Pesquisador. Muito obrigado por participar da entrevista. Ela tem como objetivo compreender os processos de criação e de manutenção de código de testes e também verificar como é realizada a verificação da qualidade desse código. Não há respostas certas ou erradas ou desejáveis ​​ou indesejáveis. Dessa forma, senta-se à vontade para dizer o que realmente pensa. Reforçando um item do termo de consentimento, nossa conversa está sendo gravada. O objetivo disso é para que eu possa obter todos os detalhes, mas ao mesmo tempo ser capaz de manter uma conversa atenta com você. Garanto que todos os seus comentários permanecerão confidenciais. O relatório final conterá os comentários dos participantes sem nenhuma referência aos indivíduos.

E. Perfeito

P. Como você começou a trabalhar com teste de software?

E. Eu comecei a trabalhar na verdade, o meu cargo sempre foi de desenvolvedor e os testes, a minha relação com os testes é muito mais o cenário de automatização de teste sejam eles testes unitários ou de integração. Como eu comecei a trabalhar com isso na prática? A gente lê muito na academia que é desejável, mas muitas vezes a gente não pode exercitar essa realidade no dia a dia do trabalho por N motivos aí. Muitas vezes relacionados a prazo, gestão, etc. A partir de 2015 com um projeto na época novo que a equipe iniciou esse projeto finalmente pela primeira vez na vida eu vi a coisa vendo com testes automatizados desde o princípio.

P. Quais foram suas fontes de aprendizado sobre código de teste?

E. A principal fonte de aprendizado foi para biblioteca específica foi a documentação oficial. De vez em quando um tutorial na internet para um ponto muito específico e o próprio repasse dos demais colegas. Porque isso? Porque na verdade o conceito em si a ideia por traz do teste de unidade já não é nova, já tem uma certa idade então de certa forma quase todo mundo sabe descrever com um nível razoável o que seria um teste unitário, mas poucas pessoas de fato colocaram em prática em determinados cenários. A parte conceitual realmente fonte provavelmente livros que eu li na época, anos e anos atrás, mas numa realidade mais concreta, a principal fonte foi documentação oficial e os próprios colegas que já trabalharam com esse tipo de metodologia em empresas anteriores. A codificação e como organizar esses testes. O que eu foco em testar, o que eu não devo testar o que eu não devo testar em sua plenitude, o que eu devo mockar, o que eu não devo mockar, esse tipo de coisa. Claro que os colegas ajudam e depois você vai pegando experiência e pegando um pouco mais de noção e vai sendo capaz de tomar essas decisões sozinho.

P. Em quais linguagens você cria testes de unidade?

E. Em Java, na verdade começou com Java inclusive, em Nodejs, seja no servidor, seja no lado cliente, utilizando o Jest e principalmente o que eu mais usei de longe foi em Go

P. Qual era linguagem do seu projeto na empresa que você trabalhava?

E. A principal em backend era Java, e a linguagem em frontend era Javascript React JSX.

P. O projeto era web?

E. Isso. Era um projeto standard que nós temos hoje em dia de um serviço web backend e uma camada frontend na frente

P. Como era seu processo de criação de teste de unidade.

E. No momento inicial, esse processo primeiro identificava se realmente seria um teste de unidade isolado, de um componente. Esse era um processo mais raro porque geralmente muitas das nossas realidades das lógicas estavam interconectadas então você precisava de um banco de dados razoavelmente aceitável. A gente usava o sqlite para isso e você em um determinado end point, determinado url, você raramente fazia um acesso simples, a apenas um repositório de dados. Você acessava as vezes 2, 3 ou 4 ou chamava outro serviço auxiliar. Então unitário puro só uma coisinha mesmo é muito raro. o que acontecia muito é o que muitas pessoas chamam de integração, que você mocka apenas o que é essencial, tipo banco de dados e utiliza implementações concretas, por exemplo, como services e acessa esse banco, utiliza essas implementações concretas dentro do seu teste ao invés de mockar tudo em 100% nós observamos que esse tipo de caminho muitas vezes mais interessante nos dá resultado melhor que realmente querer mockar de ponta a ponta exceto aquele único componente sozinho específico.

