Pesquisador. Muito obrigado por ter aceito participar dessa entrevista. Ela tem como objetivo compreender os processos de criação e de manutenção de código de testes e a qualidade do código de teste nesse processo. Não há respostas certas ou erradas ou desejáveis ou indesejáveis. Dessa forma, senta-se à vontade para dizer o que realmente pensa.

Entrevistado. Certo

P. Como mencionei, 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 e que no relatório final conterá os comentários dos participantes sem nenhuma referência aos indivíduos.

E. Entendi

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

E. Na verdade eu tive sorte, posso dizer assim. Foi no meu primeiro emprego excluindo os estágios. Eu já comecei a fazer os testes unitários, mas no primeiro momento nós fazemos porque éramos obrigados. O cliente exigia que tivesse testes unitários.

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

E. Foi um mix. Eu li muito conteúdo na internet. Não teve nenhuma fonte específica. Eu fui aprendendo na prática, na tentativa e erro, desenvolvendo os testes procurando alguns conteúdos na internet, mas não teve nenhuma fonte específica. Algumas coisas eu aprendi vendo o código de outras pessoas. Em algum dado momento, eu cheguei a ler algumas coisas, alguns capítulos de alguns livros, mas eu não vou saber dizer quais. Era um livro famoso de testes, na verdade era um livro sobre teste de software. Ele abriu um pouco a mente sobre como criar os testes. Agora estou lendo um livro de Martin Fowler sobre refactoring. Esse livro ele fala sobre muito sobre testes também, porque para você fazer o refactoring corretamente, você tem que fazer os testes unitários de forma certa para isso.

P. Em quais linguagens você cria testes?

E. Atualmente Java a maior parte da minha experiência foi em Java. Mas também eu já fiz testes em Python e C#.

P. Qual plataforma do seu projeto atual?

E. Java

P. Me conte como é seu processo de criação de teste.

E. A gente não tem um processo definido pelo time. O projeto está aberto; a única coisa que a gente tem por alto é que tem que ter testes unitários e sempre que você cria um repositório, você tem que criar o número de testes diferentes de zero. Eu tenho minha própria dinâmica, como o projeto é aberto, eu tenho minha própria dinâmica para criar os testes e a própria equipe também. Como a forma que o projeto é conduzido todo mundo cria testes unitários e cada um vai criando os testes da sua maneira e adaptando. A gente não tem de fato um processo definido de como fazer. No meu caso especificamente, eu tenho algumas regras: sempre que eu tenho algum um bug encontrado, eu tenho um teste para reproduzir o bug; sempre que eu vou ter um código de software já existente, caso não tenha código de teste já existente eu crio um teste e depois eu altero o código do software, e se eu for criar um código novo eu crio a quantidade de testes diferentes de zero, mas aí a quantidade de testes que eu vou fazer a cobertura dos Testes que fazer depende de quanto tempo eu tenho disponível. Mas aí eu tento fazer teste cobrindo uma amostragem boa, pois como a gente sempre está mexendo no código, eu sei que mesmo se eu não fizer teste para tudo, mais na frente quando eu for mexer, eu posso colocar. Como a gente está mexendo no código, está sempre “debugando”, conforme eu vou enxergando que está faltando algum teste, eu vou adicionando. Por isso que eu não me prendo tanto, se eu for adicionar algum código novo, de ter que já estar tudo fechado. Eu tento cobrir alguma coisa, sigo, pois sei que quando tiver mais na frente eu posso adicionar mais um, e adicionando de um em um, a gente vai tendo uma cobertura razoável.

