting for AT’. The tracking system provides the result to Sam and automatically sends notifications to the affected team members.

The stories that need to be moved to 'Waiting for Acceptance Testing' are retrieved through the following query:

```
"Select storyid
From atversion
Where id='"+failedtests[i]+"'"
```

For each of the affected stories, the story owners and testers are retrieved through the following two queries:

```
"Select so.name
From story s, storyowner so
Where s.ownerid=so.id and
        s.id='"+story+"'"
```

```
"Select t.name
From story s, tester t
Where s.testerid=t.id and
        s.id='"+story+"'"
```

The ‘story’ symbol shown in italics in the previous two queries refers to a variable in the source code that represents an affected story's id.

The affected user stories, story owners and testers are shown in Figure 13.

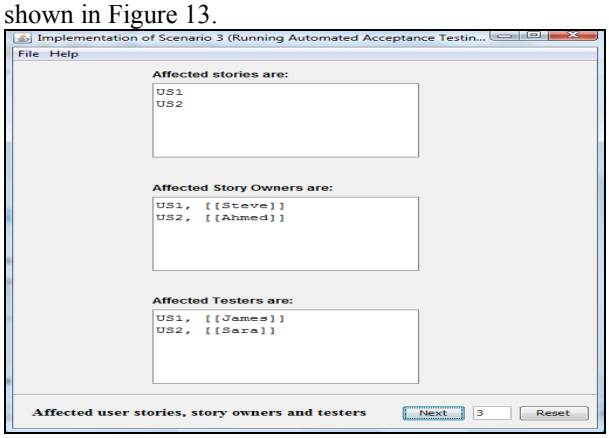

Figure 13. Implementation's step 3.

A notification message is sent automatically to each of the affected team members. Example of such message is shown in Figure 14. It shows a notification message sent to Sara, the tester responsible for US2.

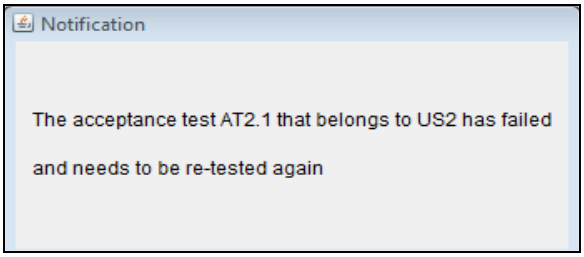

Figure 14. Implementation's step 4.

IX. Conclusions

The holistic approach helps distributed agile teams identifying potential sources of progress change and helps co-ordinate it with other team members. This overcomes the limitation of the informal methods, where team members completely rely on their understanding of how carrying out technical activities may change the progress. The holistic approach also overcomes the limitations of the formal methods. These methods use computer systems merely for registering progress information but do not help team members in identifying the point where a progress change event occurs and obviously do not provide the co-ordination support necessary to deal with such events.

Providing an effective approach that incorporates the impact of the technical activities on development progress improves the awareness of distributed agile teams regarding the actual progress. Team members will no longer need to spend time determining how their change will impact the work of the other team members so that they can notify the affected members of the Change. They will be provided with a system that helps them achieve this as they carry out their technical activities. In addition, they will not rely on static information about progress registered in a progress tracking system, but will be updated continuously with relevant information about progress changes occurring to their work.

References

[1] P. Dourish and V. Bellotti, “Awareness and coordination in shared workspaces,” Proceedings of the 1992 ACM conference on Computersupported cooperative work CSCW 92, vol. 3, no. November, pp. 107–114, 1992.

[2] J. Espinosa, S. Slaughter, R. Kraut, and J. Herbsleb, “Team Knowledge and Coordination in Geographically Distributed Software Development,” Journal of Management Information Systems, vol. 24, no. 1, pp. 135–169, Jul. 2007.

[3] K. Beck, Extreme Programming Explained: Embrace Change. Addison-Wesley Professional, 2004, p. 224.

[4] P. Xu, “Coordination In Large Agile Projects,” Review of Business, vol. 13, no. 4, pp. 29–44, 2009.

[5] J. Patton, “Secrets to Automated Acceptance Tests.” [Online]. Available: www.stickyminds.com/s.asp?F=S13798_COL_2.

[6] S. Alyahya, W. K. Ivins, and W. A. Gray, “A Holistic Approach to Developing a Progress Tracking System for Distributed Agile Teams,” in the 11th IEEE/ACIS International Conference on Computer and Information Science, 2012.

