IBM

developerWorks.

Agile planning in real life

What did and didn't work in developing releases with agile planning

Skill Level: Intermediate

Steffan Surdek (ssurdek@ca.ibm.com)
User Experience Lead
IBM

31 Mar 2009

Are you part of a team that wants to get on board the agile planning bandwagon? Are you using iterative development and still stuck doing "waterations"? This article answers the question, "How do I start developing releases with agile planning?". It shows what worked, as well as the mistakes made, to illustrate a coherent and realistic understanding of the basics of agile planning.

"We're agile, we do iterations." This is a statement you hear often from teams claiming to do agile development. There's a lot more to being agile than doing your development in iterations. Most development teams typically have specific releases with pre-determined content and pre-determined dates. At the start of each release, the management team meets with the development team for the ritual arm-twisting for commitments on both the release date and the content.

Nutshell: Agile software development

Agile software development is a group of software development methodologies that are based on similar principles; these methodologies usually promote a project-management process that encourages frequent inspection and adaptation, engineering best practices that facilitate fast delivery of high-quality software, and a leadership/business philosophy encompassing teamwork, self-organization, accountability, and development aligned with the needs of the customer and the goals of the company. Modern operations management and analysis approaches like the following share similar conceptual foundations:

- Lean manufacturing/production: A production practice

© Copyright IBM Corporation 2009. All rights reserved.

that considers wasteful any resource expenditure that doesn't create value for the end customer.

- Soft Systems Methodology: An approach that is best for the analysis of complex situations where divergent views about the definition of the problem abound (that is, "soft" problems like "How do you manage disasters?").
- Six Sigma: An approach which seeks to identify and remove the causes of defects and errors in processes using a set of quality management methods and creating a special infrastructure of people within the organization ("Black Belts" etc.) who are experts in these methods.

The challenge both teams face is that management tries to dictate both the release date and the content while the development team feels that they can only achieve one or the other. In reality, the content side of this equation may be more flexible than what the development team initially believes. Unfortunately, development teams do not always realize this and the results can often can be seen in the quality of the product at the end of the release.

Below is a list of topics and questions that can help you move your team from doing plain iterative development to agile development:

- General advice about transitioning a team to use agile planning is discussed.
- Guidance on iterations is discussed. How long should they be? Should a team only be working in iterations during a release? What should the team be doing between releases?
- To complete the knowledge foundation, user stories and the product owner are discussed. What are user stories? Who should be writing them? What is the difference between an epic and a story? How long should the work contained in a story be? Who is the product owner? What is the role of the product owner in the process?
- Plan at the product, release, and iteration levels because each of these planning levels has its own backlog. What are the product, release, and iteration backlogs? What should these different backlogs contain? What level of estimation is required for each planning level? What is the difference between a user story and a task?
- What is team velocity and how can you measure it? What kind of information can you extrapolate from team velocity?
- References to materials used in the course of learning is discussed in this article. Useful links, tools, and additional information on agile planning is provided.

© Copyright IBM Corporation 2009. All rights reserved.

Before you start

A lack of education and understanding can be a hinderance, so here are some things to consider before you start to transition:

- **Understand what you want to get out of the process.** Change is hard for some people and you need to understand where you are willing to make compromises early in the process in order to get the results you want. You can always push for more once you've demonstrated you are getting the desired results. Express that what you want is the next step in the evolution of the process.
- **Don't start the transition on a whim.** If the process is being forced on the team, and the team has not been properly trained, this can create a lot of bad feelings and a general lack of confidence in the new process.
- **Know that education is your friend.** Learn what you need to learn; get familiar with it and then get comfortable with it. Try using agile methods on a small-scale project first to work out some of the kinks. Once you've learned, educate the rest of your team. Ensure everyone uses the same vocabulary and has the same understanding of the process before you begin. This education process will help you find the gaps in your own knowledge.
- **Commit to the process.** Once you start, don't look back. Yoda must have had agile in mind when he said: "Do or do not ... There is no try."

The short version: You *need* a level of commitment from your team to be successful. To get their commitment, you need to believe in the process yourself. Agile planning can work for your team if you are committed to doing it right.

