How do you apply Scrum to maintenance and legacy code improvements? [closed]

Asked 16 years, 1 month ago Modified 13 years, 2 months ago

Viewed 13k times

26

Closed. This question is off-topic. It is not currently accepting answers.

Want to improve this question? Update the question so it's on-topic for Stack Overflow.

Closed 13 years ago.

Improve this question

As the title suggest... How can I apply a scrum process to anything that doesn't work on new code and can be estimated to some degree?

How can I apply a scrum process to maintenance and emergency fixes (which can take from 5 minutes to 2 weeks to fix) type of environment when I still would like to plan to do things?

Basically, how do I overcome unplanned tasks and tasks that are very difficult to estimate with the scrum process?

or am I simply applying the wrong process for this enviroment?

project-management scrum

Share

Improve this question

Follow

asked Nov 13, 2008 at 0:35

Jonathan

2,223 ● 4 ● 21 ● 25

12 Answers

Sorted by: Highest score (default)

22

> Basically, how do I overcome unplanned tasks and tasks that are very difficult to estimate with the scrum process? or am I simply applying the wrong process for this enviroment?

You're using the wrong process for this environment. What you need is a stack/queue management process which is separate to your planned/estimated SCRUM development process.

The reason for this is simple and twofold:

1. As you mention in your post, it is often very difficult to estimate maintenance tasks, especially where legacy systems are involved. Maintenance tasks in general and legacy systems specifically have a tendency to involve 'curly' problems, or have a long 'tail', where

one seemingly simple fix requires a slightly more difficult change to another component, which in turn requires an overhaul of the operation of some subsystem, which in turn... you get the point.

2. Quite often when dealing with maintenance tasks, *by the time you have finished estimating, you have also finished solving the problem*. This makes the process of estimation redundant as a planning tool. Those who insist on dividing estimation from solving the problem for maintenance tasks are simply adding unnecessary overhead.

Put simply, you need a queueing system. It will have these components:

- A 'pool' of tasks which have been identified as requiring attention. Newly raised items should always go into the pool, never the queue.
- A process of moving these tasks out of the pool and onto the queue. Usually a combination of business/technical knowledge is required.
- A queue of tasks which are clearly ordered such that developers responsible for servicing the queue can simply pick from the front of it.
- A method for moving items around in the queue (re-prioritising). To allow 'jumping the queue' for critical/emergency items.
- A method for delivering the completed items which does not interrupt servicing the queue. This is

important because the overhead of delivering maintenance items is usually significantly lower than development work. You don't want your maintenance team sitting around for a day waiting for the build and test teams to give them the ok each time they deliver a bugfix.

There are other nuances to queue management, but getting these in place should set you on the right path.

Share Improve this answer

Follow

answered Nov 13, 2008 at 6:58

Ben McEvoy

654 ● 4 ● 8

2 Voting up, especially for "by the time you have finished estimating, you have also finished solving the problem". – UserPioneer Sep 10, 2010 at 8:49

11

If you have that much churn in your environment, then your key is going to be shorter iterations. I've heard of teams doing daily iterations. You can also move towards a Kanban type style where you have a queue which has a fixed limit (usually very low, like 2 or 3 items) and no more items can be added until those are done.

What I'd do is try out one week iterations with the daily stand-ups, backlog prioritization, and "done, done". Then reevaluate after 5 or 6 weeks to see what could be improved. Don't be afraid to try the process as is - and don't be afraid to tweak it to your environment once you've tried it.

There was also a PDF called Agile for Support and Operations in 5 minutes that was recently posted to the Scrum Development list on Yahoo!

Share Improve this answer

Follow

answered Nov 13, 2008 at 1:12

Cory Foy

7,212 ● 4 ● 32 ● 34

5

No one said that backlog items have to be new code. Maintenance work, whether bug fixes, enhancements or data fixes can be put into the Product Backlog, estimated and prioritized. This is actually one of the biggest benefits of using Scrum - no more arguments with users about whether something is a bug fix or an enhancement.

