SIEMENS

Ingenuity for life

Using Polarion ALM in Scrum

Agile software development processes have proven their value, and even traditional development organizations are embracing their benefits. Siemens PLM Software adopted the Scrum framework for development of its Polarion application lifecycle management (ALM) solution. This paper describes the major factors that influenced the adoption of Scrum, and examines how Polarion ALM™ supports Scrum processes.

A white paper issued by: Siemens PLM Software

www.siemens.com/polarion

Contents

Why Scrum?

Agile processes have proven their value, and even traditional development organizations are embracing their benefits, if only in a hybrid way. Siemens PLM Software adopted the Scrum framework for development of its Polarion application lifecycle management (ALM) solution for several reasons, some specific to size, area of business, customers and others quite general. This paper will focus on the major factors that influenced the adoption of Scrum. Perhaps the most important reason was transparency. Before Scrum, customers would request something, product management then defined requirements, the development team committed to fulfilling them, and yet the end results were not always as expected. Delivery sometimes slipped, and if a release was rescheduled, the risk was unclear and no one could tell whether the new date was realistic. Also, the development team needed to be able to respond quickly to changes in market conditions and business strategy.

Scrum helps developers understand what they need to do to build quality software and enables them to make decisions faster. Switching from defined and predictive development to an empirical iterative incremental model is exactly what Scrum is about.

Scrum's core values
Some of the important values of Scrum include:

- Empiricism – facilitates management, development and deployment of complex products.
- Inspections and adaptations – allowing people to check and reach goals.
- Full transparency – people know the exact state of the product.
- Iterative development – generates visible increments of functionality. Progress is measured not by time or money spent, but by concrete results.
- Self-organization – people want to do their best, and will consistently do so when there is room for them to work in whatever way is most efficient for them, rather than according to some dictum. Team motivation is raised, enhancing productivity and motivation.
- Delivery – many great projects with very capable teams have failed to deliver anything. Scrum helps teams minimize this risk every step along the way. If a project is going to fail, it's better to know as soon as possible in order to kill it earlier and cut the losses.

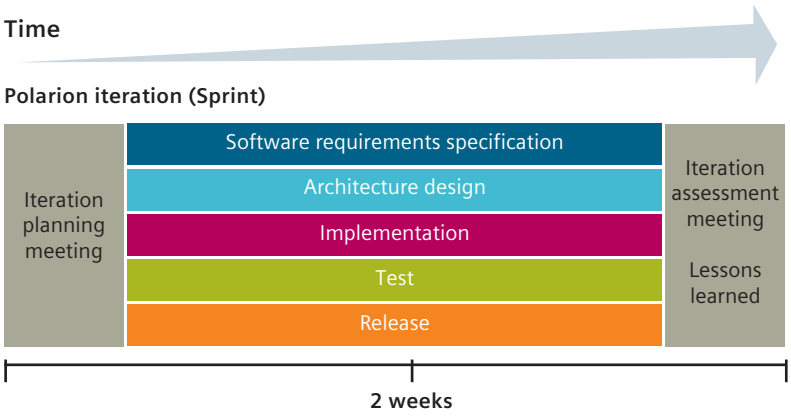

Figure 1

Iterative incremental development

Unlike the old "waterfall" approach, Scrum recommends a highly adaptive way of development with short iterations producing fully tangible results. Major benefits realized from Scrum development include:

- Shorter time to release to market
- Transparency to management and customers
- Faster reaction to market needs
- Higher customer confidence in development abilities
- Simpler synchronization of distributed teams
- Easier releases – smaller stabilization sprints, fewer things to test
- Faster feedback from the field
- Flexibility in prioritization, risk reduction

Also, some activities can be done in parallel. Specification of one feature for the next iteration may happen in parallel with implementation of another feature already specified in a previous iteration. For Polarion development, Siemens PLM Software has very short iterations of two weeks, with an iteration planning meeting at the beginning, and an iteration assessment meeting at the end of each iteration. This has proven optimal for several teams of 3 to 10 team members.

