Speaker 1: Alright so good morning first off so thank you for willing to do this can you… I mean I know what your unit does but can you in your own words describe what JUnit does?

Speaker 2: Yeah sure, well JUnit is pretty much a testing framework for Java so I think it's one of the oldest open source projects actually that was started by [Indiscernible: 00:22] and has evolved since then

so there was Unit 3 which was the first version that was publically used and Unit 4 has been around for I don't know ten fifteen years and at the moment we are working on the next major version that tries to solve some of the maintenance problems and limitations of JUnit four so I think we will get back to those during the call yeah

JUnit is really used by a lot of people so even other testing frameworks have their own JUnit runner so they can easily integrate into IDEs and build tools that just support JUnit and [Indiscernible: 1:03] special testing framework and yeah so IDEs use JUnit [Indiscernible: 1:08] testing frameworks use JUnit and of course people that just want to add tests use JUnit.

Speaker 1: Do you have any idea about the popularity of JUnit? Do you guys do any kind of analysis internally?

Speaker 2: Not really we have some statistics from Maven Central so the number of downloads I don't have them for 2015 but we did an analysis for 2014 and it was like forty-three million downloads in this year so yeah quite a lot.

Speaker 1: Okay so how long have you been working on JUnit?

Speaker 2: Yeah well I didn't write the original JUnit 4 so I became a maintainer about four or five years ago.

Speaker 3: Hmm, hmm okay and do you remember anything about the evolution of JUnit some very difficult times or decisions you had to take in those four years about evolving JUnit changing things there.

Speaker 2: Yeah so in JUnit four there were a number of things that were from the start I think a little well-meant but didn't work out in the end.

So for example we had some features that were considered experimental in the beginning so the decision was taken to have an experimental package and put the class into this experimental package. Right, so now what happens if this feature that was once experimental becomes something that we wanted to declare as stable, right? So what do we do yeah if you change the package name of the classes in this package everyone else [Indiscernible: 2:42] that's something we can't do. You could maybe deprecate the old class in the experimental package and then well copy the classes into a different package, right? So and tell everyone please use the new classes but at some point we would like to delete the old package, right? and that's something that's pretty much impossible for a library like JUnit that is this popular because someone will always complain and you might break some IDE integration or whatever.

So it's really a hard problem for JUnit to [Indiscernible: 3:18] features that are not meant to be stable yet but we would like to evolve over time because that's what just this JUnit jar that's also one thing that we try to address with JUnit five but we have more modular architecture so that we actually know which APIs are used by build tools and IDEs…

Speaker 3: So basically in this way you split it in multiple jars and then [Indiscernible: 3:46] you have a better idea of…

Speaker 2: Exactly.

Speaker 3: Okay I see.

Speaker 2: Yeah we try to talk to the IDE vendors and build tool creators directly so we had a kick off for JUnit five where we invited them [Indiscernible: 3:59] the things that they want from JUnit and what they need

yeah same thing with the experimental package also happened with packages that are declared internal so they have internal in their name yeah and at some point you see okay this feature was internal but I would like to make it public now same thing so using packages for stuff like that for declaring stuff internal or experimental is something that has not worked out, for example so that we are trying a different approach for JUnit 5 there yeah so that's one thing.

Speaker 3: And so you said… you talked about moving something from experimental to non-experimental to stable so have you already done so like what were the approaches and did they work or…

Speaker 2: Well in the end… yeah there are features like the [Indiscernible: 4:54] runner that are still in the experimental package so but no one pays attention to the package anyway so in the end we just decided to leave it there and say okay yeah you can use this it's in a jar

we will not break backwards compatibility there because it will result in an outcry by users and then so that's not what we want to do so
this has really slowed down the evolution of JUnit that you can't really do anything with the code. You can't even…

