Software engineering radio episode forty six refactoring pop one in today's episode. I'm going to talk with Martina but so MARTIN Do you want to say some words about yourself. Thank you for being on the SO. So my name is Martin Lippard and I'm working as a consultant coach for I.T. edgel it's a small consulting company in Germany in Hamburg and my main focus is working on edge on methods on the one side so I'm helping teams to become agile to adopt edge on methods to Tuesday edgel and improve their development process and on the other side. I'm working on an edge of technologies or technologies that are somehow related to edge and that's so I'm deeply involved somehow in this kind of Eclipse technology and I'm interested in spring and stuff like that and how to combine all these technologies. So one of my favorite technologies and it's related to the edge on methods of course refactoring that's the that's why I'm here today. OK so what. Today we are going to talk about our if you're trying to force a far out the question is What does refectory actually mean so refactoring means improving the internal structure of the internal design of a software system without changing the external behavior so that basically means to to improve the design through to restructure the software without adding features without removing features without without changing the real real behavior. You can you can discuss a lot. What would the behavior means if it means that the performer. And or if it includes just the you why or the A.P.I. or whatever it is. I think it depends on the context of his software but it basically means improving the internal structure to maintain and to cheaply a modify the software in the future to keep it healthy and to keep it soft. OK so where does that term come from who coined it. The term originates in and I think in a workshop paper it was published and it was a conference in the early ninety's and I think mainly William Updike he wrote a Ph D. thesis about three factoring operate on IT systems for Rob Johnson at the University of Illinois. I think it was in one thousand nine hundred three. I think and he say he researched all the basic ideas behind it. What is a factory and how does refactoring works and how to factor in different from from just changing a piece of software on the next breakthrough was was done by John Roberts and John Bryant and they built. They built a factory to support they will the real first refactoring browser refractory tool and for Smalltalk thinking there was at that point in time was more science all silent about this around this kind of refactoring ideas and the real that's a worldwide breakthrough or real real main interest in the term refactoring in the late ninety's with a book by Martin Fowler about refactoring and with the movement of these engine methods or factories as a main part of so why did Martine write that I mean at that specific time even though the idea was so. Already quite old and I think the the idea became much more important in these area of actual methods because an editor methods you typically have a you start with a very small implementation small system and Hugh basically working all the time improving an existing piece of software and the the idea of incremental design and creating the architecture of the system over time basically means that you need to improve us the structure of your existence system all the time all every day all the time you work on this on the software and I think therefore it was really important to have some some kind of guidance and timing of tool support and then maybe maybe a good book like like the program. I don't follow to give you some some some hints and tips and tricks and some provide some nuts and bolts of what to do what don't to do and the book is real it's a real I think it's a real brilliant book in but should read it. Even if you do not do agile extreme programming or whatever it's just something fundamental know how to work with the treatment systems and in the book I think he the big two parts are on the one hand he tells you a lot about what are the smelly smelly points in the system. What he calls a cold smell. So what is it was an ugly design award. One of the typical pieces of code where you should change the internal designs eternal structure and on the other hand he gives you a lot of advice and love patterns and what or how you can improve the internal structure internal design. So you give a lot of guidance how to say how to rename a method. How can we name a methods a very versatile. Sounds like a very very simple factoring but if you do not have to support you can think a lot of. If you have a huge system with valves of dependencies on one particular method. I think a lot of help to step by step implement this kind of effect without breaking system for a long time so the whole idea is based on doing small changes testing small changes all the time to to keep you happy and safe that you do not introduce a new box while you're factoring saw the software one of the first things obviously for effect training is to know when to apply them so you are talking about code smelts and. How do you actually find them. I know not everybody can can sniff the software solutions to right. Yeah they're the right that's right. I think sometimes you just need a good experience and can know how from a from from several systems you've built in the past and the other hand in the book are many cultivates described and there exist some tools to try to identify called smells try to sniff the whole smells and some tools like maybe find boxes and this is very simple tool but it's a very very fine grained level more sophisticated tool they try to analyze structures dependencies between classes dependencies