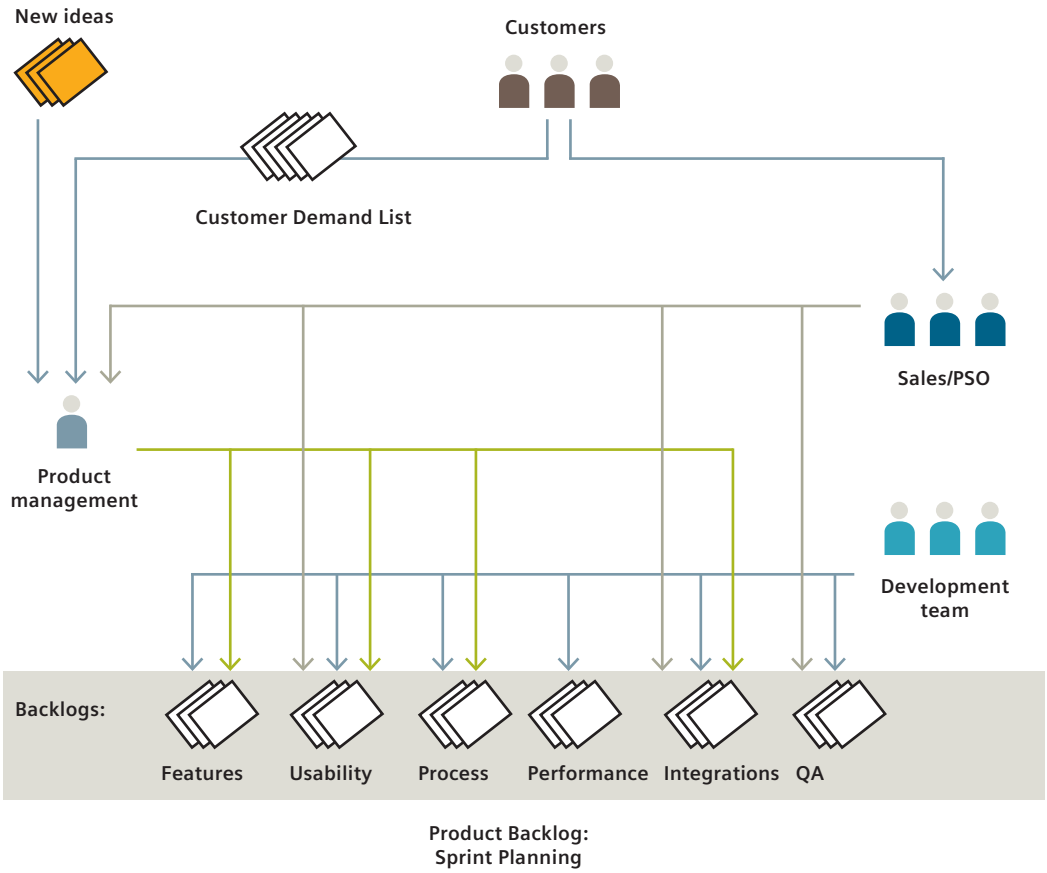

Figure 2: Inputs for product backlog

Polarion ALM in the Scrum process

The remainder of this paper assumes familiarity with the basic functionality of Polarion ALM, including its terminology, and at least some experience with the administration interface.

Work item types

Polarion ALM is configured with four work item types to support the Scrum process:

- User story: defines what functionality should be added, updated or removed. It is formulated in business language, articulates business value and groups all relevant activities together.
- Improvement: specifies positive change(s) that will appear in a future release – code improvements, documentation tasks, etc.
- Defect: errors, flaws, failures, faults
- Task: any activity consuming time and human resources. Results don't appear directly in products – write a test case, install a demo server, brainstorm discussion about a feature, etc.

Change requests or requirements are not included – those are covered by user stories.