Iterations

Let's start at the beginning with a popular question: *How long should your iterations be?* The goal of doing iterations is to allow the team to get feedback early and often from your stakeholders (customers, the product owner, and others).

A common iteration length is typically two or four weeks (sometimes up to six weeks). Shorter iterations are a bit easier to plan since there is more of a sense of urgency because of the quick turnaround. Longer iterations give the false impression that there's a lot of time for people to do their work—this can cause problems.

Iterations are the heartbeat of your team. You should always be doing iterations whether you are currently working on a release or not. One of the benefits of doing this is that it allows the team to schedule early research (*architecture spikes*), or

© Copyright IBM Corporation 2009. All rights reserved.

prototyping, or user story creation for the next release before actually working the "official" iterations of the release. These activities could help in identifying risks at an early point and coming up with mitigation plans and having them in place when needed.

You might find it is mostly better not to use variable iteration lengths during a release just to meet a specific release date. You could encounter the following challenges:

- **Resistance to change:** Constantly changing iteration lengths did not demonstrate that the team leaders had confidence in the new process.
- **Unknown team velocity:** The team did not try to measure velocity, but it would have been challenging to get a consistent measure of how much work the team was doing with varying iteration lengths.

One exception for a shorter iteration length is with the final iteration. If the final iteration contains mainly endgame activities such as globalization and bug fixing, it is less disruptive to the team at this point to have a shorter iteration.

As an alternative to scheduling a shorter iteration length for the final iteration, you can schedule a shorter length (like two weeks) for all iterations. By using this approach, if the release date is not at the end of the iteration, you will be able to wrap up the work for the release a week earlier.

On the last day of an iteration, schedule time for a demonstration and retrospective of the iteration. The demonstration of features can give everyone visibility on the progress made by the team and also provided an opportunity for early feedback.

User stories

User stories are one-liners that state customer requirements. Imagine them as what you would tell a customer to explain what you are doing. User stories are focused on what the customer needs; they do not need to spell out how to implement the feature.

The product owner talks with customers to understand their requirements and creates an initial set of high-level user stories. To facilitate the creation of more fine-grained user stories, you may want to create a customer team with a product manager, testers, real users, and interaction designers. You can use the following template for writing user stories:

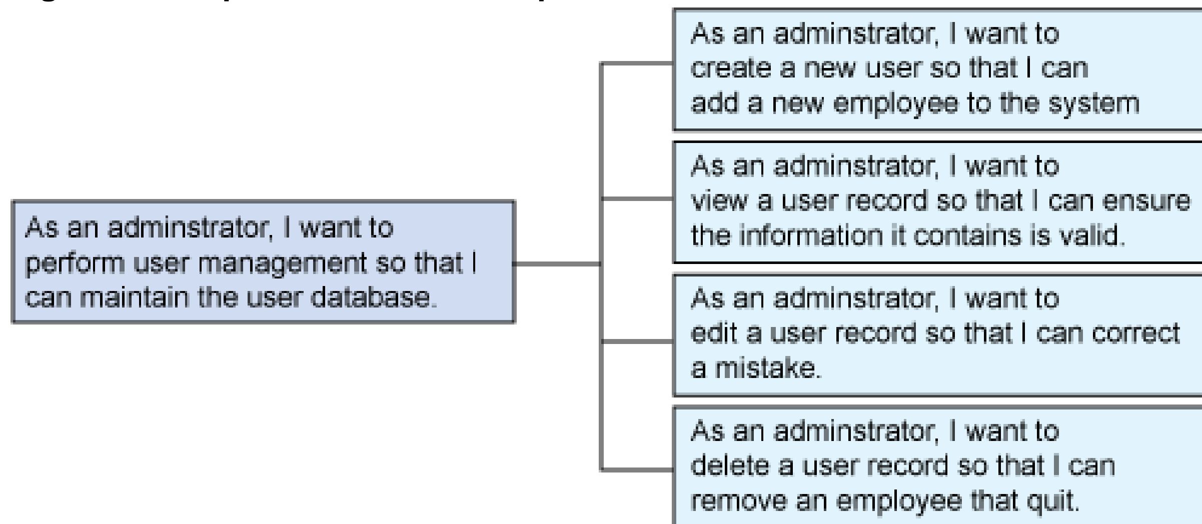
As a **<role>**, I want to **<goal>** so that **<business value>**.