we had problems we wanted to [Indiscernible: 5:31] some old naming conventions for instance variables like f underscore something and we said okay that's just not what you do nowadays you don't have such a prefix kind of notation for variables so we wanted to rename private [Indiscernible: 5:46] and we did that [Indiscernible: 5:50] and all that because they used reflection to access this field [Indiscernible: 5:55] even though there was a getter now and originally the getter wasn't there I guess and then they just used reflection to access it and of course that works and yeah so this is really stuff that is really annoying because we have to use exceptions for assertions that fail right so and they derive from assertion header and they should be serializable so if you change something internally you have to take good care that you don't break serialization so it's actually what you want to have is that it's backwards compatible also regarding serialization so if you have one serialized object and [Indiscernible: 6:35] de serialize it, it would be not the same output at least you should take care that you increment the [Indiscernible: 6:45] in Java right so if in case it's not compatible so yeah there was also some strange parts we had there.

So a lot of things that really slowed us down which eventually right now led to this JUnit 5 effort that we are in the middle of basically rewriting the JUnit in a way that is backwards compatible so we can run old tests that have been written using JUnit 4, but at the same time we are aiming at having a modular architecture with different jars to be used by different people.

So there is one jar that just contains the API that the people who want to write tests should use and there's one jar that just… two or three actually that are used by the IDEs and build tools to actually say okay we want to execute these tests and yeah we have a test engine abstraction that allows us to run JUnit 4 and JUnit 5 tests and even JUnit three tests in a single test run but JUnit 5 does not know anything about JUnit 4. So previously the JUnit 4 jar actually contained the JUnit 3 classes so [Indiscernible: 7:58]

Speaker 1: So how are you actually doing the backwards compatibility to JUnit 4, so are you still keeping the features alive or are you just [Indiscernible: 8:06] but still being able to run them, how are you really doing this backwards compatibility?

Speaker 2: Well we have introduced some abstraction of it's called test engine so this abstraction is… JUnit 4 has been provided a test engine implementation that's just a small adaptor to the JUnit 4 jar basically right so we support all the features that JUnit 4 does but have added some more features and have [Indiscernible: 8:30] them a different test engine [Indiscernible: 8:31] JUnit five test engine and that uses the JUnit five APIs and stuff like that and you have pretty much… it's pretty much a new testing framework so all the test annotation for example we have a new test annotation which is in a different package, it's in a different jar and JUnit 5 tests use this test annotation. So it's different it's not the same one, it doesn't have any parameters as the old one does but we've completely separated that and this abstraction that we have created allows us… I mean you could write a test [Indiscernible: 9:07] test engine if you want [Indiscernible: 9:09] test engine implementation so but the IDEs don't have [Indiscernible: 9:15] about the test engine implementations they just [Indiscernible: 9:18] test engine APIs so basically what we wanted to create was something that the IDEs and build tools can use to execute any test using any testing framework and not having to write custom code for it and sometimes it might make sense for them to do that to improve the [Indiscernible: 9:38] for this testing framework but you know for basic support you don't have to do that [Indiscernible: 9:44] so that's one of the goals for JUnit 5.

Speaker 1: Okay so you guys evolved this API and how much do you think is actually going to be the upgrade impact? so how hard do you think it's going to be for a client to go from JUnit 4 to JUnit 5?

Speaker 2: Yeah well one thing we know pretty much is that no one will migrate his old test [Indiscernible: 10:04] JUnit 4 tests a couple of thousand tests or something to JUnit 5 just like that right.

So that's why we take special care that it will have this test engine abstraction and to be able to run JUnit 4 and JUnit 5 tests in one test run so you don't have to do to test runs and then aggregate the results you can just create one test run and say [Indiscernible: 10:27] and the engine abstraction will then figure out okay this class seems to be a JUnit 4 class this class seems to be a JUnit 5 class and run all of them right. So you can actually keep all JUnit 4 tests and try out JUnit 5 for new tests and if you like it you might eventually well whenever you touch a test migrate that test to JUnit 5 or something, but we assure that JUnit 4 will live for quite some time and I mean it's a really stable testing framework so there's nothing that really will be a big disadvantage of having both I think.