With Waterfall, there's a tacit understanding that bugs are the responsibility of the developers. Somehow, they are on the hook to fix them without impacting the development of new code and features. So they are "free" to the users, but a massive inconvenience to the developers.

In Scrum, you recognize that all work takes time. There is no "free". So the developers freely accept that something is a bug but it still goes into the Product Backlog. Then it's up to the Customer to decide if fixing the bug is more important than adding new features. There are some bugs that you can live with.

Share Improve this answer

answered Nov 13, 2008 at 3:42

Follow

DaveB
2,113 ● 1 ● 17 ● 16

2

> As the title suggest... How can I apply a scrum process to anything that doesn't work on new code and can be estimated to some degree?

On the contrary, I've heard teams find adopting Scrum easier in the maintenance phase.. because the changes are smaller (no grand design changes) and hence easier to estimate. Any new change request is added to the product backlog, estimated by devs and then prioritized by the product owner.

> How can I apply a scrum process to maintenance and emergency fixes (which can take from 5 minutes to 2 weeks to fix) type of environment when I still would like to plan to do things?

If you're hinting at fire-fighting type of activity, keep a portion of the iteration work quota for such activities.. Based on historical trends/activity, you should be able to say e.g. we have a velocity of 10 story points per iteration (4 person-team 5day-iteration). Each of us spend about a day a week responding to emergencies. So we should only pick 8 points worth of backlog items for the next iteration to be realistic. If we don't have emergency issues, we'll pick up the next top item from the prioritized

backlog.
CoryFoy mentions a more dynamic/real-time approach with kanban post-its in his response.

> Basically, how do I overcome unplanned tasks and tasks that are very difficult to estimate with the scrum process? or am I simply applying the wrong process for this enviroment?

AFAIR Scrum doesn't mandate an estimation technique.. Use one that the team is most comfortable with.. man days / story points / etc. The only way to get better at estimation is practice and experience I believe. The more the same set of people sit together to estimate new tasks, the better their estimates get. In a maintenance kind of environment, I would assume that it's easier to estimate because the system is more or less well known to the group. If not, schedule/use spikes to get more clarity.

I sense that you're attempting to eat an elephant here.. I'd suggest the following bites

- Working Effectively with Legacy code
- Agile Estimating and Planning
- Agile Project Development with Scrum

answered Nov 13, 2008 at 8:14

Gishu
137k ● 47 ● 226 ● 311

1

Treat all fixes and improvements as individual stories and estimate accordingly. My personal view is that things that take less then 10-15 minutes to fix should just be done straight away. For those that take longer, they become part of the current 'fix & improvement' iteration cycle. As with estimating regular requirements, you take a best guess as a team. As more information comes to light, and the estimates are off adjust the iteration and upcoming sprints.

It's hard to apply a iteration cycle to fixes and improvements as more often then not, they prevent the system from working as they should and the 'business' puts pressure for them to go live asap. At this point it may work out better moving to a really short iteration cycle, like one or two weeks.

Share Improve this answer

Follow

answered Nov 13, 2008 at 0:48

Hates_

68.6k ● 6 ● 33 ● 37

1

You ask about how to use a process for emergencies. Try to reserve the word "emergency" for things that require hacking the code in the production environment with the users connected at the same time. Otherwise, stakeholders are likely to abuse the word and call an emergency on anything they would like to have really fast. Lack of process does not mean out of control: somebody must be accountable for declaring the emergency, and somebody (best practice is somebody

else) must authorize the changes outside the normal process and then take responsibility for it.

Other than that, the suggestion of using each iteration to complete a number of fixes and improvements is probably the best way to go.

Share Improve this answer

Follow

answered Nov 13, 2008 at 1:05

coryan

1,201 ● 9 ● 7

1

This depends a lot on the application life cycle. If this is a 'Sunset' application to be retried soon, of course, the focus would only be on fixing the top priority bugs.

