Speaker 1: …Implement it or you know do your users find it easy to use these artifacts or…

Speaker 2: The API is really well documented there is not an API that is not documented and even the god class I talked about [Indiscernible: 00:15] to all the implementations has documentation of 1,500 lines of actually the whole library so yeah and it's often used but there are some implementations or quirks of the library that are really unused and there is documentation for them but no one actually reads it so…

Speaker 1: Well that's the case actually with most documentation.

so do you guys actually keep changing the API?

Speaker 3: Can I ask a question what do you mean exactly by well documented what does well documented API mean?

Speaker 2: So the documentation not only says it should… this method does A and returns B, but it also provides examples of how you would use it and even multiple examples…

There is for example a method and it has three examples of how you would use it and in what situation and also explains in this situation you should prefer not to use the API

so not only plain Java doc it does this it also explains how to use it and when not to so I think that's well documented.

Speaker 1: Okay, okay sure so do you guys still change the API? do you add new features, remove features, deprecate old features?

Speaker 2: There were so Mockito 2.0 was in beta for a really long time which has breaking changes in the sense that matching has changed so previously you would define a mock and then you would define in this situation I return this and the thing in this situation was a matcher so it would take some arguments and match on that and if it returned true it would invoke the return thing

However, that depended on the library that was… well unused, no longer maintained, which is four years old now. So it was dropped, which meant that the argument type changed and we now use a different library which is actually maintained and a lot better. But that's a breaking change obviously, so that's one of the bigger changes.

Speaker 1: There should be more breaking changes to the API, right? So, the API code broke as you tried to transition from one library to another. But did the public interface also break? Did the interface also changed, in such a way that maybe client code break?

Speaker 2: Ah there were some removed methods, there were not a lot, but there were some. Especially there was initially an annotation for mocks, but it was weirdly written… and then we decided to replace it with a better one; so the old annotation was deprecated (but still there) and the new annotation you had to use which had different configuration. So not a lot of breaking changes.

Speaker 1: How do you decide to remove an old feature or deprecate an old feature and go to a new one? what really goes behind the scenes?

Speaker 2: Usually it is a new feature that we want to add and it's incompatible with previous features.

Or a system is redesigned and then we decide “okay it’s not capable of doing this, but we don’t really care this was a feature we implemented, but regret later” so then we are fine with deprecating it and removing it later. So… what else… well mostly it.

Speaker 1: Who really makes these decisions in to the redesign? Do you guys get together as a team? Do you take your clients into account, maybe? or do you… who do you talk to really to make the new design?

Speaker 2: It’s a discussion of the core team but it’s a public discussion. Usually there is an issue created tracking the change and then feedback is asked. Then after a certain amount of time they say “okay feedback is processed; we continue or we don’t.”

Speaker 1: And typically this feedback is from whom?

Speaker 2: From the Mockito users so…

Speaker 1: Users as well.

Speaker 2: Yeah.

Speaker 1: Okay.

Speaker 2: Yeah it’s a public issue and the users of the Mockito API usually comment on these issues.

Speaker 1: Okay.

Speaker 3: And normally how many users are involved in this?

Speaker 2: Not a lot, I would say. A couple of tens of developers… it’s just a small subgroup you see providing this feedback. Rarely a new developer joins the discussion.

Speaker 3: And do you know these users? Are there current users that participate in discussions? Or how do they end up writing something or answering in that particular issue?

Speaker 2: You mean they are well known users.

Speaker 3: Hmm, hmm.

Speaker 2: Yes, there are some users that almost always comment: “I like this or I don’t like it” and they are heavy users of the library. They use it almost every day and have big knowledge of the whole codebase because they use a lot of features. They mostly requested the new features if…

Speaker 1: Do they ever contribute to the library themselves?

Speaker 2: Sometimes yes, yes. Not a lot. The development is currently kind of still because the core team is… they have a real job at their companies, so currently they don’t have a lot of time. Usually during vacation you see a big increase in activity and so then you also see the changes.

Speaker 1: So you mentioned deprecation as well, so you guys actually use it. What do you really think about deprecation?

Speaker 2: I think you postpone the problem; because if you deprecate something you say…

If you say we deprecated, you say: “okay it’s either badly written or it’s unused” or you say: “we can’t continue supporting this” but deprecation says we will do this later. Usually this means: “oh crap we are on new major version a lot of stuff suddenly breaks.”