So we think new projects of course would start using JUnit 5 but old projects yeah we have to find out so it’s really hard to get feedback on that kind of thing but if you only have an Alpha version out or even a milestone version because people tend to wait for the final release and then complain.

Speaker 1: Earlier you mentioned that you guys deprecate maybe experimental features to make them more concrete and move the classes into a non-experimental package. Is that the only way you use deprecation? or are there other ways in which you guys deprecate features.

Speaker 2: Well we try to deprecate classes.

For example, the JUnit 4 jar contained Hamcrest core, we tried to get rid of that: […] Hamcrest wasn’t ready at that time when JUnit was released (Junit 4.7, I think). So, there were some classes like TypeSafe[…], for example, that are now in Hamcrest core, but previously they weren’t there, so we have their old implementations, so we deprecated those classes. We put the deprecated annotation and we say “okay, at some point in the future this will be deleted.”

At the same time, we would like to maintain backwards compatibility as much as possible and not break people, so yeah I think those classes will never be deleted. They will always stay there, I guess.

It’s really hard and we will hopefully find a way to do it in JUnit 5 (to delete deprecated code), but in JUnit 4 it’s really, really hard to do that, because the benefit does not outweigh its costs. […] If a couple of thousand people cannot run their tests tomorrow, or JUnit contains one more deprecated class…

You can just say okay I want to maintain the high code quality, then, of course, you would delete this class… but yeah it is a tradeoff I guess and at that point I think the decision has always been taken to leave this feature in even though it is deprecated and has been deprecated for a long time.

Speaker 3: Can you briefly reiterate why you decide to deprecate a certain feature and why do you… so since you are already basically, if I understand, that these features will stay there, until the end of the time.

Speaker 2: It’s not so much in features, I think it’s more in code that we’ve moved elsewhere.

Like an exception class that has been moved to a different package now. It is now deprecated because people like to use the other one. But yeah I see your point I mean you could say: “okay, why do you deprecate things in the first place if you never delete them?”

Speaker 3: I would like to understand what you expect from the clients sending this sort of message right.

Speaker 2: […] If you move a class to a different package for example and deprecate the old one, then, of course, we would like them to use a new class.

Instead features I don’t think many features have been deprecated. […] There are some features that are in the experimental packages, but I don’t think we haven’t deprecated anything there… nothing I can remember at the moment.

Speaker 1: Has there been a case where you guys have actually deprecated something and a lot of clients have actually complained to you.

Speaker 2: Well we had some beta releases, like I said, where we actually changed some internal code and broke someone so.

Speaker 1: No but specifically with deprecation I mean did you mark something as deprecated and then had a complaint.

Speaker 2: Let me think about that, I don't remember anything at the moment no.

Speaker 3: And in your opinion and experience… what's your take on deprecation in general? You find it useful or what do you think about deprecation as a feature itself?

Speaker 2: For JUnit it hasn't really worked out to deprecate things and then get rid of them so… might work for some other libraries… I don't know what we did wrong…

I think for JUnit 5 we are now trying really to do it the other way around: to declare things as experimental or internal first, and then say "okay, we can use this as experimental but we will really delete it at some point or change it if we think it should be changed" and not use deprecation that much.

Yeah but it's really hard to do deprecation in such a library. It's an open source project and then people use it a lot of people use it and, as I said, it doesn't really work for Junit. That's my experience.

Speaker 1: So do you think these new annotations that you guys are introducing, which I think was one of them was the @api annotation, which you want to mark something as experimental or beta. Do you think that these will have a better or higher impact on clients to actually transition away from these features to a new feature, as opposed to deprecation?

Speaker 2: Well we have to find out I guess…

So it's an experiment in itself, right… At least we don't have experimental in the package name or internal anymore, because that doesn't work.

yeah let's assume we have a feature that's been tagged experimental and we want to change that now. And then someone says: "but all of my ten thousand tests will then break" and you don't have an easy solution for him to migrate to the new feature… then that's probably a point where we will spend a lot of time discussing "okay maybe we should leave this feature then in", or something like that. It's a really a community driven project and such complaints, if they are well stated in time (before the release), I think will be considered and taken seriously.