User stories can have a hierarchy as well. Epics are user stories that talk about features or functionality at a higher level. You can iterate over these, break them down and generate new user stories and epics. Figure 1 illustrates an epic that was

© Copyright IBM Corporation 2009. All rights reserved.

decomposed into multiple smaller user stories.

Figure 1. An epic broken into multiple user stories



User stories represent the work the team must accomplish to meet the requirements of the system. How big should the user stories be? For epics, it doesn't necessarily matter; these will be broken down in smaller pieces later. For stories that the team will work on during the iteration, the work should take two to three days to build and should be functionality driven whenever possible.

For example, user stories can be sized to fit the iteration, such as using four-week iterations. This made for potentially big stories. At the end of the iterations, you might find that the team is not able to contain the work they had committed to.

You could consider having high-, medium-, and low-priority user stories in the iteration. That way, if the team ran late, the low-priority stuff could fall out of the iteration. However, this solution may not address the root causes of the problem.

It is normal to have a lot of user stories in your release backlog. Look at it as a reality check. If you have fewer stories but these stories take more time to implement, you don't have the good representation of the work that you need to have.

You should try taking the building blocks approach when building user stories. What is the minimum functional behaviour you can deliver? Start from that point, then create another story for the next set of functional behaviour and keep going until you have created stories to cover the entire feature.

It's important to note that stories may cut through the different components of your application. In this case, the team just needs to create the tasks for each component during the iteration planning phase.

© Copyright IBM Corporation 2009. All rights reserved.

The product owner

The next thing to do is to identify your product owner. The product owner is someone who is in contact with the customers and has an understanding of their needs. The product owner identifies the features that are required in the product for it to be successful on the market and meet customer needs.

The relationship between the product owner and development team is very similar to a customer and provider relationship. The important point to remember about agile planning is that the development team should always be working on items the product owner considers important. This is why both the product backlog and release backlogs are prioritized lists.

Even if the product owner is the ultimate decider in terms of priorities, there still needs to be a healthy relationship between the development team and the product owner. The development team must ensure that its needs, in terms of architecture work that is required to deliver on the release backlog, are understood and properly prioritized.

The product owner's responsibility is ensuring the product and release backlogs are properly prioritized to ensure the team is always working on the most important work items.

Product backlog

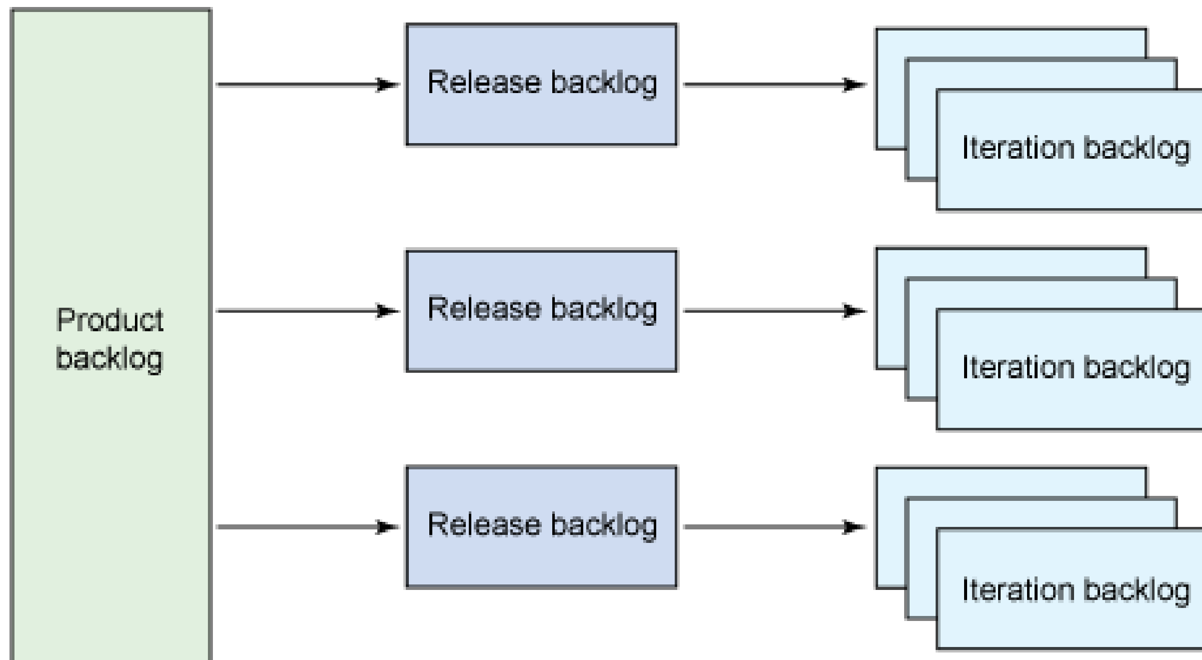
The product owner is responsible for the product backlog, a prioritized set of user stories that need to be implemented in the product.