[7] I. Sommerville, Software engineering (9th edition). Addison Wesley, 2010.
[8] K. Beck, M. Beedle, A. V. Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries, J. Kern, B. Marick, R. C. Martin, S. Mellor, K. Schwaber, J. Sutherland, and D. Thomas, "Manifesto for Agile Software Development," The Agile Alliance, 2001. [Online]. Available: http://agilemanifesto.org/.
[9] M. C. Paulk, "Extreme programming from a CMM perspective," IEEE Software, vol. 18, no. 6, 2001.
[10] "Summary of the subworkshop on extreme programming," Nordic Journal of Computing, vol. 9, no. 3, 2002.
[11] P. Abrahamsson, O. Salo, J. Ronkainen, and J. Warsta, "Agile software development methods," Vtt Publications, vol. 478, no. 3, pp. 167–168, 2002.
[12] VersionOne, "5th Annual State of Agile Development Survey Final summary report," 2010.
[13] "Agile Practices and Principles Survey Results," 2008. [Online]. Available: http://www.ambysoft.com/surveys/practicesPrinciples2008.html.
[14] M. Paasivaara, S. Durasiewicz, and C. Lassenius, "Using Scrum in Distributed Agile Development: A Multiple Case Study," in Fourth IEEE International Conference on Global Software Engineering, 2009, pp. 195–204.
[15] E. Therrien, "Overcoming the Challenges of Building a Distributed Agile Organization," in Agile Conference, 2008, pp. 368–372.
[16] W. Williams and M. Stout, "Colossal, Scattered, and Chaotic (Planning with a Large Distributed Team)," in Agile Conference, 2008, pp. 356–361.
[17] M. Korkala, P. Abrahamsson, and P. Kyllonen, "A Case Study on the Impact of Customer Communication on Defects in Agile Software Development.," in Agile 2006, 2006, pp. 76–88.
[18] M. Kajko-Mattsson, G. Azizyan, and M. K. Magarian, "Classes of Distributed Agile Development Problems," in Agile Conference, 2010, pp. 51–58.
[19] M. Vax and S. Michaud, "Distributed Agile: Growing a Practice Together," in Agile 2008 Conference, 2008, pp. 310–314.
[20] M. Cottmeyer, "The Good and Bad of Agile Offshore Development," Agile 2008 Conference, pp. 362–367, 2008.
[21] S. H. Rayhan and N. Haque, "Incremental Adoption of Scrum for Successful Delivery of an IT Project in a Remote Setup," Agile 2008 Conference, pp. 351–355, 2008.
[22] K. Beck and M. Fowler, Planning Extreme Programming. Addison-Wesley, 2001, p. xvii, 139 p.
[23] A. Cockburn, Crystal-Clear a Human-Powered Methodology for Small Teams. Pearson Education, Inc., 2005, p. 312 ST – Crystal–Clear a Human–Powered Methodolog.
[24] A. Danait, "Agile offshore techniques - a case study," Agile Development Conference ADC05, 2005.
[25] M. Dubakov and P. Stevens, "Agile Tools: The Good, the Bad and the Ugly." Report, TargetProcess, Inc, 2008.
[26] T. Chau and F. Maurer, "A case study of wiki-based experience repository at a medium-sized software company," KCAP 05 Proceedings of the 3rd international conference on Knowledge capture, p. 185, 2005.
[27] Mircrosoft, "MS Project." [Online]. Available: http://www.microsoft.com/project/en-us/project-management.aspx.
[28] G. Azizyan, M. K. Magarian, and M. Kajko-Matsson, "Survey of Agile Tool Usage and Needs," 2011 AGILE Conference, pp. 29–38, 2011.
[29] V. Heikkilä, "Tool Support for Development Management in Agile Methods," Helsinki University of Technology, 2008.
[30] "Rally." [Online]. Available: http://www.rallydev.com/.
[31] "Mingle." [Online]. Available: http://www.thoughtworks-studios.com/mingle-agile-project-management.
[32] "VersionOne." [Online]. Available: www.versionone.com.
[33] "TargetProcess." [Online]. Available: www.targetprocess.com.
[34] "On Time." [Online]. Available: www.axosoft.com/.
[35] K. Schwaber, "Agile project management with Scrum," Redmond Microsoft Press, p. 163, 2004.
[36] "eXPlainPMT." [Online]. Available: https://github.com/explainpmt/explainpmt.
[37] D. J. Anderson, "Using Cumulative Flow Diagrams." The Coad Letter – Agile Management, Report, 2004.
[38] "ScrumWorks." [Online]. Available: http://www.collab.net/products/scrumworks/.
[39] "ExtremePlanner." [Online]. Available: www.extremeplanner.com.
[40] "XPlanner." [Online]. Available: http://xplanner.codehaus.org/.
[41] "Pivotal Tracker." [Online]. Available: www.pivotaltracker.com.
[42] "Scrum VSTS." [Online]. Available: www.scrumforteamsystem.co.uk.
[43] "Agilefant." [Online]. Available: www.agilefant.org.
[44] "IceScrum." [Online]. Available: www.icescrum.org.
[45] "Planbox." [Online]. Available: www.planbox.com.
[46] "XP StoryStudio." [Online]. Available: www.xpstorystudio.com/.
[47] "XPWeb." [Online]. Available: xpweb.sourceforge.net/.
[48] "AgileWrap." [Online]. Available: www.agilewrap.com/.
[49] "ScrumDesk." [Online]. Available: www.scrumdesk.com/.
[50] "SpiraTeam." [Online]. Available: www.inflectra.com/spirateam/.
[51] "Leankit." [Online]. Available: leankitkanban.com/.
[52] "DevSuite." [Online]. Available: www.techexcel.com/devsuite/.
[53] "TinyPM." [Online]. Available: www.tinypm.com/.
[54] "Planigle." [Online]. Available: planigle.com/.
[55] "Acunote." [Online]. Available: www.acunote.com/.
[56] "AgileZen." [Online]. Available: www.agilezen.com/.
[57] "ScrumPad." [Online]. Available: www.code71.com/.
[58] "AgileBuddy." [Online]. Available: www.agilebuddy.com/.
[59] "Daily Scrum." [Online]. Available: http://daily-scrum.com/.
[60] "Express." [Online]. Available: agileexpress.sourceforge.net/.
[61] "Agile Tracking." [Online]. Available: https://sites.google.com/site/agiletrackingtool/home.
[62] P. Duvall, S. Matyas, and A. Glover, Continuous integration: improving software quality and reducing risk. Addison-Wesley Professional, 2007, p. 336.
[63] L. Crispin and T. House, Testing Extreme Programming. Addison-Wesley Professional, 2002.
[64] NetBeans, "Netbeans IDE." [Online]. Available: http://www.netbeans.org.
[65] "MySQL." [Online]. Available: http://www.mysql.com/.