Define done

One of my favorite phrases in software development is "feature complete." A figurative fig leaf for a nonplused product owner, it also doubles as a technical-sounding term for a team trying to fool stakeholders at a demo. It's one of my favorites because it makes me laugh each time I hear it.

I was working with a team who delivered two separate demos, two sprints apart, where they described the big feature they had been working on – but not finished – as being "feature complete." It was only after they almost got away with it for the second time that some innocent, but brave, meeting attendee raised their hand and asked "what comes after feature complete?"

The team looked at each other, then to the scrum master, then to the product owner, then again at each other, before a very junior member responded with "well that would be done – like actually done."

The attendee, satisfied with the answer, said "thanks." And everyone quickly moved on to the next agenda item so as not to call attention to the situation.

A team with a clear understanding of what it means for work to be done is a team with a superpower.

Beyond the obvious distinctions between something being "feature complete" or "shipped to production," there are all the little ambiguities about each individual work item.

Leaving planning with a crisp definition of what it means for the work to be done is key to success.

When I'm trying to define done, I seek to understand the following:

1. What are the boundaries of this feature?
2. How will this be accepted by the product owner?
3. Can this be completed with the resources we have now?
4. Do we have the same end in mind?

Draw the boundaries

It was a major version upgrade of the web framework. And it wasn't going well.

"Remind me again why we're doing this now?" asked the team lead, Andrea. She had been in meetings all morning, and was just getting up-to-speed on the work on the board.

The pair of developers playing whack-a-bug with the upgrade barely looked up from their pairing station.

"We've had three retros in the last couple of months when we identified all the problems caused by not keeping the libraries we use current," one of the developers, Hamid, reminded the lead.

"It's almost done anyway, we've changed so much we can't back it out now. It would be more helpful if you fixed some of these bugs," said the other, Jose.

"Did anyone let QA know? They're going to want to do a full regression if we're doing a major framework upgrade," Andrea asked, trying to stay calm.

"That's going to be a problem," noted, Maria, the scrum master, "we haven't resolved the outstanding items from the last full regression they did," she said

While the product owner was always open to the idea that the team needed time to improve the system, he wasn't technical and didn't always understand the tradeoffs. One thing he did understand, the pressure from

the business to wrap up the remaining work he had forecasted for the quarter.

This last-minute framework upgrades thing was complicating his plans. He didn't want to show up to the next product meeting with another round of bad news.

Effective planning means defining the acceptable boundaries for each work item.

After a couple of late nights, the framework upgrade was completed. The automated test suite was passing again. The team rallied together to get the work done, but QA found some issues in their full regression, and the product owner ended up being the bearer of bad news.

Boundaries help us understand things like:

- How "polished" the result needs to be
- How performant the implementation needs to be
- How much unrelated improvement work is acceptable

Scope vs boundaries

Most teams talk about the "scope" of the work. An attempt to define what the feature does or does not do; the behaviors of the feature. However, this definition of "scope" is limited. It fails to account for other ways the work can unexpectedly grow and expand.

I prefer to think about the boundaries of the work. What's "in bounds" and what's "out of bounds." Thinking

in these terms allows us to explore what's acceptable to do or not do as we plan the work.

Effective planning means defining the acceptable boundaries for each work item. These boundaries are subject to negotiation, and we should expect them to change over time or from one work item to the next.

Set boundaries around each piece of work and the team will be calling out "out of bounds" work before it's too late and you're trying to convince the product owner to "negotiate scope" before another failed sprint.

Minimum viable "polish"

Would adding animations to the user interface be inside or outside the boundaries of this work? How much should we worry about the written copy on our first pass? Is a simple table view enough to display this data?

Whether you call it "attention to detail" or "gold plating," it's going to come up in planning. But before planning starts, the team needs to agree on what's acceptable in terms of how the work looks to the end user. I recommend finding what's just "polished" enough and targeting that state for planning. Anything else is out of bounds, for now.

System improvements

Refactoring, experimenting with new technology, improving the test suite. These ongoing concerns are part of each decision and implementation choice made by the team. Decide upfront when the team makes improvements to the system and how much of their time is set aside for these tasks. Many teams find success by building in "slack" time and not running at 100% capacity.

Discussing and documenting them before the work starts means we'll have a clear idea of how to proceed *when* the work starts.

During planning I like to ask myself the following:

- Does our plan for implementing this work clearly define what we're not going to do?
- Are there any outstanding improvement tasks (e.g. from a retrospective) that we can negotiate with the product owner?
- Does this work touch any problematic parts of the system that have caused issues in the past?
- Are there outside forces (e.g. fixed deadlines) that suggest a smaller boundary around the work?
- Is the business looking for especially innovative or creative solutions with this work? Does that imply a larger boundary?