If the product is 'Mature' and has a roadmap and is continuing to evolve you would have fixes and enhancements to take care of. There's a lot of impetus to keep the design clean and evolve by refactoring. This may mean periodic minor and major releases [except for eFixes - emergency fixes/hotfixes]. You can pratice agile to your hearts delight as enhancement and fixes could be story boarded and made part of your Sprint backlog. The entire list would make your Product Backlog.

Bottom line: If you want to refactor and keep your application design clean [programmers tend to take shortcut if the focus is exclusively bug fixing], it could only happen with a 'living' application. One that is evolved and updated. And agile is natural fit.

Even if you have only fixes (that it's a Turing Complete ;) or Sunset application), it helps if they can all be rolled into a sprint and rolled into production end of each sprint. If the fixes need be rolled into production as and when they're fixed, it's much more difficult to apply Scrum.

Share Improve this answer

Follow

answered Nov 13, 2008 at 1:15

Vyas Bharghava

6,510 ● 9 ● 42 ● 62

1

We have applied scrum in this context.

Some of the keys of success.
1. Everyone in the enterprise buy the scrum idea (this is **crucial** for success)
2. Sprint of about 2 weeks (our 2-3 firsts sprint where of 1 week to understand the process)
3. Under no circumstance a point could be added to the current sprint 4. If a real emergency arise stop the sprint, do a retrospective and start a new sprint.
5. Take time for retrospection (time to share though about the last sprint and to analyze it)
6. In a sprint insert at least one task to improve the process (often added to the backlog in the retrospective); it's good for the troop morale and in the end of the day you will be in your way to have less emergency.
7. TIME BOXED daily stand up meeting

For the estimation, usually the more you estimate the more precise you become. What is good with Scrum is that each programmer pick his task and can set a new

estimate if he think it's not realist. And if you still have some issue with estimation, let your team find a solution... you can be surprise with what they come with.

For the 2 weeks fix. If it's the original estimation, cut it in smaller pieces. If you did an more optimistic estimation (let say 2-3 days), the issue should rise as a blocker in the stand up meeting. Maybe somebody else have ideas about how to fix it. You can decide to do some pair programming to find a solution. Sometime, just to describe the bug to another programmer help a lot to debug. You can also delay it after other task in the sprint. The idea is to deliver fully functional tasks. If you don't have time to fix it in full and to demonstrate it, even if your at 90% done (yeah! we know what it means), you consider it not done in the sprint. In the next sprint you will be able to address it with the correct time estimation.

Finally, from what I understood, Scrum is more about having "tools" to improve your process. You start with something simple. You do small iteration. In each iteration you have a **FIX TARGET** to complete instead of an infinite bug list. Because you pick your tasks from the list in the planning (oppose to being assign them), you become more engage to deliver it. With the stand up meeting, you meet your pair every day with your TODO list... you want to respect your engagement you did the day before. With the iteration, you take the time to talk to each other and to identified what's going good and what's should be improve. You also take action to improve it and to continue doing what's working. Don't be afraid to

change anything, even what I said ;) / even any basic of the Scrum itself... The real key is to adapt Scrum to what your team need to be happy and productive. You will not see it after one iteration, but many of them....

Share Improve this answer

Follow

answered Nov 13, 2008 at 6:12

Hapkido

1,421 ● 1 ● 9 ● 13

1

I'd highly recommend looking at what value sprints/iterations would give you. It makes sense to apply them when there are enough tasks to do that they need to be prioritized and when your stakeholders need to know roughly when something will be done.

In this case I'd highly recommend to combine three approaches:

- schedule as many incoming tasks as possible for the next iteration earliest
- use Yesterday's Weather to plan for how much buffer you need to plan for to deal with tasks that have to be dealt with immediately
- use very short sprints, so as to maximize the number of tasks that can wait until at least the start of the next iteration

In fact that is true vor *every* Agile/Scrum project, as they are in maintenance mode - adding to an existing, working system - from iteration 2.

In case that iterations don't provide enough value, you might want to take a look at a kanban/queuing system instead. Basically, when you are finished with a task, just pull the next task from a prioritized task queue.