Your team may already have a requirements or a line-item database. If this is the case, these make a good starting point to build your product backlog. To convert your existing database, the product owner should go over the requirements database and identify the high-level user stories associated with each of them.

The backlog needs to be prioritized because it allows the product owner to select the pieces of a requirement that are more critical to implement. One requirement may break down into ten user stories; however, if the development team completes the five most critical stories, it may be sufficient to meet the customer requirement.

Figure 2. How the product backlog feeds all the others

© Copyright IBM Corporation 2009. All rights reserved.



The high-level stories in the product backlog need to be estimated by the team using story points. All groups involved in delivery of the feature (development, quality assurance, and documentation) participate in the estimation process.

The product backlog is meant to be dynamic with items being added, removed, and reprioritized. At the same time, it needs to be maintainable. There should not be more work than for 18 months. When that threshold is passed, lower priority items will get knocked off the list.

Why should the backlog be kept short? This suggestion comes from Lean principles (see sidebar). There's no purpose in keeping more work in a queue than you can realistically do in a reasonable amount of time. If a good idea falls off the backlog, it can return if it is a really good idea, and you are in touch with your customer needs. So let stuff fall off the list and don't try to keep these in some secret, hidden, back-pocket backlog list.

Release backlog

At the start of a release, the product owner looks to the product backlog and identifies the user stories to be included in the next release. These items are then moved to the release backlog.

Usually, this is the point in the process where the ritual arm-twisting begins. The development team must now work closely with the product owner to ensure that the work that was identified looks appropriate for the size and duration of the release. Tracking the team velocity during the release will provide a clearer picture of how

© Copyright IBM Corporation 2009. All rights reserved.

much of this work will be achieved in the end.

During the creation of the release backlog, the time is right to break down the high-level user stories into smaller user stories. To facilitate the creation of these stories, you can create a customer team with a product manager, testers, real users, and interaction designers.

The customer team works in concert with the product owner to prioritize the new stories and add them to the release backlog. The development team then estimates these new stories and assigns them story points. The goal at this stage is to give development a good sense of what is required in the release in smaller chunks to estimate.

As the development team works through user stories during the iterations, the product owner and development team gain knowledge. This knowledge may generate new user stories or demonstrate that some stories are less critical. For new stories, they need to go through the estimation and prioritization process once again. For less critical stories, the product owner should remove them from the release backlog or lower their priority in the backlog. The team can also use this new knowledge in order to change estimates of existing stories in the backlog.

The release backlog is dynamic, but once the iteration starts, the product owner can no longer change the work that was selected for the iteration. If, after starting the work, the team gains knowledge indicating certain items will not fit in the release or in the iteration, this is a different situation (I'll discuss it later in the article). During the iteration, the team must be able to focus on their deliverables.

Iteration backlog

At the start of the iteration, the product owner can re-prioritize the release backlog so the development team knows the items it needs to work on next. The product owner and development team will take a half-day on the first day of the iteration for the iteration planning meeting. During this meeting, the team selects as many high-priority items as it can contain from the release backlog and adds these to the iteration backlog.