between packages and sizes of methods for example they can tell you. For example you have a you have a class of containers and methods quite long. Maybe you should think about extract method or split into different methods. So there's some tools I think. So is a drug or something that that you can say there is is it hard scientific things to say this is a code smell or not and said rather something that comes out of experience and and more like soft know how I think it's is more. More a question of soft know how in an experience. It's more like you you have to implement a new feature for a piece of software and you take a look at the structure and you find out that the structures and we does not work for a new feature or it's quite complicated or you think you find in the code many different pieces you need to change to implement a new feature and I think they you have a good feeling about what part of the of the structure is not good and does not fit into this new feature and I think then based on this experience in this just reading code you can identify a lot of these cults melts just yourself without any kind of tools. So how does that fit into the daily work. How do you do re factoring so I mean some of the projects that I know say you do a period of free factoring after delivering a release for example so so is that if a little approach or how do you do that. My point of view is that refactoring belongs to your daily programming work so if you if you implement a suffer. It's just like creating a new class is really factoring all the time so I am I'm refactoring my my project on my piece of code all the time I rename methods I split methods I extract code I rename classes all the time to keep the design healthy all the time because if you if you think about I just I just put this piece into the software and doesn't matter if it's actually in love it smells or not doesn't matter after the next release. We have plenty of time to to improve the structure my experience is that you will never get the time to improve the structure and even if you get the time it's much more complicated to the refactoring after you have implemented for several weeks or months features that. And if you do. Refactoring all the time so it's really it's I think it's fun for me it's maybe you can think it's more important than programming. OK So what you're saying is such a period of arguments refectory is not something that you should do then in the next quest from yes. How do you may actually make people Reflektor. Because obviously this is what you're saying is it's something that every program is supposed to do as an ongoing task. Yes and they're much like tester of new element for example and there are a lot of programmers start will probably most likely not do it. So how do you deal with a problem. I think it's a different problem because it's I think some of the belongs to every developer to do. Refactoring all the time and it's just it's get hard if it gets hard if you have a team where five people do a factory of time and then last and then they work just from a natural point of view and some people don't. So it's quite hard to convince them that a factory is good. So maybe you can you can do pepper Graham with them and try to give them a try to maybe not to teach them refactoring but to give them an impression of how we factory really works and that refactoring is if you do it only do it in the daily basis. It's really it's an easy task. It's not a complicated stuff. It's really with a tool support you have today in the modern I.D.E.'s it's really easy to a factory. It's more easier than that and creating a new class maybe so it's really easy. And if I should do but if someone is convinced that a factory is not necessary at all and it's just useless work so maybe it's hard to to can convince those people. Just from these theoretical points of view but I think it's just if you if you if you want to create a software that's healthy. Over time the factoring is you cannot do it without refactoring Yeah that's certainly true. We have already I mean our new mantra testers Neuromancer what what a strong relationship between refactoring and history of development. I think the relation between test driven development and refactoring. Think about it. I think test of development is a development approach lead to rule of movement and you create a test class and then you implement the code and because of refactoring many belong to implementing the code on the on the once I it's just you. You write this case and implement the code and think open I need to respect for some stuff and then you do your fact with the stuff and you make the code test green. So that's one aspect Another aspect is that our effect if you do not have real safe to support and do many of the factories stuff. I mean you really need a test you in a test case to be sure that you do not change the behavior. You do not introduce backs and stuff like that so refactoring without full support and without a good test suite is like is like doing some crazy stuff without a safety net so you really need to test is a safety and that's that's the second level and the third level I think is what what some people say that you need to a factory a test cases as well not just cold but it can also think of cults melt in test cases so many similar tests methods or huge setups of whatever. And we need to effectively test cases as well. Yeah. You need to have the same quality in the test cases and so in the real code. Yeah exactly. OK what do I do. I mean. So basically what you're saying is if I if I don't have any test cases and I have better quality of code. I'm basically screwed because I can do and refiguring so and I will have to this bad quality forever so what