So it’s allowing you to postpone the inevitable I think.

Speaker 1: But do you guys actually use it as it was intended? Do you actually delete your features, later? for instance Java hasn’t done this since Java 1.1, so do you guys actually clean it up?

Speaker 2: There was actually last week a pull-request has been merged that removed old deprecated features, so the external contributor went through the code looked for all deprecated annotations and removed all those methods and classes. So, they are actually now removed. They were not previously… I am not sure if the core team saw that these still exist.

Speaker 1: But do you know how long they were in there maybe?

Speaker 2: A really long time, yes.

Speaker 1: So a couple of years, versions or…

Speaker 2: Yes, well not major versions, but timewise a long time, yeah.

Speaker 1: Do you think that this would have an impact on client upgrade behavior? Do you think that now that you’ve removed these deprecated methods, people may not actually go to a new version because it’ll actually break their code?

Speaker 2: I think the transition to the new code is easy enough.

Speaker 1: Why, why would you say easy?

Speaker 2: Because the features removed were implemented in a different way, so that the API calls were changed, but the functionality per se was not really changed. So the type of the methods… So with the matchers we use a different library; now transitioning from one library to the other is relatively simple, but it is something you have to go through and it’s a breaking change because you have to change all your matching algorithm, but the changes per se are not that hard to do.

Speaker 1: And is there anything that maybe you as an API developer could do to lessen the burden of the change?

Speaker 2: Yeah, so an upgrade guide will be written to guide the developers transitioning from version 1 to version 2.

Speaker 1: This upgrade guide would contain what?

Speaker 2: “We deprecated this feature use this now” or “it’s completely removed” or “you know you have to change the matching from this to that, so this class has been… is equivalent to that class in that library.”

Speaker 1: And but do you think that that’s going to be enough? Do you think that when clients actually encounter a deprecated feature, they will really make the transition? Or do you think they will just continue using them and would refuse to use the latest major version?

Speaker 2: I think whenever we deprecate something, they will not change it until they have to switch to the new version where it is removed. If they see "okay this is removed now, we are investigating" if it's easy enough, they will do it, but if it's not easy and it's really a lot of time, they will postpone it until it's really necessary to upgrade to a new major version.

Speaker 1: Would you prefer that they were always on the latest version? Or is it okay for you as an API developer that no one really seems to be using the latest version but prefers hanging back?

Speaker 2: Well there is the problem that… so the codebase is currently for Java 5, because Android (which is one of the big users of Mockito) requires this. and yeah they mostly use 1, I think, because it's mostly compatible with their codebase, which is, I think, fine, but if they transition to a newer Java version they really should upgrade. So, it's not really a problem because mocking can still be done in version 1 (works fine), but it's better implemented in version 2.

Speaker 1: And is there anything you guys can do as developers to incentivize people to move?

Speaker 2: I think the best way would be to introduce new features, or speed up, or just performance increases in the new version, or compatibility with the new Java version. So the plan is to make it integrate more fluently with Java 8 which is really a big change. If you then transition your codebase to Java 8 you really want to use that new version, it's a lot nicer and cleaner so…

Speaker 1: But do you think if you just kept the deprecated features in there, this would also act as an incentive? Because then no one's code would break and they could continue going on and on and on.

Speaker 2: So remain using the deprecated features. I think that's only the case if it was implemented better in the later version; if there was really a bug or design flaw with a deprecated feature you really should remove it to take away the possibility to use that wrongly implemented feature. Only if the new version has benefits without any changes… yeah you could keep it, but no I would remove it because you really say don't use this feature it's old, it's bad, or it has flaws, or just deprecate it.

Speaker 1: So then do you as an API developer think it needs to be made more explicit in Java about the dangers of using a deprecated feature? Maybe your documentation itself on its own might not be enough.

Speaker 2: Yeah, maybe you could add some sort of flag saying "yes it's fine I am using deprecated features." The only fear I have is that if you introduce such mechanics, probably everyone will do it because they will encounter some sort form of deprecation. So I am not sure what the effect is.

Yeah, if you keep those deprecated features, like in Java it's really easy to bump to a later version. So the level of effort required to get to the new version is really low and therefore people are more willing to use that new feature. So it has two sides yeah.