P. Existe algum fluxograma, template... que padronize esse processo?

E. Não (comentado na questão anterior)

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

E. Como a gente está utilizando Java e a gente não utiliza nenhum framework específico, pois nossa aplicação roda em background, o JUnit é suficiente para isso. Ele nos dá todas as ferramentas que a gente precisa, a gente usa o JUnit 5 que é poderoso, tem várias features interessantes. A gente também usa o Mockito, que é outra ferramenta em Java para poder mockar objetos e a gente utiliza para poder isolar o nosso código de dependências externas, então o que for dependência externa a gente mocha e assim a gente consegue criar um teste unitário isolado, e que não depende de nada. Junto a isso, como gerenciador de dependência a gente usa o Gradle. Ele te dá algumas tasks para executar os testes, e junto a isso a gente usa o Jenkins que é a ferramenta de Integração contínua e como é que funciona: Quando eu adiciono os testes eu posso executar eles localmente, eu posso executar individualmente ou eu posso utilizar o Gradle para executar os testes para mim ou eu posso, quando eu crio um pull request, quando a gente desenvolve o pull request, a gente cria um branch e quando a gente cria o pull request no Jenkins a gente “starta” um job no Jenkins e ele já executa os testes de unidade e o pull request só fica habilitado para merge depois que esses testes passam. Quando a gente faz o merge a gente também roda os testes unitários e quando a gente gera um “release”, os testes unitários também são executados. Agora a gente também teste de integração e o que a gente chama de teste de integração são testes que a gente não está mais mochando nada, a gente não está utilizando o Mockito, a gente de fato chama os serviços. A gente também chama o Jenkins e o Gradle para isso mas a gente usa uma task específica que é uma task para os testes de integração e aí a gente utiliza o Docker para poder criar as nossas dependências, então se eu dependo, por exemplo, de um banco de dados eu crio esse banco de dados em um container, e quando eu executo isso no Jenkins, o Jenkins cria o container pra mim com as dependências que eu preciso e a gente roda os testes de integração chamando os serviços externos que não serão externos nesse caso, mas não estamos mais mochando nada, está chamando os serviços de fato, mas dentro de um ambiente controlado. Aí a gente roda os testes de integração. Só que isso não está, a gente ainda está no processo de integração disso, a gente já tem essa suíte pronta de teste de integração, utilizando container, a gente não tem isso integrado ao pipeline. É o que a gente está trabalhando agora. E a gente também tem os testes pela equipe de QA, que também são automatizados. Eu sei que eles usam Cucumber, JUnit, mas isso não está integrado com o nosso código principal, mas a gente está trabalhando para integrar de forma que pelo pipeline a gente rode tudo.