do I do in that situation. I think I think the real challenge then is to write test cases all that cold and then refect with cold but it's not always that easy as it sounds like because in most cases if you do not have test cases it's really really hard and really complicated to introduce test cases for first and classes that are written without any unit tests and mind and they are not really testable. You cannot down whatever injective pendency the whatever. So you really need to to maybe some first step into that class to make it testable and then write a piece of test code and then maybe change in the other small piece of the class to make it even more testable and then write some more tests and if you have a good solid test case or test suite for that class. You can you can do the real factoring work. Of course the small change at the beginning was also refactoring and I think there is a book about that you are working efficiently with legacy code. Yes written by Michael feathers and it's titled Yes working effectively with legacy code as a real real nice book where you describe a lot of techniques how to implement those tests and what techniques you can apply to implement those tests like smoke tests and stuff like that. So the real it is a nice book talking about books we already mentioned the follow book so end. That I think gives us a connection of pedants so why should I read it. I mean what what's the benefit of reading through that I think the benefit is that you first of all get an idea about typical smells or it's one one factor and another aspect is that you get a real real great impression how to step by step prefectural a piece of code for a specific situation and he provides a lot of stuff like how you can step by step name a method or how you can split hire a key or you can change an inheritance Iraqis and how you can change relationships which in classes and stuff like that and it's a real it's maybe maybe not a book which which you need to to read from the first to the last page once but it's really worth to two to read all these patterns and all these these three factoring guidance as provided in the book to to get a feeling how I can change the internal structure without changing the behavior and how it can move forward in small steps makes me wonder whether there are any other reflect trends patterns of books that I should read which we construe terrific during I'm gather there there exist some some some books some some more boxes of them. Martin fall book is The refactoring book of course and there's are a factor in work book written by William wake. It's mostly kind of practical guidance how to learn factoring and have to do some practical examples and so I bet it's also nice book and if you take a look at those those two books they're both and basically the amount of power they put on these these base level of factoring they're working on the class level on the relay. Chip level between single classes and all these options or indeed concepts like inheritance or relationships and stuff like that they're not working on high levels. I think there's a certain sense in there that maybe to be factoring pedants the focus on more and more high level fact rings like separating presentation from from domain. It's really remains of ache in the book the guidance and examples. So their next step. I think is a book by written by Joshua curiously about refactoring to patterns where he analyzed what a typical situations and typical cults melts away for example Sure. Ray should introduce an observer pattern of how you can can introduce or extract a visitor pattern from from from from your code or and or other pedants a lot of patterns in that book and how can we factor to patterns so to goth patterns I think and on the other side. The opposite effect as well. So how can remove and necessary implement a pattern from your code where it's not not really necessary or makes a code more more complicated than it needs to be. And I actually believe that every factory every second can be done in both ways. So yes so to say forward and backward Yeah yeah. So you can extract a method you can inline and have it and both were so it's not really extract method it's the battery factory in line method. They even if it seems seem so but it's really worth to get in the question of both sites. You're foreign to mention and some tools like the original Smalltalk browser and so which truth should I be aware of and which should I use. I think the Today the morning. In the current ID implementations like like Eclipse for Java or idea for Java or whatever. Since some love tools I do not want to do advertising tools here but all these modern I.D.E.'s they all implement refactoring support on the base level so you can can you can rename a class can do all these renaming class methods whatever and you can can do movements and most of them provide even events to factor in support on the on the real on the level of single keystrokes in just rename class with a single keystroke can extract a local variety will with a single keystroke as we really need to have some summer. In your blood and in your fingers to have all these keys drugs available all the time to do the vectoring all the time and this infection suppose really it's great. I don't want to work without such an effect and support in such I.D.E.'s because it's really really great support what it was important. On the one hand is that those that there exist. Two different kinds of refactoring support or affection support and such tools on the one hand you have to support that really is safe. Can you can believe the ID that it takes care of changing all the references to your name class and changing all these whatever that you do not change the behavior