Define what's acceptable

The team was really excited, they finished the card a couple of days early and were ready to send it off to the product owner for acceptance testing.

"Great!" Exclaimed the product owner. "I'll take a look right now."

The team enjoyed about three minutes of glee before the message appeared in the chat.

"It's totally bombing when I try to upload the customer's data import. Want me to try something else?"

Effective planning defines how the work will be evaluated and what it takes to be considered acceptable by the business.

A few days earlier I sat in on this team's planning meeting about a feature to allow customers to upload exported files from another system. It was a pretty good planning session, and most of the details were covered. What they missed was understanding how the product owner was going to test the feature before considering it complete.

Earlier in the week the product owner got a real data export file from the customer who had requested the feature and planned to use it for testing.

Meanwhile, the team was building and testing the feature using a data export file from a test version of the customer's system.

Unfortunately for the team, the customer's data was real. And it was a mess.

Most of the data had come from sloppy manual entry. Fields which should have been the same were formatted differently, there was little consistency on the format of dates, some fields that should have been populated were blank.

Garbage in, garbage out. It's the type of thing that bites you as a software developer a few times before you build up your defenses.

The team cleaned up the problems and ended up completing the feature a day early, but not after feeling like they got burned by this "bad customer data."

During the retrospective I asked the team what would have happened had they used the customer's data file for their testing.

"We would have caught it right away!" said a member of the team.

"I would have never asked for acceptance if I knew there would be a problem," said another.

Even the product owner felt a little bad.

"Sorry, but I didn't even think to share this with you guys," she said.

A process for acceptance

Effective planning defines how the work will be evaluated and what it takes to be considered acceptable by the business.

There should be a defined process for evaluating and accepting work when the team says it's ready. That means understanding who will have the final say in accepting the work, what they'll be doing, and how they plan to make that decision.

Creating a clear and shared understanding of how the work will be evaluated means we know when it's ready for review. Knowing what it takes to be ready for review allows us to work backwards and create a complete plan. A complete plan means our work has the best chance of getting accepted during the first review.

A place for acceptance

The work should be available for a review in an environment that best replicates what our customer or end

user will see once the work is in production. A demo of the work on a developer's laptop might be good for quick feedback, but it's not a replacement for a thorough review. The reasons (excuses) why we can't review the work in a realistic environment are often hints towards deficiencies in the system or our processes.

Planning to review the work in a realistic production environment means the team is forced to plan for migrating data, deployment challenges, realistic sample data, and more.

During planning I like to ask myself the following:

- Will this work require deviating from the standard process for evaluating work?
- Will special tools or data be required to evaluate the work and consider it done?
- Will other people need to be consulted as part of the process of evaluating the completeness of work?
- What assumptions about the work are being made by the person doing the evaluating? What assumptions am I making about how it will be evaluated?

Define what's needed

"What do you mean we need approval? From who?!" said Jeff.

Effective planning accounts for all the resources required to complete the work.

Jeff was the normally calm developer who had been working on a big overhaul of the learning management system for weeks. He was starting to show signs of breaking.

“The data needs to be encrypted at rest, and the database we’re using doesn’t do that, so I need to get approval for the more expensive server,” said Dan, the ops engineer, as he leaned on the cubicle wall.

“How long is that going to take?” asked Jeff.

“I should be able to turn it around in a week or so, so like next Thursday,” replied Dan.

Exasperated, Jeff slunk down in his chair. “Next Thursday it is then,” he muttered.

“We better let product know, they wanted this shipped tomorrow,” Alex, another developer, suggested.

Dependent work

Effective planning accounts for all the resources required to complete the work. One of these resources is work that other teams are doing and on which your team depends. This may take the form of another service and an API contract or simply a design decision waiting on the results of a usability test. Mitigate the risk of dependent work by waiting to start, creating a “plan b” in the event something goes wrong, or agreeing to a simpler solution that will be iterated on in future work.

For example, if the mobile team is waiting on the backend team to expose a new API, they can:

1. Delay the work until later
2. Write their implementation against a pre-determined API contract
3. Stub out the backend work with fake data, to be replaced later

Third-party vendors

Most pragmatic teams recognize when a component of the system isn't part of their core business. There are myriad of services that fill the gaps in the systems we're building. Sending SMS messages, geocoding lookups, and video encoding are all examples of services we'd rather not build ourselves.

However, when we identify that a third-party vendor is required for upcoming work we can't always just sign up and start using the service. Before the work starts, we need to secure approval for using the service, deal with payment or procurement, and understand how the service works. If the team can't move fast enough to answer these questions during planning, a revised definition of done will be needed to account for the future connection to these outside vendors.