P. Existia algum fluxograma, template... que padronizasse esse processo?

E. Em um primeiro momento a empresa tentou iniciar essa onda de testes utilizando BDD. Caso você não esteja familiarizado com isso na prática eu recomendo dar uma pesquisada sobre o Cucumber, Gherkin, enfim, o nome da lib que a gente usava era Cucumber e a ideia era que a pessoa responsável por testes que nós tínhamos uma equipe de teste, os testadores que eles escrevessem textualmente o que é o teste, então "quando o usuário clicar aqui faça aquilo" e depois nós programadores utilizássemos essas frases para realizar a ação concreta e servia como teste de documentação. Na pratica nós vimos que esse caminho nesse momento, sim. Existia templates. Nós vimos que essa maneira de se criar teste rapidamente foge do controle, do termo de organização, sendo bem franco, é muito fácil quase incentivado você fazer certos tipos de gambiarra por falta de termo melhor, e dificultar a manutenção do seu código de teste. Então nós decidimos que mesmo trazendo overhead para os programadores escreverem os testes do zero nós tínhamos uma manutenção mais baixa a longo prazo que deixando o pessoal de teste escrever esse texto do BDD. Então no fim das contas, digamos que BDD não funcionou tão bem quanto alardeado, pelo menos no nosso caso e voltamos para uma metodologia, uma visão sobre os testes mais selecional. Fica tudo a cargo do programador.

P. Quais ferramentas eram utilizadas para criar/ executar teste?

E. Era utilizado basicamente eclipse, que era a ide e as bibliotecas Java standard de execução de testes como JUnit. Na época do Cucumber tinha obviamente o Cucumber runner para fazer essa transformação. Então do nosso lado de programador realmente não mudou muita coisa. Para pessoa que escrevia os testes na época do BDD, além disso ela utilizava editor de texto, o Sublime mas em tese poderia ser qualquer coisa e como ela só escrevia o txt para gente nada mudou muito nesse sentido. Então as ferramentas eram tudo que a gente já usava.

P. Como era o processo de verificação de qualidade do código de teste criado, por exemplo: verificação de escrita...?

E. Para ver se pegava realmente os casos que tinham que pegar?

P. Não só isso, estrutura do código, se tinha alguém que fazia a revisão desse teste depois de criado, e também compatibilidade de requisito, cobertura...

E. No geral nesse primeiro estágio do BDD não tinha muito controle porque a coisa vinha muito unidirecional, vinha do teste para o desenvolvimento e nosso trabalho era concretizar aquilo, fazer com que aquilo rodasse. Quando a gente largou mão do BDD, num primeiro momento como a gente tinha um backlog de testes, não feitos em relação ao que foi feito deploy era meio grande então num primeiro momento, nos organizamos para focar nos pontos em que eram mais críticos para o negócio em segundo lugar parece os que vinham mais reclamação. Na época o cliente estava começando a usar sistema então você não tinha tantas requisições assim. Então nosso foco era primeiro junto ao pessoal de negócios o que é que é essencial que não tenha falhas? Então vamos sentar, vamos listar literalmente abrir um documento, um doc no Google docs, e botar ponto a ponto o que não pode falhar nunca. E aí a gente focou em criar testes para esses casos. Em relação a estrutura dos testes em si, na época do BDD a coisa estava caótica, quando a gente largou aquilo, rapidamente nós batemos em um problema que era ter dados reais, ou reais o bastante para fins de teste. Então as vezes eu precisava fazer um teste por exemplo de inventário, mas antes mesmo de eu tocar no inventario, por a gente utilizar os serviços concretos, você antes tinha que ter usuário, produto, estoque, tinha que ter aquelas N dependências. Na época um colega nosso indicou uma biblioteca Java chamada environment que ele mesmo criou em outra empresa, quando eles começaram a utilizar teste, e essa própria lib environment nos deu uma estrutura muito boa, apesar de não ser o foco dela. Uma estrutura que primeiro eu definia de maneira declarativa em alto nível o que eu precisava para determinado teste, então "olha para o teste tal eu preciso de estoque" só que quando eu digo estoque, está implícito que para estoque eu preciso de produto eu preciso de usuário, de XYZ, então essa lib cuidou dessa parte de dados e isso meio que mudou nossa organização daí pra frente.