of your existing system and it can be safe. It's really does not change the behavior and it takes care of that ensures here for you that and on the other side there are some reflections of a more advanced more complicated and more general maybe like at change method signature or even extract method in some situations it's quite difficult in Java to implement. Where the ID cannot give you the give you the safety of the behavior is not changed. So it's somehow it's great. We have all these ID support cars and it's easy to mix easy to factor and have all the references and stuff like that but it does not give you the possibility to skip your year safety net to skip the test cases always and test cases even if you have greater support by the ID So but but there are other X. specialized training tools outside of typical I.D.E.'s So for you if I only talk about what the the I.D.S. support. I think that some folks have for Eclipse is because six plugins would provide additional factual support and I think for for the urls to you will see some some some plugins and some some additions that provide factory support for that I'm not aware of special other tools. It gives you some provides this will help us and support. OK So basically I actually used the idea and that said I think so. One thing that I find quite artists. Today people talk about a lot about Java and be it being statically typed and the advantage of Java offering more support for I.D.S. because you can reason about the type and therefore have better possibilities for competition for example they're even refactoring. Even though I mean the air original refectory browser that we've talked about here was full Smalltalk so fine and I mean typing which so. How does that fit together even though respect trying to pierce to be something that is really good really. Well supported by statically typed languages. Only it is a concept the boss introduced by dynamically typed languages. And I think the main reason is that most of the research was done in the early Nineties in these are great on the world in the Smalltalk area is a lot of possibilities of implementing stuff into the ID end and stuff like that. So was the research stuff for that. And the refactoring support of those buildings as mater of fact in browser was was really great for the point in time but if you take a look at it compared to statically typed languages. It's it's really complicated to do rename class or even a method in Smalltalk and finding out where is this method called and where it's not called with another method meant and they sometimes they just say OK I just really every method and system that is called whatever to whatever new without analyzing if it's really the same method or its invocation of the same method or completely different method because it cannot know which method was meant because it can and can analyze a sniffly typing of that system. So it's a lot easier to implement this is all Merrick's refactoring support for steadily type languages and it's much more complicated to do it for the name of languages. I came believe that there might might appears in some refactoring support tools for Forth the name for type languages but it's a lot more difficult to implement and to realize. Interestingly people were working on for example a Ruby Ruby on Rails project. They told me that they are really big fans of refactoring that only that they have that they do. Not need to refactor so much in the dynamic minute anaemic language world like they do really all they need in these stereotyped languages I do not know why they could not explain to me why they do not need to so much in the dynamic world but it was just an observation or maybe just a feeling and I found it quite interesting. I have no experience myself with these type stuff so I cannot really prove it or explain why why. Where does this feeling comes from but it was an interesting observation I think maybe if you don't have such a good idea to support you do it less friction area but I mean who knows. I read some some reports that have that there is and there is a risk of refectory yourself to death so to say so that you just do refactoring all the time and you do much more effective than you do actually implementation bark stead a common problem or I mean isn't it neat if true to its human that you can just hack away and then refactoring will take care of for the ugly stuff that you are doing. And that will still be an efficient process of implementation. I think you need a good balance between programming refactoring I think and I know many people and sometimes I belong to this group our I think I can I can I can improve this more and I can I can improve in more and more and more in its beautiful and more beautiful and more beautiful code and I love this piece of code to make it even more better and whatever. And I think it's just a novel has been a lot of time doing that stuff without really without really improving the internal structure without getting something. Kind of benefit out of it. I think you need a good balance between these two forces of implementing stuff or ending features and re factoring and I think the main goal or the the main the main motivation for me is to keep sawfish is soft and healthy not to keep it over healthy or over soft or completely flexible for everything just being pragmatic and say what. What are the current features I need to implement for us for this piece of software and what is a good structure for that and if implemented feature in the I think after implementing the feature others design looks like. Think of cases smelts somehow and improve it but just improve it until it does not smell so just another prove to the to the ideal best design ever known or whatever. OK