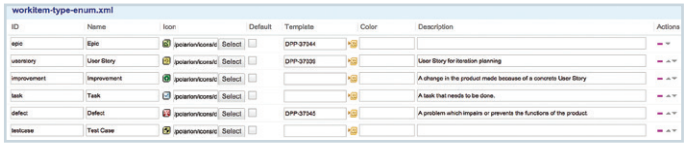

Figure 3: Work item type configuration in Polarion ALM

User story attributes

Attributes are reflected in custom fields defined for each work item type. For user stories we track:

- Author of request (whom to ask for clarifications)
- Backlog (list of all requests, sorted by priority)
- Relationships between user stories (People may require similar or related things. We need to see those relationships to simplify prioritization and grouping in the product backlog.)
- Product edition(s) that will include a feature
- Doc not required – flag if a user story does not require documentation
- Who requested the functionality (customer, prospect, other)
- Responsible developer – this may seem a contradiction to a team-oriented approach but there is a reason for it. We found it useful to have a single person responsible for each user story who checks all the activities around it, and who also leads the demos of the feature when the product is ready.
- State – the most important states are: "open" (new, to do), "in progress" (there is active work on it), "implemented" (implementation activities are finished) and "done."
- Initial estimate – this is typically empirical data, which the team agrees on. The time spent and remaining estimates are calculated automatically using Polarion ALM (via inherited fields) from linked child work items (improvements, defects, tasks).
- There are more attributes specific to our development cycles, but these are the main ones.

Improvement attributes

As with any implementation-related work item, an improvement has references to the build in which it was implemented (so testers know which build to review), in which build it was reviewed by QA, the branch it was committed to, the assignee, time estimates, etc.

Improvement prioritization is typically done in the corresponding user story. All improvements planned to a sprint should be linked to a user story.

Defect attributes

Attributes are similar to those of improvements. Defects can be taken in a sprint without linking to a user story, and they may be prioritized separately. The most important attributes include:

- Build (or product version) where the problem was discovered.[1]
- Severity – the impact on customers or internal users.
- Customer – if reported by a customer, it needs higher priority. Also, the customer may need a patch, so we have to track who should be provided with one.[2]
- Build in which the problem was resolved, branches, assignee, estimates and time spent, etc.
- Known issue – defect is not resolved and should be mentioned in "known issues" list for the release.

Task attributes

This type of work item doesn't have direct connection to customer or builds, therefore it doesn't have any specific attributes. This item also must be linked to a user story to be selected for a sprint.

Linking of work items

Perhaps less important than work item types, Polarion's linking capabilities help us in creating work breakdown structure, and we benefit from the planning features, which take into consideration various types of links.

The most important link types are:

- Implements: the relationship of improvements, defects and tasks to the user story. Until linked child items are resolved, the user story is not considered done.
- Depends on: the linked item must be processed first.
- Relates to: flags some relationship between work items; provides a hint for developers to review if there is anything relevant or important in the linked item.
- Parent: links work items of the same type. Used for decomposition of complex user stories.
- Follows: some work items may be resolved in terms of the request, but it turns out they need further work – usability improvement, or a defect resulting from a fix of another.

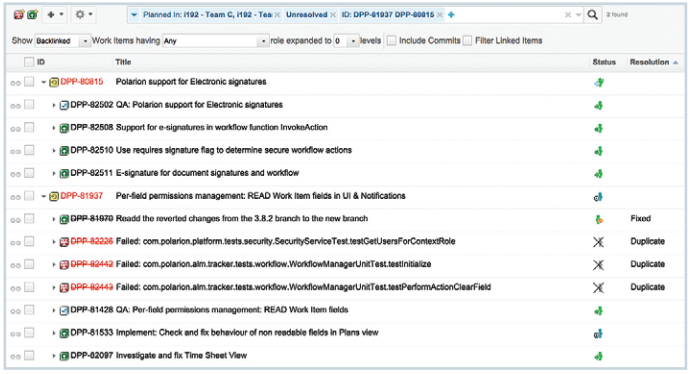

Figure 4: Example of work breakdown structure

Product backlog

Typically the product owner writes up items for the product backlog in a Word or Excel® document and then simply reshuffles them according to priority. This approach could easily cause all kinds of problems except for the fact that Polarion ALM enables efficient and coordinated management of such artifacts.

Composing user stories

Across all owners and stakeholders, we use three ways of composing user stories:

- Through the Polarion web user interface ("create work item")
- Through email sent to the Polarion Mailet extension (a server extension that can receive email messages and turn them into work items)
- Through LiveDocs, Polarion's exclusive office document synchronization feature

Regardless of the authoring method, created work items appear in the tracker and it is relatively easy for all stakeholders to find them using Polarion's query builder (such a query could be "type:userstory AND backlog:usability"). We often embed queries into wiki pages so stakeholders don't have to formulate queries. The one shown in Figure 6 below collects all backlogs and displays the top items.