Speaker 1: Do you have any questions?

Speaker 3: Yeah, so I have a couple of questions. The first one is: you say that we deprecate features that we know people don't really use, you said something like this…

Speaker 2: Hmm, hmm.

Speaker 3: I am curious to know how you know which kind of features people use.

Speaker 2: There is no definitive proof, but if there's never an issue created for that specific feature, for a really long time, and you don't know any public project that actually uses the feature, you can be quite certain but you can never be completely certain.

Speaker 3: What do you mean public projects so you track the projects that use your API.

Speaker 2: Hmm, hmm.

Speaker 3: How?

Speaker 2: Via plain search, you can track some public… for example Android is open source you can track you can search the code and search

Speaker 3: What are you looking for?

Speaker 2: Mostly if that specific feature is actually used in the whole project. So you do a plain text search and you see.

Speaker 3: Okay and now I have another question: What do you think about deprecation as a feature itself of the Java language?

Speaker 2: Well, yeah, I think it's an easy way out for developers of an API, because yeah it's really easy to edit and notify your users, but you do not actually remove the whole feature. So you are stuck with the sense that a user can be willing to keep using it and not be incentivized to actually stop using it. So there's no negative effect for using it other than your IDE saying "it's deprecated". That's the only thing. So yeah it's still the best communication method as of today, but I think there are better ways, but I am not sure how these would work.

Speaker 3: Do you have some in mind?

Speaker 2: No.

Speaker 3: There should be a better way.

Speaker 2: Yeah, there should be. But this is fine for now, there are…

Speaker 3: Why is it…

Speaker 2: Well, because you have these features you can quickly move to a newer version and enjoy new features, for example, and you can drop the deprecated features whenever you want. So, in that sense, it is really easier to keep up with the pace and the new versions… and that's more secure, and probably also fast. So I think that's a good thing, but you are never incentivized to actually stop using it, so in that sense as a user.

Speaker 3: As an API developer, do you want your users to stop using those features? Why do you put deprecation in the first place?

Speaker 2: So there are two reasons: (1) One is "it's plain wrong" or there's just a flaw; which means "don't use this, because we implemented it badly" and there's the other one (2) where you say "okay it's unused and it's too costly to maintain" and, in that sense, you want to get rid of it because the benefits do not outweigh the costs.

Speaker 3: You said something about it's a way of communication with the users… What do you want to communicate them?

Speaker 2: That the feature should be removed, but you are now able to anticipate earlier, which usually means later because you postpone it until you really have to do so. It's nice for you to say "hey be aware we will remove this feature later, just so you know" which is nice thing my opinion.

Speaker 3: Can I ask you a last question?

Speaker 2: Yeah.

Speaker 3: So as a user of an API what do you think when you see a deprecated feature and how do you react to it?

Speaker 2: I would first wonder why is this removed and I would assume that…

Speaker 3: So I am talking about as a user you see a feature being deprecated not removed.

Speaker 2: Yeah so why was this feature… is the intention to remove it? so why has this been deprecated? so I would assume that the documentation shows "this is the reason we are not supporting this feature anymore" and then I would continue developing and make an issue: "hey this is deprecated we should fix it later" and give the low priority and then when we have time we fix it. or link it to the issue "we should update this library" and say "okay then we first have to drop this and that feature so."

Speaker 3: So as a user what do you expect as the right documentation of a deprecated method?

Speaker 2: To explain why… So why is this being removed. What is the reason was it a design flaw? Or is there a better way? How easy is it to change it? So, if there was a new implementation (which is the replacement), what is it? where can I find it? And, how easy it is to switch to that? These sort of questions mostly the 'why'.

Speaker 1: And would you consider a good behavior as replacing it? So would you say actually replacing the call as recommended by the API developer themselves? Or would you just create your own in house replacement? or use another API even? Or…

Speaker 2: Yeah if the developer says hey we removed this but you can use this I would use…

Speaker 1: You would follow the…

Speaker 2: Yeah I would follow the suggestion because probably they know better what a good replacement is and if you have to figure that out yourself you're probably going to make mistakes so why not use the suggestion of the one who developed the feature that knows the best to replace it.

Speaker 1: Okay I think…

Speaker 3: That's it.

Speaker 2: That's it yeah.

Speaker 1: Thanks very much for your time.