so it should. If you take an economic perspective it's basically an investment into the future. So you're saying OK now I have this piece of code it tends to bad design and I assume that by improving the design you have a higher productivity in the future. And how do you judge whether this refectory will pay off. It's an investment and will not be something that is just beautifying. I think I think really joshing or marrying it in money somehow. It's quite difficult because it's it's you know who can who can really measure how maintainable a piece of software is or how how much is a cause in five years to add new features. I think it really pays off quickly if you have to have these offices and healthy design and if you factor all the time because it doesn't matter if you implement a featured. Today or in five years into such a piece of software it would always cause the same does not cause OK if you shoot me if you think about the future. Now it's cheap to integrate into the system. Think about it tomorrow. It costs a lot of money because we need to change a lot of code and we need to effect through the basic basic architecture of the system whatever. So I think it's more this is kind of idea of if you if you are going to implement a piece of software that lives for for a longer period of time me for a month or for four years or me for a dozen of years. I think it's an inherent goal of keeping this piece of software healthy and I think the only way to keep this piece of software healthy is to to refactoring from the first day on time so you can and you can from my point of view can live without it for him on the other hand I'm pretty sure there exist projects where you just implement say one way software or throwaway suffer. That's what you think you're doing. Maybe and I know think this is this is really true because I know of people they implementing software for for brokers and those brokers they really need software for tomorrow and just for two more interest for this specific contract or for the specific market situation or whatever and they need a really highly specified piece of software just for for this deal. So they're there they program the software in doesn't matter if they do copy and paste or whatever they just needed for one day and they threw it away after the day so and then they needed. Maybe another piece off and next day soon. What you're saying is basically that it's a technique then I'm using on a more fine grained level. And in there also. It is something that is somehow need to to judge when to do by your experience so on. I'm wondering is should a man in show or a project leader care about refactoring or stare anything from the management level that that you can do with recounts terrific trying to enforce it. Or is it just some fine grain stuff that a good developer does. I think the goal from the management point of view is to take care of a good design in your software because you would like to have a piece of software that is maintainable and changeable in the future that is your management goal you would like to have a piece off of that in two years you can you can add features with the same speed in the same mall and he as you do now. So maybe maybe it's a golf Amendment but I think it's often a government from the management point of view and maybe doesn't matter how the team realizes this goal of how the team ensures that adding this feature whatever features need or whatever changes and requirements they may may or may occur in the future can be implemented fast and quickly in a high quality level into the sufferer. Maybe it doesn't matter how the team ensures that it can that be able in the future to do this but I think that refactoring is the most important technique for the for the team to ensure this goal so it's not really I think refactoring as as taking league itself does not matter to the management so much more the fact that you create with refactoring and that matters to the management is there some some advice that you could give to a project manager or whether when he should. Bring in someone that looks at the air that improves the techniques in the team. So some kind of of of hint that that men are showcased so obviously the team is not doing really bring in there. In a proper way that reason we now have quality problems and find a way of improving three Think drink knowledge in my team. I think I think of course getting people into the team that are experienced in refactoring that they can teach other people like that can show other people how to effectively help that is only says it's always useful if you have a consultant or whatever for for a limited period of time and sort of project. It is experience of this kind of techniques but I think I think for four teams that I know or where I was involved in it's difficult to say OK he has a book I read this book. It's good. Sounds good. Just do it from the on the management because all the developers think he's a manager who does not know what really happens here and so the coincided to the system that didn't tell everything and we will throw it away just a book put it into the every other there on the board. Yeah OK so thanks a lot for for we will have another episode that is going to be going to talk about it with Martin as well so and stay tuned for more information about refactoring in the next episode. Thanks for listening to soft engineering radio. If you want to get more information about software engineering radio or if you want to give us feedback that you can also contact the team at Team entries in our comments system on the website so other people can see what you think software engineer and radio wants to thank this episode radio as well as all of its creative commons see the website.