Why I Still Use Use Cases

Alistair Cockburn

XP pretty much banned use cases, replacing them with the similar sounding "user stories" (see [[A user story is to a use case as a gazelle is to a gazebo]]}, and as a result agile zealots have been happy to dump use cases in the trash (along with their project managers, estimates, plans, and architectures). Scrum did similar, using the "product backlog" instead of user stories. Yet as I go around projects, I keep running across organizations suffering from three particular, real, painful, and expensive problems:

1. User stories and backlog items don't give the designers a _context_ to work from – when is the user doing this, and what is the context of their operation, what is their larger goal at this moment?
2. User stories and backlog items don't give the project team any sense of "completeness" – what I keep finding is that the development team bids/estimates the projects at (e.g.) 270 story points, and then as soon as they start working, that number keeps increasing, seemingly without bound. The developers are depressed and the sponsors are equally depressed by this. How big is this project, really?
3. Related to completeness, user stories and backlog items don't provide a good-enough mechanism for _looking ahead_ at the difficulty of upcoming work (In principle they could, just in practice they don't) – I keep hearing this complaint, "We asked our customer (product owner) a question and she/he took 2 weeks to get us an answer. We must have the wrong person in this role." No, they don't have the wrong person, they have a broken process – certain types of questions take a long time to research, as the various departments and user groups work out what is the correct, balanced answer for the whole lot of them. Staring at the set of extension conditions in a use case lets the analysts suss out which ones will be easy and which will be difficult, and to stage their research accordingly. User stories and backlog items are not set out in that granularity early enough on for that assessment - the extension conditions are usually detected mid-sprint, when it is too late.

Use cases are, indeed, heavier and more difficult than either user stories or backlog items, but they bring value for that extra weight. As not-Einstein said: "Make things as simple as possible, but no simpler." (The attribution to Einstein has been debunked, it seems.)

In particular, use cases fix those three problems.

I've been testing this out for the last 3 years – I walk in and ask, "On your agile project(s), do you by any chance suffer from any of these three things?" ... and then list the three ... Much stronger than even I expect, there _hasn't been a single organization I asked these of that hasn't said, "Oh, Yes, and How!"

In other words, naïve use of user stories and backlog items is causing very real, very expensive damage to companies. Expensive is bad.

The most recent was an executive who was telling about having "delivered" a Scrum project, but it was discovered at acceptance time that there were a whole series of recipients of the system whose needs hadn't been identified at all. This company is now faced with writing what may amount to a fixed-price contract because the recipients aren't sure they trust the company to deliver _complete_ and useful software.

Very expensive, very bad. The time spent writing the use cases and doing a more thorough requirements investigation would have paid off handsomely, in this case.

Here 5 reasons why I still write use cases ([[Structuring use cases with goals|my kind]] of course :-)

1. The list of goal names provides executives with the shortest summary of what the system will contribute to the business and the users. It also provides a project planning skeleton, to be used to build initial priorities, estimates, team allocation and timing. It is the first part of the _completeness_ question.
2. The main success scenario of each use case provides everyone involved with an agreement as to what the system will basically do, also, sometimes more importantly, what it _will not do_. It provides the _context_ for each specific line item requirement, a context that is very hard to get anywhere else.
3. The extension conditions of each use case provide the requirements analysts a framework for investigating all the little, niggling things that somehow take up 80% of the development time and budget. It provides a _look ahead_ mechanism, so the customer / product owner / business analyst can spot issues that are likely to take a long time to get answers for. These issues can and should then be put ahead of the schedule, so that the answers can be ready when the development team gets around to working on them. The use case extension conditions are the second part of the _completeness_ question.
4. The use case extension scenario fragments provide answers to the many detailed, often tricky business questions progammers ask: "What are we supposed to do in this case?" (which is normally answered by, "I don't know, I've never thought about that case.") In other words, it is a thinking / documentation framework that matches the _if...then...else_ statement that helps the programmers think through issues. Except it is done at investigation time, not programming time.
5. The full use case set shows that the investigators have throught through every user's needs, every goal they have with respect to the system, and every business variant involved. It is the final part of the _completeness_ question. (And yes, I did indeed sit down and walk through 240 use cases with a client, at the end of which, I turned to her and asked: "And is that everything?" She said, Yes, and we built that, delivered it, got paid for it, and it is still in use ten years later.)

The are several sticky parts for people using use cases as I describe:

- It is really easy to think all the use cases have to be written up front, when in fact the writing should be staged over the life of the project. The "thinking' part here is working out how much should be written up front to get the project estimate into a safe place, and which parts can be deferred to just-in-time investigation.
- These days, iteration/sprint lengths are so short that it is not practical to implement an entire use case in just one of them. That means additional work is needed, to create user stories or backlog items for each use case, track that each one get developed, and ensure that the complete set of user stories or backlog items do indeed deliver the subset of the use cases needed for the particular release.
- Writing good use cases (or any other requirements) requires thinking, communicating, and thinking again. It is much easier just to write user-story tags on index cards and let the project blow up later.

If you don't want to think, find a new profession. Software development requires thinking, at all levels.

p.p.s. I know I've said these things many times over the last few years, I've presented them, and I thought I had written it up. But it seems not. I'm glad to have the chance to repair that omission.