P. Dentro da empresa, como é o processo de verificação de qualidade do código de teste criado?

E. Eu não vou falar da empresa, vou falar do time, porque a empresa não tem uma diretriz que todo mundo tem que seguir. Cada time trabalha de uma forma. O meu time especificamente a gente usa o Peer review, minha primeira experiência com peer review e eu estou gostando bastante, estou achando mais efetivo que ferramentas que eu utilizei no passado. Em outras empresas a gente usava o Sonar para poder fazer essa análise de código, só que talvez não fosse um problema da ferramenta em si. O Sonar te dá várias coisas que podem ser melhoradas no código, mas havia um problema na forma como os desenvolvedores resolviam os problemas. Estava muito focado em fazer sumir o problema, mas não necessariamente em resolver da melhor forma possível, então muitas vezes o código ficava mais complexo para poder resolver aquilo que o Sonar estava apontando. Então a gente acabava tendo um código que ao meu ver era ruim, mas que passava nas métricas do Sonar. Hoje, apesar da gente estar implantando o Sonar agora, a gente usa o peer review e o meu sentimento é que funciona muito melhor do que isso. Do que usar esse tipo de ferramenta. Quando a gente cria um pull request, pelo menos 2 desenvolvedores têm que revisar o código e a gente acaba encontrando uma série de coisas. O review serve para verificar se o que está sendo “commitado” vai quebrar, mas também serve para avaliar esse tipo de coisa, se o cara poderia ter feito um design melhor, se o código poderia ser mais legível. A gente requer pelo menos 2 reviewers, mas como a equipe é grande a gente pode ter mais que 2 reviewers, mas a gente espera que pelo menos 2 pessoas revisem esse teste para que possa seguir a diante. A nossa regra é tem que ter 2 aprovações, mas a pessoa que abriu o pull request, se ele achar que precisa de review de outras pessoas ele pode pedir para outras pessoas e esperar que essas pessoas revisem o código dele, isso faz com o que o nosso código tenha uma qualidade de escrita razoável. Os testes unitários ajudam bastante nessa qualidade, porque a gente tem uma quantidade grande de testes unitários. Então sempre que a gente quebra alguma coisa, quando a gente faz o pull request que o Jenkins vai apontar que tem algum problema e também os testes unitários induzem a gente a ter um design melhor, porque ao depender de como você constrói o código ele fica impossibilitado de ser testado. E sempre quando a gente faz o código a gente já faz pensando nisso e quando a gente se depara com um código que não dá para testar a gente refatora ele para que ele possa ser testado. Por exemplo, a gente tem um código que se comunica com web services e a gente usa uma biblioteca que usa métodos estáticos para fazer isso e a gente já viu que isso não funciona, isso acaba bloqueando muito na hora de fazer testes unitários. Você não consegue criar os mocks e a única forma de testar é tendo um end point funcionando para poder executar. Então a gente já sabe que não é mais para usar isso, então qualquer coisa nova que precisa se comunicar com web services a gente não usa mais essa biblioteca e o que já existe a gente está mudando para usar outra biblioteca que permite a gente mockar as chamadas. A gente tem um código razoável. Agora a gente está implantando o Sonar. Eu não sei como isso vai ser administrado daqui para frente. A gente está colocando agora, já tem alguns serviços que está rodando o Sonar, mas por enquanto a gente não está olhando nada, a gente só configurou; a gente consegue ver algumas métricas, a única coisa que a gente já percebeu e que está vendo agora é a cobertura dos testes. Ele apontou X cobertura de testes e aí a gente vai trabalhar em cima disso para melhorar a cobertura, mas não decidimos como vamos abordar a questão da ferramenta. Em relação a entregar o que foi pedido, a gente tem uma equipe de QA que é dedicado apenas a equipe e eles desenvolvem junto com a gente. A gente tem algumas features para implementar, então dado a um determinado requisito, ao passo que está complementando a feature, eles estão implementando os testes automatizados. Então quando a gente termina, a gente manda, faz a implantação para teste e eles rodam os testes. E. Eles verificam o que foi especificado, criam os testes em cima disso e então validam. Normalmente a gente tenta fazer isso dentro de uma interação.

P. Passando para o processo de manutenção do código, como ele é feito dentro da equipe?

E. Eu não sei como te responder. Porque a gente não faz distinção entre feature e manutenção. Do ponto de vista do desenvolvimento, no final das contas é tudo manutenção. Porque se você vai adicionar uma feature você vai ter que mexer no código. Como desenvolvedor, não consigo ver distinção no que estou fazendo hoje. Uma única diferença. Quando eu mexo em alguma coisa que está funcionando para não quebrar. Nesse caso, a gente tem algumas estratégias, a gente tenta seguir. A gente pode criar uma espécie de flag, de forma que se você, eu já precisei criar um novo processo dentro de um processo que já estava implantado em produção, a gente adicionou uma forma de habilitar e desabilitar essa funcionalidade em produção de maneira que se desse algum problema, a gente resolve. A gente implementou com essa flag, a gente fez a implantação, viu que estava funcionando, se desse algum problema a gente só desabilitava que já resolvia o problema em produção e a gente tentava resolver o problema internamente. Funcionando, a gente removia a flag para que ficasse sempre disponível em produção. Outra estratégia que a gente utiliza é: a gente precisa implementar uma feature nova, só que a gente já tem o código que faz isso, só que esse código não é genérico o suficiente, porém a gente não quer correr o risco de alterar isso para não quebrar o que já está em produção. Então a gente copia o que já tem, isso a gente fez o teste agora porque antes a gente refatorava primeiro para deixar genérico, alterava o que já estava lá e colocava os dois para apontar para o novo. Só que a gente está testando, a gente fez isso agora para não correr o risco de quebrar o que está em produção. O que a gente está fazendo: a gente copiou o que já tinha, a classe que já tinha, refatorou para que ela fique genérica, a gente está em processo de teste disso agora, uma vez a gente validando, a gente pega o antigo e substitui pelo novo, que aí a gente não tem o risco de se quebrar alguma coisa a gente não quebra o que está estável em produção. Então a gente cria uma cópia, faz as alterações que forem necessárias, aponta a nova feature para essa nova implementação, uma vez que está tudo funcionando, a gente pega o código antigo e aponta para o novo também. É uma forma da gente fazer a transição mais segura. A gente começou a testar isso agora basicamente isso. A diferença é que com a manutenção, a gente tem que ter um cuidado maior porque o código está em produção e tem que tomar cuidado para não quebrar nada. Então a gente sempre adiciona mais testes unitários, então vai alterar o que já existe, a gente adiciona mais testes unitários para dar uma confiabilidade maior, adiciona se já existir código para isso, a gente copia e depois que for validado a gente faz a alteração de fato para o código antigo.