Speaker 1: But don't you think that when you have these experimental annotations […] it doesn't really show up in the IDE, because when you use a deprecated feature it comes with a strikethrough right.

So it's at least made in some way explicit to the developer that you know this feature might you shouldn't use this feature maybe use the replacement. Do you think having these annotations would you know solve that? Because here there is no IDE support or at least no native IDE support.

Speaker 2: Yeah so we don't know if they will solve anything right, because we have tried. It's our best help I guess…

I mean of course you could mark everything as deprecated, that's not stable or not maintained, but that sends out a message, I think. You do that because there is IDE support for @deprecated, right? But a new feature that you would like people to try out and you mark it as deprecated, it's kind of strange.

Speaker 1: In Java 9 they are introducing an upgrade on the deprecated annotation where you can actually give proper reasons in there about experimental or removed because unsafe… so there are multiple enums that are coming in there. Would that be a viable alternative.

Speaker 2: Maybe I haven't looked at that yet, but yeah it still says deprecated right so…

Speaker 1: Yeah so the issue is with the deprecated.

Speaker 2: Well this is my personal opinion, so I haven't discussed this with the rest of the team. Maybe they think of it differently, but yeah I think it's hard.

What we want to do at least with the API annotation is that we have some kind of automated check, so that we don't break backwards compatibility, right. If we rename something internal, then it should be fine but we really would like the build to fail, for example, if we have a stable API and we rename this method and this should really [Indiscernible: 19:08] so something… so that there's also one thing that we would like the API annotation that will be introduced to give us…

Speaker 3: So do I understand correctly that your plan is to not use deprecation in 5 in general?

Speaker 2: We have one inner constant deprecated for the API annotation, so you can say @apiDeprecated and then it would make sense probably to also use the deprecated annotation, because, as you mentioned, there is IDE support for that.

We are really in the early stage of rewriting the whole thing so it's kind of hard to think so far in to the future… do we want to deprecate things? Of course the hope is that we don't have to do this, but I am sure we will… yeah it's hard to foresee the future.

Speaker 3: Yeah but I am saying from your experience…

Speaker 2: Yeah it hasn't really worked out in JUnit 4, so I would say "okay, as I mentioned before we tried to do it the other way around, so we don't declare things as stable, but the thing should be stable and the rest is kind of yeah it's experimental or something in between" But yeah that's the approach we are taking I guess, but there might come some point in the future where we think "okay this feature is now solve differently"

I can imagine, for example, we have I don't know if you have looked at the new API and we have a lot of extension points people can supply their own extensions. There might come a time where some of the methods or some of the extension points change, so I think then it might there might be a point in time where we say: "okay this is now the new extension point which gives you more power to supply something but this old way still works kind of" then we might deprecate that thing, but if we ever delete it well I am not so sure about that.

Speaker 1: Okay so breaking changes quite hard, if I can really summarize what you said. I think introducing breaking changes is quite impossible in JUnit in general.

Speaker 2: Yeah.

Speaker 1: Okay, that's very different from most APIs, I guess. Can we maybe talk about so the versions that are being used of Junit? do you guys have any control over how people have upgraded behavior of versions for instance do you think that people stay back on older versions or do they go to newer versions more often.

Speaker 2: Well I haven't really looked in to the most recent statistics, so I think I could even send you the current statistics from Maven Central. What I remember from last year when I looked into it, it was really lots of people still using old versions, but I can't really tell you any percentages right now because I don't remember that exactly.

------ cleaned up to here ------

Speaker 1: As a developer yourself of JUnit would you prefer them to be on the newer versions? or let's say that you released your unified version and would you prefer that everyone who used JUnit 3, JUnit 4 just transition to JUnit 5 especially because it has backwards compatibility, or you are okay with them just hanging back with older versions?

Speaker 2: Well I think for the JUnit 4 kind of product line thing, we really took good care of maintaining backwards compatibility, so I think there's no reason to stay behind and use old versions.