Prioritizing user stories

Here our process differs from typical Scrum. We have several relatively independent stakeholders, all committed to a common goal, but still in pursuit of their own targets. Therefore, each backlog is prioritized separately by the backlog owner, who defines the threshold of his or her items to flag those which must appear in the product backlog and, ideally, should be discussed by the team.

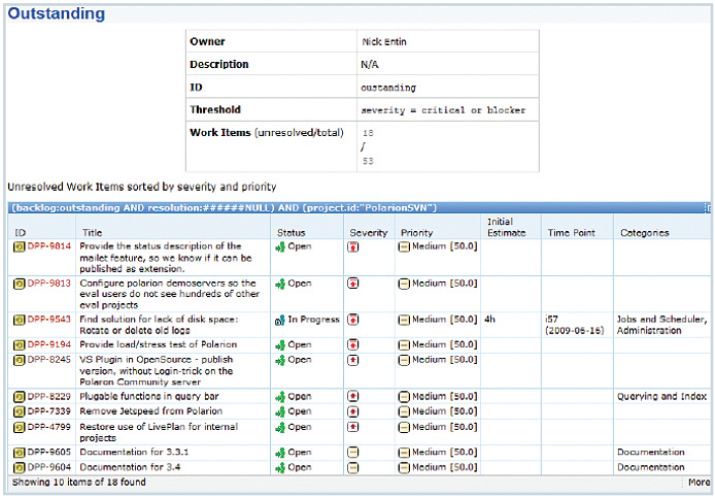

Figure 5: A stakeholder backlog

Extracting from stakeholder backlogs to product backlog

Our next step is to collect all the required items for the product backlog. Using Polarion ALM, it is quite easy to create a wiki page that collects all the "top" (highest priority) items from various backlogs – we simply embed an instance of the {workitems} macro, supplying the correct query to fetch items from the backlogs maintained in the tracker.

Next, the product owner prioritizes the list. We have defined a custom integer field "Product Backlog Priority" (PBP) which the product owner uses to sort the items accordingly.[3]

A highly useful Polarion feature lets you click "more" in a backlog table embedded in a wiki page, which opens the work items table in the tracker (a prime example of Polarion's integrated approach to tools).

The PBP attribute also helps to track down whether there were some changes in a particular backlog that are not yet reflected in the common one. For example, a query that retrieves all the "important" user stories that don't have the PBP field set (Figure 8).

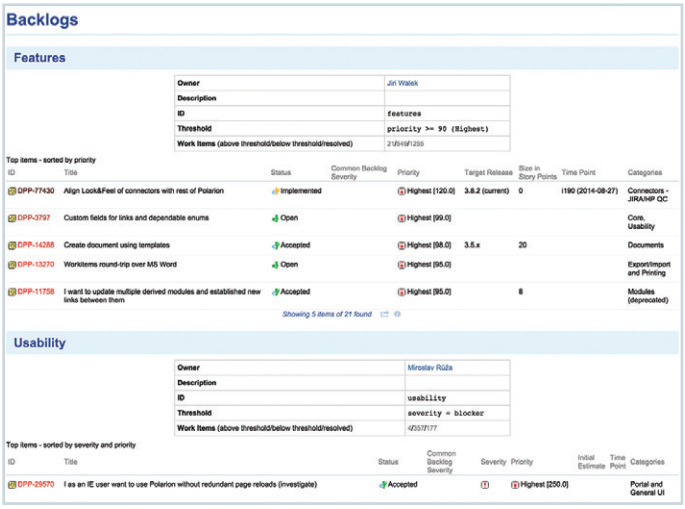

Figure 6: Backlogs wiki page with embedded queries displaying top items different backlogs

Additional tips from our development process

We actively use Polarion's auto-assignment feature when creating new work items. This enables immediate assignment to a senior developer, who will potentially lead the implementation. The developer receives an email notification and sees the new item assigned to him. This encourages early review of posted user stories, provides read-filtered input for the planning meetings, etc.