P. Nesse caso por exemplo. Quando terminava a escrita, não tinha necessariamente uma revisão direta por outro profissional, mas ela tinha que seguir aqueles critérios que foram descritos anteriormente.

E. Isso. Ele tinha que seguir esses critérios, é claro que os testes automatizados não substituíam o (teste) manual, então muitos desses critérios eram visíveis e verificáveis manualmente usando o sistema. E a revisão de código a pesar de existir na empresa naquela época ainda era uma prática, aliás até o tempo que eu saí, ainda estávamos nessa situação, de ser ainda uma prática muito para inglês ver, por assim dizer, muito insipiente, a empresa deixou que apenas algumas pessoas pudessem revisar código então, eu por exemplo, sendo desenvolvedor não poderia revisar o código do "Nildo" desenvolvedor. Quem teria que revisar tanto teu código quanto o meu era o "Zé da silva" que era o nosso arquiteto. E o arquiteto a grosso modo tinha mais o que fazer da vida que passar o dia inteiro fazendo revisão. Outros arquitetos não são muito cuidadosos com a revisão então basicamente se o build passasse, que era compilar no Jenkins e passar pelos testes definidos de acordo com o CI simplesmente tocava para frente e pronto. Então isso era muito comum acontecer e por causa disso essas revisões do teste era meio que uma piada, não eram definitivamente levada a sério.

P. Você comentou que a manutenção de teste não era algo que ocorria com certa frequência, uma vez que as vezes eram feitas coisas para acelerar o processo, mas eventualmente não era feita manutenção disso. O processo de manutenção ocorria depois disso em algum momento ou era algo que de fato era deixado para segundo plano?

E. Essa questão da manutenção do teste só foi uma coisa grave mesmo enquanto a gente usou o BDD, porque as vezes alterações simples na regra que a gente precisava envolvia alterar N classes de teste, não era nem do sistema era estrutura de teste, e tinha que ser alterado para você conseguir fazer passar aquele teste e muitas vezes você não tinha prazo para isso então realmente foi um pesadelo essa parte na questão do BDD. A partir do momento que a gente abandonou o BDD e usou aquela biblioteca como eu te falei e basicamente usou o JUnit + bibliotecas auxiliares, a manutenção ficou algo muito mais tranquilo, muito mais razoável de se fazer. Uma vez que era criado um teste raramente você visitava aquilo de graça você revisitava aquele arquivo, aquela classe de teste quando você implementa uma nova regra, quando removeu uma regra existente, em alguns momentos algumas regras ou situações bem corner case do sistema a gente as vezes implementava primeiro o teste, o que seria o mundo ideal mas nem sempre a gente consegue fazer na pratica, algumas delas que eram mais importantes, a gente primeiro implementava os testes, rodava ele garantia que estava falhando que de fato fazer a implantação para passar naquele teste. Uma vez que a gente saiu do BDD as coisas se normalizaram bastante e essa manutenção era feita com frequência sempre que o código relacionado a ela era mexido.

P. O que você conhece sobre test smell?

E. Como é que nós, não vou dizer que nós caçávamos test smell, mas uma das ferramentas que a gente integrou no nosso build na época do Java foi o SonarCube, para quem não conhece, não sei se já mexeu com esse cara. Nós integramos o sonar na nossa pipeline e instruímos ele a procurar code smells na base como um todo. Ele não olhava apenas para o fonte oficial, mas também para os códigos de teste. Nem todas as regras dele nós utilizamos, porque tem algumas regras que são referentes à estilo, por exemplo e a empresa já tinha o próprio estilo então não faz sentido você reclamar em relação aquilo então era ignorado quando possível, então os smells a gente pegava justamente pela indicação do sonar. Os smells eram coletados de uma vez só, mas eram divididos em 2 relatórios. Um relatório para o fonte em verdade e outro para o fonte referente aos testes. É claro que muitas vezes o foco era diminuir as issues que ele apontava no fonte principal então se eu tivesse smells no teste até um determinado nível, não vou saber dizer agora a configuração exata, mas um determinado nível de tolerância o sonar deixar o build passar, desde que o teste em si passasse, então se o teste passou mas se tem uma certa duplicidade, ele só avisava no relatório mas não reclamava, mas no fonte principal esse nível de tolerância era consideravelmente menor, a gente foi reduzindo ele com o passar do tempo, conforme se adequava as regras e mesmo que passasse teste, passasse tudo, se tivesse alguns tipos de smell considerados importantes, ainda assim o sonar dava reject. E o que a gente combinou na equipe é, o meu código só vai ser revisado se o meu build do Jenkins passou se os meus testes passaram e se o sonar não achou nada locker, em vermelhinho. Então apenas esse caso é que ele sobe para fila de revisão. isso forçava o pessoal a não introduzir novos smells ou se introduzisse, com uma frequência bem menor que o que introduzia código na nossa base como um todo, mesmo assim com o tempo, depois de um ano e pouco, alguns itens começaram a acumular entre eles uns de fácil resolução mas que você acabava não resolvendo por questões de prazo, então nesses casos a gente utilizava um júnior da equipe porque realmente eram coisas muito simples de se resolver, que apenas foram acumulando, a gente utilizava um júnior e colocava uma tarefa para a pessoa corrigir isso porque era interessante porque ela já dava uma olhada em trechos de código que as vezes ela não tenha tido a oportunidade de trabalhar e como aquele ia ser revisado a gente não se preocupava tanto assim.