If you have JUnit 4.5, for example, I don't see any reason why you shouldn't upgrade to JUnit 4.12, so that's something

I mean now with JUnit 5 it's a bit different because as I mentioned if you have lots of old tests JUnit 4 tests and you want to write the new test using JUnit 5 then you will have both on the classpath; and it's not a conflict, it's just different artefacts due to the modular architecture and this test engine mechanism. It's completely fine and I think then yeah we require JUnit 4.12, for the JUnit 4 test engine, that hooks into the JUnit 5, then people will use multiple versions at the same time actually because of the architecture.

In general I would say yeah I am of course fine when people want to use older versions but I don't understand that, because there have been some improvements and new features and I think people should look into them and use them to simplify their tests. So, as a JUnit developer, I would like them to use the most recent version.

Speaker 1: Do you do anything to incentivize this though? So do you maybe advertise new features do you do something to maybe push them to upgrade?

Speaker 2: Well yeah we do advertise new features. Originally I always step into this JUnit maintainer role, because I use JUnit in my company and they run a really old version of JUnit and I looked into the new versions and they had really cool new features I wanted to try out.

So I prepared some presentations to make colleagues know about the new features in Junit. Then I went to conferences and held talks there and tried to convince people that it makes sense to look in to new features and they can really improve their tests using them.

So we do some advertisement there. We also have a mailing list of course, where we do that but… yeah I think some people just don't care I mean and they are also I think a lot of software projects use JUnit and not all software projects are actively being worked on all the time, so I think some of those downloads maybe just come from old software projects that have been created I don’t know ten years ago and use some JUnit version and are just maintained sporadically… like some maintenance work on them and but it’s not the top priority for such software projects whenever they… to upgrade to new versions right…

So if you have those old software projects which you just touch from time to time really carefully then it is not the first thing you do is you upgrade all the software drivers to the newest version because then you might break something even though it’s JUnit the risk is really small. Yeah so I think that’s part of the reason people are still doing that and I don’t think this will change so…

Speaker 3: Okay so the last question so you said that you basically get information about the usage for clients from Maiden and you understand which version of JUnit basically is that correct. Is that enough information to have and decide what to do? Would you like to ideally have more information?

Speaker 2: Yeah sure it would be great to have more information. For example, in JUnit 4: Which one our people use? how many people use parameterized tests, how many people use series, how many people use rules which rules and all that stuff” yeah it would be really cool to have all that information.

Speaker 3: How would you use that?

Speaker 2: Well I think it would be good to know what focus our work on. So if nobody uses a certain feature, then it doesn’t make a lot of sense to spend a lot of time working on it. This is an open source project so of course most people work on the features they are interested in themselves the most, but we still have we have some roadmap, we think about what’s most important for people, and what should we work on and just for that it would be very useful to access this information.

So we have some parts of that information from open source projects, so we can actually they have some I forgot their names but there are some online platforms that try to look at GitHub repos and all that stuff, and give you some usage information on: how many people called this method, how many people used this class and stuff like that. But, I mean, that’s just open source projects, but just for JUnit I think it would really be interesting to see what people actually use also in enterprise kind of setups and stuff like that.

Speaker 3: And having that only from open source only would not be…

Speaker 2: It would be a start, better nothing… but I think it’s only a small fraction of the actual users.

Speaker 3: So you think that basically in open source setting they have a different way to use JUnit compared to company.

Speaker 2: This is just my assumption. I think open source projects tend to use more of the latest features of things. They want to stay up to date with all their dependencies and then spend some of their work doing that updating to the newest JUnit version for example, trying out new features, writing parameterized tests, so spending really a large amount of their time having good tests… I think there might be a difference to other software projects that are not that visible right so you don’t publish your source code.

Speaker 3: Imagine if you had a possibility to see for example how in the project on GitHub use Junit?

Speaker 2: Yeah I think it would be useful yes. I think it’s just a small fraction of projects so you shouldn’t base all your decisions on that then, but I think we would consider it.