The trick to contain this to a half-day is to have your release backlog ready to go. If the team is creating the user stories from iteration to iteration (instead of as early as possible), you can waste valuable time, and it can take longer to do your planning. You'll also never really know how much work is planned for your release.

During the planning meeting, for each user story in the iteration backlog, the team needs to break down all the tasks required for the story to be considered complete. The tasks from everyone involved in the story (development, quality assurance, documentation) need to be identified. The team estimates each of these tasks in

© Copyright IBM Corporation 2009. All rights reserved.

hours and adds these estimates to the plan.

The biggest concern here is over-commitment. In your initial iterations, look at the number of hours in the estimates to ensure the work will fit in the time allocated for the iteration. If the team finishes work ahead of schedule, it can always go back to the release backlog and pick some additional work items that can be completed inside the iteration.

The other concern is not identifying all the tasks required to complete a story. This will unfortunately happen in your initial iterations. These forgotten items pile up and may cause you to not be able to complete a story in an iteration. Here are two examples of these mistakes:

- Having a bucket called "Testing" which doesn't include the time for writing the test cases.
- Not including a task for the development team to write automated unit tests.

Before the team goes back to the release backlog well, ensure the stories in the iteration are really done (then ensure they are done two more times!). In other words, are all the tasks complete? If some testing is outstanding for a user story, it will be to your advantage to have the developer help the tester complete the testing. If there are defects remaining for user stories developed in the iteration, fix them now before moving on. Build quality in; it's not how much work you do in an iteration, it's all about how bug free the delivered work is.

You will want to establish exit criteria for your iterations. For example, all high-priority (severity one and two) defects must be resolved for the story to be considered complete. Any unresolved low-priority (severity three and four) defects must be resolved in the following iteration and will be added to the top of the iteration backlog. It's important not to carry your defects for too long a time. You need to build quality into your product as you go. This way, you will be in a much better place during the end game part of your release.

If the team realizes during the course of the iteration that a user story is more complicated than originally anticipated and cannot be contained in the iteration or even in the release, then you need to raise this issue as early as possible. It is important to identify your options. Is there a way to scale down the story and still deliver something of value to the customer in the current release? If so, create appropriate user stories and discuss them with your product owner. The scaled down version may just be an interim measure: What you would do if you had the time to do it right? Identify the user stories for these items and discuss them with your product owner. They may make it in subsequent releases of the product.

The team can add any kind of work in the iteration plan. If an architecture spike is

© Copyright IBM Corporation 2009. All rights reserved.

required for a feature, add a user story for it to the release backlog and have tasks such as "Design," "Building Prototype," or "Research." The good thing about adding a user story to the release backlog for the spike is that it allows the product owner to prioritize it. If the product owner feels the cost of doing the research is too high, the architecture spike story will be lower on the list or the feature will be de-scoped from the release.

Story points

In the product and release backlogs, the development team estimates and assigns story points to each user story. Why use story points instead of a unit of time such as hours? The main reason is because in the early stages of development, you don't know how much work you really have for a story. And if you do all the analysis and design up front, you may be preventing the design from evolving as you know more about the feature.

Another reason to use story points versus units of time is that the ideal time from one person to another can vary. In cases such as this, if the person doing the estimate is not the person doing the work, the estimate may be wrong.

Finally, another reason to use story points is that what was originally an ideal time estimate may be misinterpreted as an elapsed time estimate. Elapsed time is the time it actually takes to do the work after factoring in all of the interruptions the employee has in a normal workday. So, for example, a task estimated at five days may actually take eight to nine days to complete once the daily meetings, e-mails, phone calls, and reviews are taken into account.

When using story points, what you are actually doing is comparing the relative complexity of one feature to another. If you take a story worth one point and compare it with a story worth five points, what you are essentially saying is that the second story is five times bigger to do than the first one. Notice that it was not stated that it would necessarily take five times more time to do.

There may be user stories that the team cannot estimate because there are still too many unknowns. In cases such as this, the team can add a user story to the backlog to do some additional research (this is also referred to as an architecture spike). Once the research is done, the team can revisit if they are now in a better position to provide an estimate.