P. Nesse processo, vocês só criam testes unitários ou vocês alteram os testes unitários? Como funciona isso?

E. A gente altera os testes unitários. A gente não altera tanto, são testes que estão mais estáveis. Surge uma necessidade maior de alterar os testes unitários caso a gente precise fazer alterações que muda a estrutura do código, que, como a gente refatora bastante, isso é uma coisa que acaba dando uma qualidade maior ao código. Então ao depender você precise alterar o teste. A gente não tem problema com os testes estarem quebrados, já tive esse problema no passado, porque em outras empresas a gente não tinha uma ferramenta de integração contínua, e a gente não rodava os testes com frequência, então às vezes quando a gente ia rodar, os testes estavam quebrados. Como a gente está sempre rodando os testes sempre que a gente abre um pull request, na verdade sempre que a gente faz um push a gente roda também. Toda vez que a gente faz um push os testes unitários são executados. Se a gente quebrar o teste a gente já identifica isso cedo e a gente vai lá e ajusta. Em relação a, levando em consideração que a gente não quebrou nada, as vezes acontece de alterar os testes para melhorar o teste. Não acontece com tanta frequência, mas acontece também.

P. O que você conhece sobre test smell?

E. Nunca ouvi falar.

P. Ok.

E. Você fala test smell do tipo code smells? Aí teria test smell?

P. Isso, exatamente.

E. Ah sim. Não sabia que tinha um termo específico para test smell não.

P. Ok. Só reforçando o conceito. "Test smells são pontos fracos na estrutura do código de teste, tendo um impacto negativo na sustentabilidade da atividade de teste, pois os tornam complexos, difíceis de entender e modificar, prejudicando a repetibilidade, independência e estabilidade dos testes.” Você tem alguma dúvida, pois as perguntas seguintes têm a ver com test smell, dentro dessa rotina de criação e manutenção de teste.

E. Não. Pode seguir. Embora eu nunca tinha escutado o termo, faz sentido, porque na verdade eu encarava tudo como code smell, mas tem algumas estratégias sim, algumas diretrizes que eu sigo para os testes unitários. Embora eu nunca tenha visto o termo, provavelmente algumas coisas possa ser que eu já siga, enfim. Pode seguir.

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

E. Embora como eu falei, eu nunca tenha lido nada especificamente em relação a isso, eu sei algumas coisas que não deve ter no teste. É semelhante ao código principal. Esse tipo de coisa é identificado na hora do peer review. Na hora que você abre o pull request e o seu teste está com smells, no caso a ideia é que o reviewer vai identificar isso e vai solicitar alterações. No processo de peer review. A gente não tem algo automático que vai dizer que aquilo é um test smell, isso é feito pelo peer review.