P. Conceitualmente você tem alguma dúvida sobre esse assunto?

E. Não. Não consigo vir nada a cabeça.

P. Como test smells são tratados no processo de criação do código de teste?

E. Contemplado na questão sobre o que conhecia sobre test smell.

P. Como test smells são tratados no processo de manutenção do código de teste?

E. Contemplado na questão sobre o que conhecia sobre test smell.

P. Para você, como seria possível prevenir/evitar a inserção de test smell no código de teste no momento que ele é criado? Quais técnicas de programação, processos organizacionais podem ser adotados?

E. Eu acho que a melhor maneira de evitar isso é a própria experiência. Porque muitos desses smells aconteceram porque o time não tinha um grau de maturidade tão grande para testes. Poucas pessoas que efetivamente tinham trabalhado de forma mais ostensiva com isso na época, então muitos dos smells eram na verdade falta de experiência e falta de orientação. Isso acho que era o fator que de longe mais consegue cortar esses smells. E é claro que outro fator relacionado aliado a isso é uma review, um code review um pouco mais coerente, nosso caso mesmo não tendo um code review perto do aceitável, para mim não era aceitável e continua não sendo aceitável, ainda sim nós conseguimos mitigar muito desse risco com as regras do sonar por exemplo. Então nas linguagens que o pessoal tem e consegue configurar linters verificadores de código, esse tipo de coisa, se você utilizar essas ferramentas, mesmo falhando em áreas importantes como code review, você consegue manter um nível baixo o bastante de smells. Para evitar é realmente só a experiência a prática, não digo nem informação. Muitas coisas que fizemos eu acho que fazer um curso por exemplo de testes em Java não resolveria de toda forma, só o aprendizado pratico mesmo que resolve algumas coisas.

P. Estou chegando ao final da entrevista, você tem alguma pergunta para mim, quer acrescentar alguma informação ou tem alguma sugestão para melhorar nossa entrevista?