To simplify prioritization the "weight" or "initial estimate" of a user story is important, and automatic assignment helps to get initial review and communication going even before the planning meeting.

Also, we have configured the user story work item type to aggregate the values of remaining estimate fields from any child items.

So by brainstorming and reaching agreement at the planning meeting, we identify an initial estimate value for each user story. Later, however, when it is decomposed into improvements, tasks and defects, we can often spot variations – that particular work actually may take less time, or some additional task may be added that had not been anticipated, and was discovered after implementation began. This data is extremely helpful for re-planning of any user story that was not finished to the next sprint.

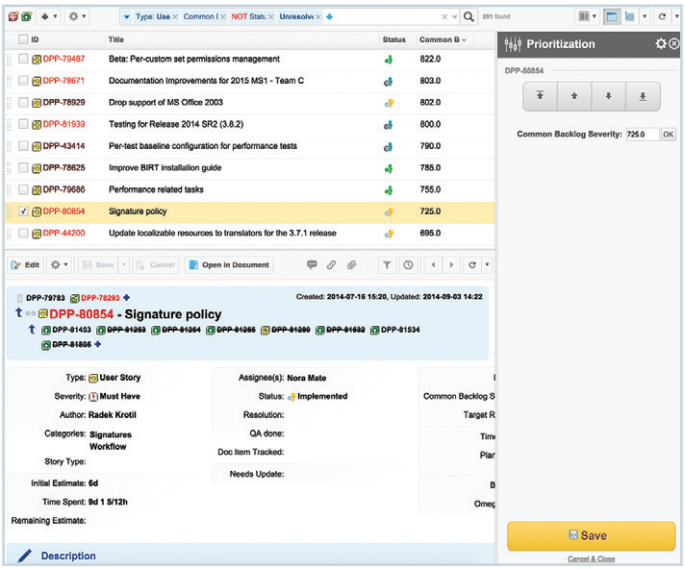

Figure 7: Tracker items comprising the product backlog displayed in a wiki page

Figure 8: Wiki page section with query for potentially missing backlog entries

Tips

- Use auto-assignment for new work items
- Configure the user story work item type to aggregate the values of remaining estimate fields from any child items

The sprint: meetings

Meetings are possibly the most important assets of Scrum. Meetings are when the team commits to the product owner on the amount of work (features) they will address over each sprint. They discuss the progress in daily Scrums and, finally, access results at the last meeting. This section and the next describe how we manage those meetings with the Polarion development team.

The planning meeting

The goal of the planning meeting is to ensure that the team fully understands the product backlog items, to commit the team to implementing agreed-on items in the upcoming sprint, and to ensure proper distribution of work among team members. During the planning meeting, dependencies between teams are also identified to allow as much parallel work by the teams as possible, keeping the same focus for the iteration.

The planning entity for the sprint is the user story. Each one has a customer (the person who formulated the software requirement) and an owner – typically a senior developer, who then follows the user story through the full lifecycle.

Typically the meeting is split into two parts. The first part involves the product owner and possibly other stakeholders, to ensure common understanding of things to be done and to commit the team to specific work items.

The second part is a purely internal meeting, where the team decides who will implement what and splits the user stories into concrete task and improvement work items, and validates capacity of the team using the Polarion ALM's LivePlan feature.[4] Normally the user stories in the product backlog have been inspected and time-estimated by developers in advance. Team members come prepared with questions, and perhaps concerns about conflict with some agreements or principles, or inconsistencies.

Out of the planning meeting come the user stories selected for the sprint, which becomes a time point assignment in Polarion ALM. Results of the planning meeting are presented in a special wiki page showing the agreed-upon sprint backlog (Figure 9).