Share Improve this answer

Follow

answered Nov 13, 2008 at 18:44

Ilja Preuß

2,421 ● 17 ● 15

1

In my opinion it depends on how often you have a 'real' release. In our specific case we have one major release each year and some minor releases during the year.

This means that when a sprint is done, it's not immediately deployed on our production server. Most of the times a few sprints will take place before we have our complete 'project' finished. Of course we demo our sprints and we deploy our sprints on our testing-server. The 'project' in his totality will undergo some end-to-end testing and will finally be deployed to our production servers -> this is a minor release. We may decide that we will not deploy it immediately to our production server, when it's for instance dependant on other products/projects that need to be upgraden first. We then deploy this in our major release.

But when issues arrise on our production server, immediate action may be required. So, no time to ask a product owner for the goal or importance (if we even have one for sunc an issues) because it blocks our clients from working with our application. In such an urgent cases,

these kind of issues will not be put into a Product Backlog or sprint but are pure maintenance tasks to be solved, tested and deployed as soon as possible and as an individual item.

How do we combine this with our Sprint? In order to keep our Team members focused on the sprint, we decide to 'opt in - opt out' our people into the Team. This means that one or more people will not be part of a Team for a certain Sprint and can focus on other jobs like these urgent fixes. The next Sprint this person will again be part of the Team and someone else will be responsable for emergency calls.

Another option could be that we foresee like 20% of the time in a Sprint for 'unplanned tasks' but this will give wrong indication about the amout of work we can do in a Sprint (we will not have the same amount of urgent fixes during each sprint). We also want our Team members to be focused on the Sprint and doing these urgent fixes in a Sprint will distract our team members. 'context-switching' will also mean time-loss and we try to avoid that.

It all depends on your 'environment' and on how fast urgent issues should be fixed.

answered Jun 23, 2009 at 20:44

DiVer

56 ● 1

1

Treat all "bug fixes" that don't have a story as new code. Estimate them, and work them as normal. By treating them as new stories you will build up a library of stories and tests. Only then can you begin to *pin* the behavior of the application.

Take a look at *Working Effectively with Legacy Code* by Michael Feathers. Here is a link to an excerpt. http://www.objectmentor.com/resources/articles/WorkingEffectivelyWithLegacyCode.pdf

-Jason

Share Improve this answer Follow

edited Jan 17, 2011 at 18:58

answered Nov 13, 2008 at 1:02

Jason Hernandez

2,967 ● 1 ● 19 ● 31

0

I have had some success by recognizing that some percentage of a Sprint consists of those unplanned "fires" that need to be addressed. Even in a short sprint, these can happen. If the development team has these responsibilities, then during sprint planning, only stories are committed to the sprint that allow enough headroom for these other unplanned activities to occur and be handled as needed. If during the sprint, no "fires" ignite, then the tam can pull in stories from the top of the backlog. In many ways, it becomes a queue.

The advantage is that there is commitment to the backlog. The disadvantage is that there is this hole in capacity that can be communicated as an opportunity to drag the team into non-critical tasks. Velocity can also vary widely if that gap in capacity is filled with unplanned work that is not also tracked.

One way I have gotten around part of this is to create a "support" story that fills out the rest of that capacity with a task that represents the available hours the team can allocate to support activities. As support situations enter the sprint that cannot be deferred to the backlog, then a new story is created with tasks, and the placeholder support story and tasks are re-estimated to account for the new injection. If support incidents do not enter the sprint, then this support story is reduced as backlog items come in to fill the unused capacity.

The result is that sprints retain some of the desired flow and people feel like they aren't going to get burned when they need to take on supporting work. Velocities are more consistent, burndowns track normally, but there is also a record of the support injections that happen each sprint. Then during the sprint retrospective we can evaluate the planned and unplanned work and if action needs to be taken differently next sprint, we can do so. If that means a new sprint duration, or a pointed conversation with someone in the organization, then we have data to back up the new decisions.

 answered Oct 21, 2011 at 15:13

Follow

Glenn

31 ● 1