Account for team capacity

If you've ever been surprised by a Monday holiday, or forgot that half the team was attending a mandatory training, you've probably forgotten to account for the team's capacity during planning. It's easy to assume that everyone is working a full day for the entire iteration, but when that's not the case, the missing time has a bigger impact than we expect.

I encourage teams to start planning by reviewing the calendar and making note of holidays, time off, meetings, and other absences. If the person on the team who really knows the payments system is going to be in the Caribbean next sprint, you might think twice about taking on work to enhance recurring billing. Understand who will be present

for the work being planned and adjust the specific items and quantity of work accordingly.

During planning I like to ask myself the following:

- Is this work dependent on other teams or individuals? Will they be done with their work when I need them to be?
- Is there the possibility that the implementation choice I make will require new technology to be deployed?
- Is there a dependency on a third-party vendor or service? Will it need to be approved?
- Is there anything about this work that would trigger a review from another team or individual? (e.g. a security, regulatory, or legal review)
- Is the team planning or expecting to be unavailable in the coming days or weeks?

Define a shared end state

It was about halfway through the two-week sprint. The work had been progressing smoothly and the team was ready to get some early feedback on the new reports they had been building. The product owner joined us in the team room and everyone huddled around one of the workstations.

After the quick demo, it was obvious there was some confusion.

Effective planning results in a shared understanding of the end state of the work.

“Okay, so the report looks great, it’s exactly what we talked about. But, I’m not sure how you got there,” explained Sam, the product owner, “can you do it again?”

“Sure! I’ll start over from the dashboard,” replied Shira enthusiastically.

Shira ran through the demo again, this time more slowly. “Okay, I’m on the dashboard, then I click the reports link, and… give it a second… and now I’m viewing the reports,” she said looking at Sam.

“That! Right there, where did we just go?” asked the Sam. He was clearly confused about a transition despite the slower demo.

“I think I see the problem,” another developer, Darrell, interjected before expanding, “it looks like things are jumping around because we’re not *technically* on the dashboard anymore, we’re now looking at the new reporting service.”

“Um, okay, so the reports aren’t on the dashboard?” said Sam, now even more confused.

Shira jumped in. “You get to them *from* the dashboard,” adding, “but they’re not *on* the dashboard.”

Sensing further confusion, I asked Sam to clarify his expectations. “It sounds like you’re confused about how the end-user will access the reports and what is happening with the redirection and new address in the browser. Is that right?”

“Yes, exactly. I don’t get why the browser is redirecting so many times and the page seems to be flashing or whatever. It’s just weird,” Sam said.

We spent another fifteen minutes getting into the details of this new reporting service that wasn’t exactly part of the dashboard, but was built to look similar. The redirects and flashing observed by Sam were a result of the

authentication token being passed around between the dashboard and this new reporting service.

In the end, Sam begrudgingly accepted the work. A few months later the team decided to move the reports back into the dashboard codebase and the reporting service was decommissioned.

Tell a story through the system

Effective planning results in a shared understanding of the end state of the work. This end state is a combination of new or modified behaviors, the user's flow through the system, the look and feel, the expected side effects, and how it will be used. It's an aggregation of our expectations and assumptions.

I like to ensure that everyone understands the desired end state by pretending like I'm telling a story about the work. A story like "first, I log in, and then I navigate to my profile, there I see a new link, when I click it..." and so on. While not extremely detailed, this approach helps catch basic misunderstands or differences in expectations.

Describe and write scenarios

If the "tell a story" technique is working well, I recommend teams explore using Gherkin and BDD style scenarios to add more detail. Gherkin is a simple domain-specific language that describes examples of system behavior in a simple to write and understand format.

A simple example:

```
Given I am a new user
When I view my profile
Then I should be prompted to pick an avatar
```

Tooling for writing Gherkin and consuming the scenarios as tests exist in almost every language and many popular frameworks. But a warning, don't start with tests. If you're new to Gherkin, do not attempt to automate your behaviors as a test suite right away. Instead, use them first as a tool for improving understanding and communication on the team.

During planning I like to ask myself the following:

- How will a user find this new behavior? Use this new behavior? Deal with things going wrong?
- Is this like an existing feature? Can I use it as a guide?
- How do I assume this will work? What about other members of the team?
- What expectations exist about this work? (i.e. performance, usability, accessibility, design)

Summary

When we have defined what it means to be successful, we can plan to achieve that success. If we rush through planning and leave with a vague understanding of what it means to be done, we can only hope to get lucky. A strong definition of done is essential to effective planning.

Effective planning defines:

- The acceptable boundaries for each work items
- How the work will be evaluated and what it takes to be considered acceptable by the business

- The resources required to complete the work
- A shared understanding of the end state of the work