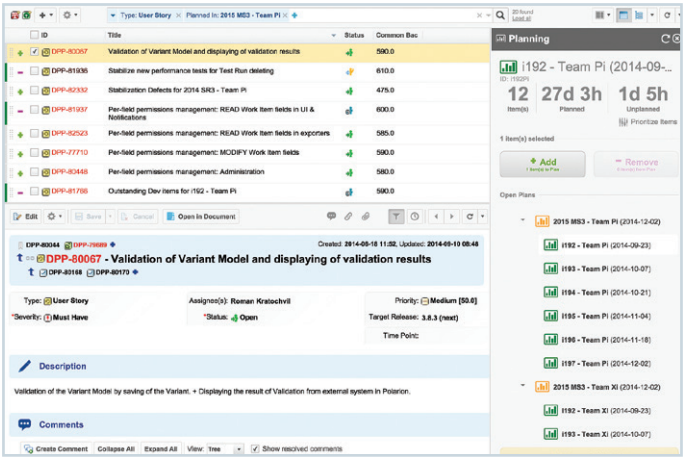

Figure 9: Sprint backlog in the integrated wiki

Slipping user stories

We pay special attention to user stories committed to a sprint, but not completed. It is very natural to slip to the next sprint because "it's just taking a little longer."

One expects that as soon as it is moved to next sprint, it will be done on the first day. Experience shows that developers often leave unfinished user stories to the end of the iteration because they are easy to complete. But in reality, they get behind with other tasks, and the slipped user story remains unfinished and slips even further into the next sprint.

A critical question in our planning meetings is: "If this user story was not addressed in the last sprint, how can we ensure that our new commitment to this user story will actually be realized?"

Daily scrums

Daily scrums may be the most complicated part of Scrum because it requires changing perceptions. Too many of us interpret meetings as a means of getting tasks and reporting back, but Scrum in general, and daily scrums in particular, are about helping the team to understand the current situation and to adjust if necessary. Daily scrums let team members synchronize, and all can check whether sprint goals are still feasible, and if not, make decisions about what to change. No reports are made, and the Scrum Master poses a simple question: "Are we sure we'll meet our sprint goals? Please show/explain how we do that!" We use the wiki task board to track progress of our sprint execution (Figure 10).[5]

Since our teams are small, we move daily Scrums to the lunch hour. When the most important questions are answered the team may go to lunch and discuss low-level level details, if needed.

The assessment meeting

Every iteration ends with an assessment meeting, in which every developer presents his/her work, either as a document (if the task was to "specify," etc.), or as a demo of the implementation in the product. Each user story should already have been tested by QA.[6]

For the assessment meetings we check only those items marked as "done."[7] As input for the assessment meeting we use yet another, more compressed, variant of the task board (Figure 11).

The final part of the assessment meeting is for introspection and lessons learned. Ideally, it is a time to discuss how we could optimize the process to implement more over a sprint, but typically we find ourselves trying to identify things that were less than perfect in our process, and/or the implementation of some feature in the sprint. We also try to identify additional synchronization risks, problems of communication, involve additional people to show them a dependency that was not fulfilled, and discuss other subjects.