Initially, it will be challenging to estimate story points because your team has no reference points to compare with. As the team does more and more estimates, it will start getting better at it. To do the story point estimates, you can use the Planning Poker technique.

Planning Poker is a consensus-based estimation technique for estimating, mostly

© Copyright IBM Corporation 2009. All rights reserved.

used to estimate effort or relative size of tasks in software development. It attempts to minimize anchoring (early comments by team members that "anchor" a task time by expressing it aloud) by asking each team member to play their estimate card (the card that express the values 0, 0.5, 1, 2, 3, 5, 8, 13, 20, 40, 100 but do not expressing units; units are decided by the moderator) such that it cannot be seen by the other players. After each player has selected a card, all cards are exposed at once.

Velocity

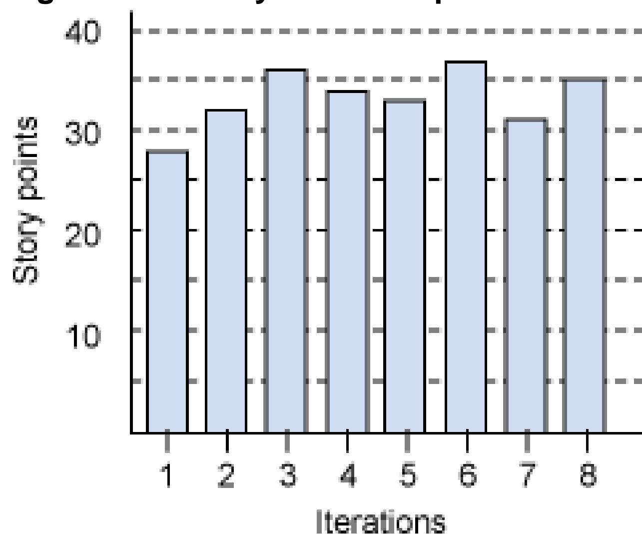
Velocity is the long-term tracking of how much work has been done by a team per iteration. It is measured in the same units as in the release backlog, meaning story points.

The team initially assigns story points for each story in the release backlog. At the start of each iteration, the team selects the stories it will work on during the course of the iteration. To earn the points assigned to a story, the team must complete all tasks for that story within the iteration.

If the team does not complete one or more tasks of a user story, the team earns zero points for the story at the end of the current iteration and the story will be deferred to the next iteration. The team will earn the points for the story in the iteration in which it completes the remaining tasks.

As the team completes iterations, tallies up the story points, and looks at the data of previous iterations, a picture such as the one in Figure 3 begins to emerge. Notice how the team does not earn the same number of points in each iteration.

Figure 3. Velocity over multiple iterations



The nice thing about velocity is that after a few iterations, you can start using the

© Copyright IBM Corporation 2009. All rights reserved.

numbers your tracking charts provide to extrapolate how much of your remaining release backlog items you will be able to complete.

Table 1. Story points per iteration

Iteration	Points
1	28
2	32
3	36
4	34
5	33
6	37
7	31
8	35

Table 1 shows story points per iteration. Using the data in Table 1, you can extrapolate the following information:

- The average number of story points per iteration is 33.
- The current velocity of the team is 35 story points.
- The average of the three slowest iterations is 30 story points.

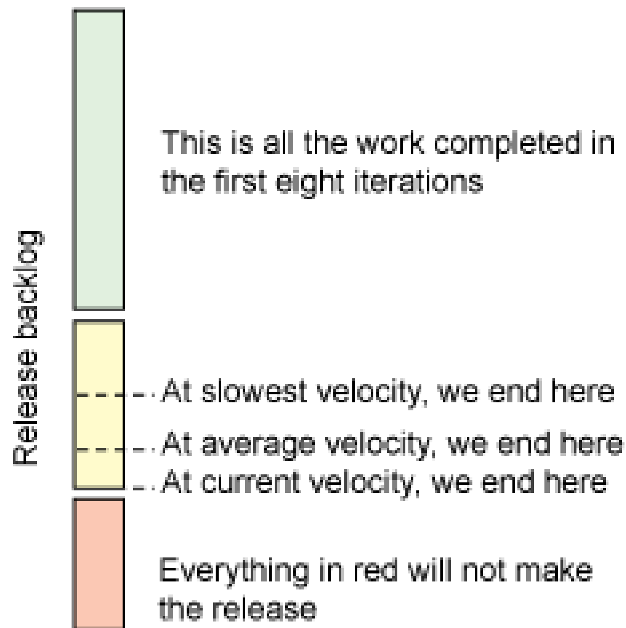
So assuming that you have 6 iterations left in your release, you can begin to make the following assumptions:

- At average velocity, the team will complete an additional 198 story points (6 x 33).
- At current velocity, the team will complete an additional 210 story points (6 x 35).
- At slowest velocity, the team will complete an additional 180 story points (6 x 30).

In Figure 4, you can see the release backlog split into three sections. The middle section represents the work that can be contained in the last 6 iterations using the assumptions above.

Figure 4. Extrapolating where the work ends

© Copyright IBM Corporation 2009. All rights reserved.



What Figure 4 also communicates is that the development team cannot complete all the user stories in the release backlog. However, if the team always works from the top of the list down, the odds are high that the team has always been working on the most important items for the release.

In conclusion

The main goal of agile planning is having as much of the "known" work as possible on the table and visible to all. This article puts emphasis on the "known" because as you gain knowledge about what you are doing, you may be adding new stories to the backlog. This allows the product owner to continually prioritize the release backlog and ensure development is always working on what is most important.

Product owners usually survive if not all stories make it into a release. They tend to get upset about it, though, when the team spends too much time on items that are not critical and then cannot fit all the important stories in the release because they ran out of time. So treat product owners as your customer and continually give them what they want first.

Any opinions I've expressed in this article are my own and not necessarily those of IBM.

© Copyright IBM Corporation 2009. All rights reserved.

Resources

Learn

- Check out these articles used in the creation of this article:
  - *Agile Estimating and Planning* (Prentice Hall PTR, 2005) as a guide to estimating and planning agile projects, complete with philosophical and practical take on agile estimating and planning.
  - *User Stories Applied: For Agile Software Development* (Addison-Wesley, 2004) takes you into the importance of user stories in agile development.
- Also see training for agile development.
- Planning Poker is a technique that helps distributed teams estimate together.
- The series "Architectural manifesto: Adopting agile development" (developerWorks, March 2008-January 2009) can help you decide if agile development is right for your needs:
  - Part 1: About the processes, benefits, and requirements
  - Part 2: How its used for different projects; customer effects
  - Part 3: The role of stakeholders
  - Part 4: Defining requirements
  - Part 5: User stories and prioritization
  - Part 6: From the customer's viewpoint
  - Part 7: Estimating work effort
  - Part 8: Modeling
  - Part 9: The agile-SOA link
- In this podcast, "Scott Ambler on Agile development" (developerWorks, April 2007), IBM's Agile Development Practice expert explains this iterative and incremental approach to development, lays out the business case for it, and dispels some myths in the process.
- "Agile software development: A tour of its origins and authors" (developerWorks, March 2007) aims to show you what "agile" truly means.
- In the developerWorks Linux zone, find more resources for Linux developers (including developers who are new to Linux), and scan our most popular articles and tutorials.
- See all Linux tips and Linux tutorials on developerWorks.

© Copyright IBM Corporation 2009. All rights reserved.

Get products and technologies

- With IBM trial software, available for download directly from developerWorks, build your next development project on Linux.

About the author

Steffan Surdek
Steffan Surdek is the User Experience Lead and self-proclaimed Agile Champion for the WebSphere® Business Services Fabric team. He uses his experience facilitating the two-day Software Group Agile Workshops to help his team work on improving their agile processes. You can follow his blog and contact him online via his Web site at http://www.surdek.ca.

Trademarks

IBM, the IBM logo, ibm.com, DB2, developerWorks, Lotus, Rational, Tivoli, and WebSphere are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. These and other IBM trademarked terms are marked on their first occurrence in this information with the appropriate symbol (® or ™), indicating US registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. See the current list of IBM trademarks.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft is a trademark of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

© Copyright IBM Corporation 2009. All rights reserved.