E. Talvez, eu não sei qual o teu foco exato da tua pesquisa, mas uma pergunta talvez que seja interessante coletar esse dado é o quão a percepção dos entrevistados em relação a qual é a barreira para se iniciar os testes automatizados e vou meio que te dar a resposta. No meu caso que eu digo isso, porque como eu disse, você vai ver no questionário que dessa empresa em que todas as respostas foram baseadas, eu passei 6 anos. E ela trabalha com Java. Mas se você lembrar do início da entrevista, quando eu citei as linguagens que eu escrevi teste, eu falei que de longe a linguagem onde eu mais escrevi testes era em Go. Se você der uma olhada no meu currículo, você vai ver que a empresa logo em seguida dessa é justamente a que eu trabalhei com Go e eu trabalhei male male um ano. Então, como é que em um ano você escreveu mais testes que nos últimos 6? Muito simples de responder essa questão. A barreira mental e até de programação mesmo para se escrever um teste em Go é muito muito próxima do 0, porque nunca é 0. que eu quero dizer com isso é que se eu começar um projeto novo em Java hoje, por exemplo e tiver que testar com JUnit apenas, não vamos colocar mais nenhuma biblioteca externa hoje eu tenho que iniciar o meu projeto e em seguida abrir pelo menos uma aba do meu navegador e dar uma pesquisada em como configurar o JUnit, qual que é a dependência, como faço o setup esse tipo de coisa. Tem uma certa carga mental para fazer esse tipo de coisa. Depois tem que lembrar que setar uma tag de testes separada, ver quais são as notações. Teria que correr atrás, isso porque já fiz. Se eu não tivesse feito, a barreira ia ser maior. Se você compara com o Go, como eu começo criar um teste em Go? Cria um arquivo com o nome do arquivo que já existe, coloco "underline test" e começo a escrever. Pronto. Runtime a configuração ja existe, é automática é inerente a linguagem, é perfeito. Outro exemplo semelhante, em React, que o Facebook utiliza e já vem pré-configurado com o Jest para fazer os testes... como eu começo a fazer um teste em react? é praticamente a mesma coisa: cria um outro arquivo, se eu quero testar um arquivo "fulanodetal.js" crio um "fulanodetal.js" e começo a escrever, porque o ambiente está pré-configurando. então esse tipo de coisa diminui muito a barreira e basicamente o programador, mesmo um programador mais relaxado por assim dizer, fica desculpa de porque não fez o teste porque muitas vezes o teste não é feita porque a gerência, coordenação dos projetos as vezes falha em não exigir esse topo de coisa e não deixar tempo para o pessoal fazer e em outros momentos falham porque o desenvolvedor dá essa desculpa: "ah é um negócio muito urgente, vocês precisavam isso antes da sexta, eu fiz e depois eu faço o teste e fica o débito. E aí talvez nunca seja feito. Só quando a bomba explodir outra vez. E se você tiver essa barreira muito baixa essa desculpa do programador começa a ficar difícil de se engolir, tanto para gestão para coordenação quanto para restante da equipe. "Ah, mas porque você não fez teste disso?" "Ah...." você não tem em quem jogar a culpa, não tem como dizer que o cachorro comeu a lição de casa se você não tem cachorro. Esse é o único comentário que eu faria, não sei se é o teu foco. Se não é me desculpe de perder o seu tempo.

P. Tranquilo. O foco da minha pesquisa não é de fato nessa parte introdutória. O foco da minha pesquisa é em test smell. É entender dentro da indústria como funciona os processos de criação e manutenção, primeiramente e depois como os test smells são tratados nesses 2 processos. Se eventualmente ele só e tratado no momento da manutenção e é refatorado o teste ou se desde o início ele já vem sendo pensado e de alguma forma evitada sua inserção dentro do código de tese. O foco da minha pesquisa é nesse sentido. É claro que para que isso ocorra as empresas tem que ter a atividade de teste acontecendo, então o seu depoimento é importante, não só para mim, eu já trabalhei com empresas que tem essa resistência até hoje com a atividade de teste, muito mais por não identificar a importância do teste em si do que necessariamente a linguagem a ser utilizada por exemplo, mas eu posso passar essa ideia de pesquisa por exemplo para o meu grupo de pesquisa, a gente tem um grupo de pesquisa dentro da UFBA, dentro de engenharia de software e vem sendo feitas pesquisas em diversas áreas na parte de teste. Entao tem gente pesquisando a parte de teste de regressão, eu estou pesquisando a parte de test smell como outros colegas, então pode surgir alguma oportunidade de pesquisa através da sua ideia que de fato é relevante.

E. De toda forma estamos aí caso precise de algum esclarecimento, mas o único comentário era esse mesmo.

P. tranquilo. Muito obrigado pelas informações, pelo seu relato. Eu vou parar aqui a gravação, mas caso você tenha algum outro contato que você possa me indicar para que eu possa realizar a pesquisa, eu vou agradecer bastante.

E. Eu comentei de um outro colega que já trabalhava com teste antes, eu vou ver se consigo falar com ele agora e se ele aceitar eu já te passo o contato no Linkedin ou por e-mail. Com esse cara acho que vai dar para fazer uma entrevista muito legal, mais legal do que essa que ele tem mais experiência essa área.

P. bacana, só um detalhe que eu te peço é que não comente que você estiver conversando com ele não passar nenhum detalhe da nossa entrevista.

E. Com certeza

P. Muito obrigado. Boa tarde para você.