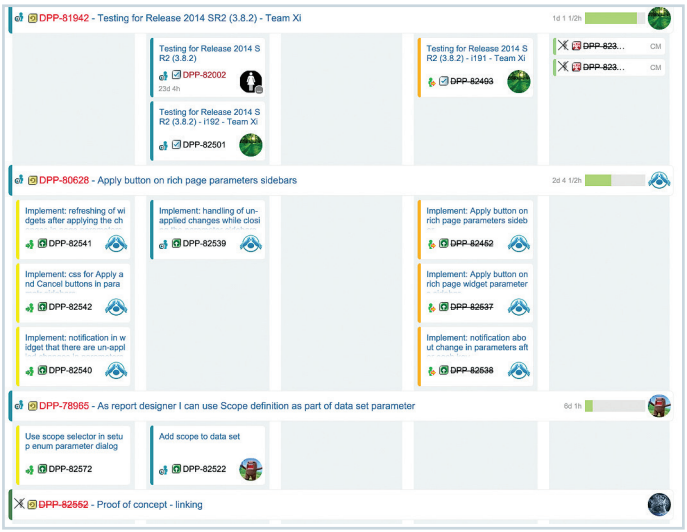

Figure 10: Task board in the wiki

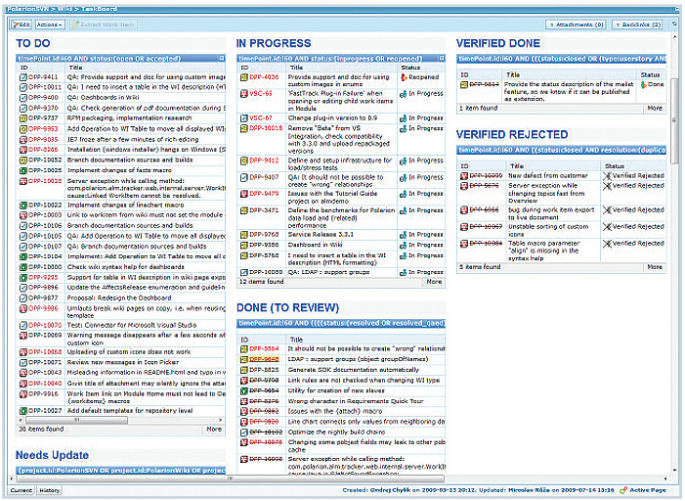

Figure 11: Task board summary on the wiki

The sprint: development

During a sprint, the development team continuously integrates all changes, and updated versions of the product are installed on the internal servers daily to prove stability and allow earlier testing of new functionality by other people (testers, document writers and others).

Every developer should know his/her personal plan, which matches the team plan set during the planning meeting. Developers track tasks via:

- Personal queries, like "assigned to me in current time point"
- Email notifications of newly assigned work items
- The Live Plan chart and corresponding wiki pages in the Polarion system

Burn-down charts

We configure the Live Plan view to show only entities assigned to a time point (usually the current sprint). It shows only leaves of the work item work breakdown structure (that is, it doesn't render user stories on the plan if it has child items – improvements, tasks or defects) (Figure 12).

The Live Plan also reflects all non-working days (configured in the global working calendar), and personal days off (configured in developers' personal working calendars). We get very clear information about whether the sprint goals are still achievable in the sprint time frame. The plan is ordered by priorities and severities in the way developers have agreed upon in the planning meeting. Correspondingly, less important items should be at the end of the plan. We also have wiki pages that highlight the progress of the team and remaining time (Figure 13).

Polarion's road map view also gives a clear picture of the work items in tabular form (Figure 14).

Testing and documentation

As implementation enters the "implemented" state, it should be taken over to QA and documentation. There is automatic testing through unit tests and so on, but every feature should pass QA control to ensure consistency of the implementation, acceptable levels of usability, that licensing and configuration for different product lines is correctly implemented, and that there is common user acceptance.

Typically QA and documentation starts in parallel with development based on a specification document (normally a wiki page). Final definition of the test cases and documentation typically happen at the point when implementation is really considered done, and the first round of review (also by QA) is passed.[8] The user story is populated with corresponding QA and documentation tasks (there may be several of each), and the user story owner sets the flags that proper QA and documentation are complete.

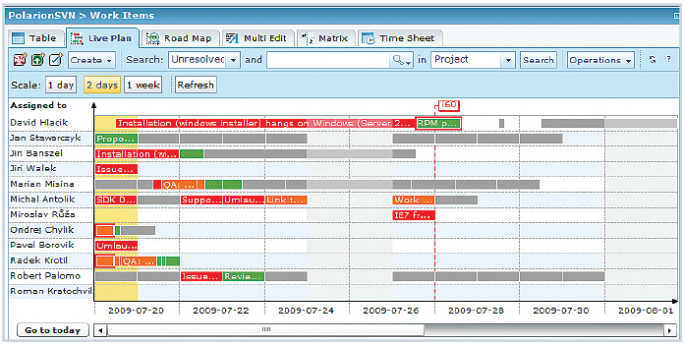

Figure 12: Burn-down chart in the Live Plan

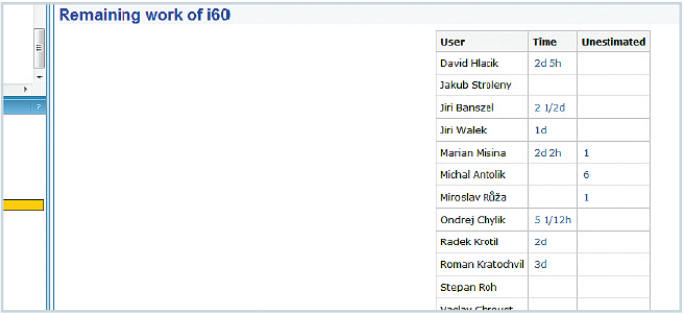

Figure 13: Remaining time estimate in the wiki

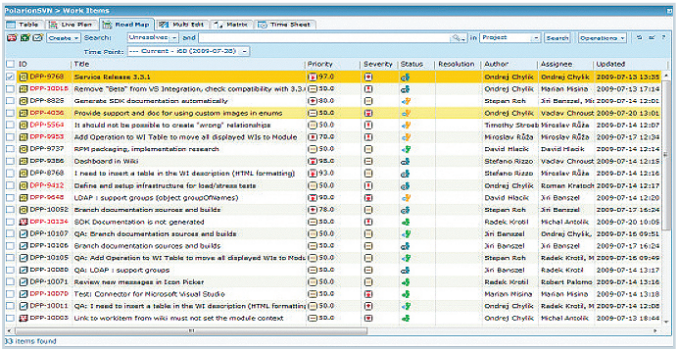

Figure 14: Polarion ALM road map view

Conclusion

This paper provides insights into how Scrum and Polarion ALM can work in conjunction. Possible next steps include:

- If you are new to Polarion ALM, our test drive server will enable you to explore Polarion ALM as much as you like without any time limits. For information and to create your user account, visit https://polarion.plm.automation.siemens.com/products/alm/demo.
- You can find a constantly growing collection of extensions for Polarion at http://extensions.polarion.com, including examples of the task board, workflow functions, integrations with third-party solutions, project templates and more.
- You can ask questions and discuss with other customers your approach to using Polarion at http://forums.polarion.com.

Notes

1. This is a string field, because it should allow values like "before version 3.2.1" or "after nightly build Apr 12th 2008."
2. Other work item types also have the customer attribute, but it reflects who has requested the feature or proposed an improvement, so its role is less important than in the defect type.
3. You might also configure a special "hat" product owner. We often use our own table settings to expose the PBP column, so we can easily reshuffle items.
4. LivePlan reveals over-/under-tasked people, potential bottlenecks, etc. as soon as you enter tentative plan data.
5. The wiki task board is a free extension available on the Polarion extensions portal at http://extensions.polarion.com.
6. Documentation may not happen in parallel with each iteration. The reality is that documenting all things together is not always possible. We use a special user story (UNDONE: release X.Y.Z) to link all the things to be addressed in a stabilization sprint before a corresponding release.
7. From a workflow point of view, user stories are marked as "implemented" (programming is finished), "done" (QAed, documented), "verified-done" (when corresponding stakeholder agrees that this functionality is really what was requested and expected). Those lacking documentation are still marked as "done," anticipating completion of documentation in a stabilization sprint.
8. Otherwise unforeseen issues might cause implementation to vary from the original specification and lead to refinement or even change of the specification. Of course the product owner or corresponding stakeholders are the ones who ultimately decide any change of specification.

Siemens PLM Software

Headquarters
Granite Park One
5800 Granite Parkway
Suite 600
Plano, TX 75024
USA
+1 972 987 3000

Americas
Granite Park One
5800 Granite Parkway
Suite 600
Plano, TX 75024
USA
+1 314 264 8499

Europe
Stephenson House
Sir William Siemens Square
Frimley, Camberley
Surrey, GU16 8QD
+44 (0) 1276 413200

Asia-Pacific
Suites 4301-4302, 43/F
AIA Kowloon Tower,
Landmark East
100 How Ming Street
Kwun Tong, Kowloon
Hong Kong
+852 2230 3308

About Siemens PLM Software

Siemens PLM Software, a business unit of the Siemens Digital Factory Division, is a leading global provider of product lifecycle management (PLM) and manufacturing operations management (MOM) software, systems and services with over 15 million licensed seats and more than 140,000 customers worldwide. Headquartered in Plano, Texas, Siemens PLM Software works collaboratively with its customers to provide industry software solutions that help companies everywhere achieve a sustainable competitive advantage by making real the innovations that matter. For more information on Siemens PLM Software products and services, visit www.siemens.com/plm.

www.siemens.com/plm

55674-A9 4/17 F