P. Essa verificação de test smell ela ocorre da mesma forma para manutenção?

E. Sim. A gente não tem distinção. Para a gente é alteração no código a gente vai tratar sempre da mesma forma.

P. Para você, como seria possível prevenir/evitar a inserção de test smell no código de teste no momento que o teste é criado?

E. Essa é uma pergunta difícil, seria perguntar como evitar code smell. Eu não conheço, eu não sei se existe uma técnica, mas baseado no que eu sei, é mais uma questão do time ter conhecimento do que são esses smells e evitar fazer. Eu vejo mais como um alinhamento, uma coisa mais de leitura. Da mesma forma que a gente tem que ter uma leitura sobre code smells, sobre como evitar, como não fazer. O que são code smells para não fazer. Na mesma forma que os testes unitários teriam a mesma coisa e o processo de peer review é que ajuda isso, porque quando você cria um código (de software) ou um teste que tenha alguma coisa que code smell, test smell, o peer review vai lá e te fala "isso aqui não é interessante você fazer por causa disso, disso e disso," a tendência é que no próximo teste você não vai mais fazer isso. Então é o processo de peer review que guia a gente em tudo isso.

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

E. A única pergunta que eu gostaria de fazer é sobre sua pesquisa, não sei se eu poderia fazer. Estou curioso, não sei se você pode dar mais detalhes. Eu queria entender mais o que você está fazendo.

P. Posso sim. Agora que a entrevista chegou ao final, eu posso dar mais alguns detalhes sem prejudicar a entrevista. Minha pesquisa é focada em test smell no contexto industrial. Então a ideia da minha pesquisa é saber qual é o nível de conhecimento de profissionais a respeito desse assunto na indústria e como os test smells são tratados, tanto na criação quanto na manutenção para identificar alguma possibilidade de melhoria desse processo dentro da indústria. Por isso que eu perguntei como funcionava a criação do código de teste, a manutenção e depois como test smells se enquadrava dentro dessas etapas. Pois como você comentou, o profissional pode não conhecer o termo, mas de alguma forma ele trabalha com isso no dia a dia, então, por mais que ele não saiba que exista um termo específico chamado "test smell" de alguma forma ele pode tratar isso no seu dia a dia e a ideia é justamente obter essas informações e obter alguma forma de melhorar esse processo. É um tanto difícil pensar que é possível evitar que ele seja inserido no início, mas é o que eu venho buscando dentro do mestrado, identificar se existe alguma possibilidade de evitar essa inserção com alguma ferramenta ou basicamente o conhecimento dos profissionais.

E. É o mesmo caso do code smell. Como evitar adicionar code smells. Eu penso em duas alternativas, em duas coisas que você pode olhar. Uma é existe um catálogo de code smells que o pessoal tem e você tem que evitar fazer isso. Talvez se você tivesse um catálogo de test smells para isso e a outra coisa seria uma espécie de checklist que o código principal tem, alguns checklists que também são aplicados nos testes, porque no final das contas é tudo código. Então o checklist, essas ferramentas de link que analisam o código, ele analisa o teste unitário também. Algumas coisas referentes ao teste ele não vai avaliar, mas ele consegue avaliar alguma coisa no código de teste. Talvez, eu não sei qual o objetivo de vocês, fazer uma espécie de ferramenta de link de checklist, faria uma espécie de plugin para as IDES e aí ela faria essa verificação.

P. Já existe um catálogo em estudos, mas há uma demanda por implementação voltada para isso mesmo.

E. É porque é complicado. Isso acaba expandindo para outras coisas. Eu vejo como um dos problemas é que quando um código não é desenvolvido pensando nos testes acaba dificultando muito a criação dos testes. Eu fico pensando se talvez se houvesse alguma espécie de análise no código principal, para identificar se existe algo que vá dificultar na criação dos testes, por exemplo. Uma coisa que demorei um tempo para descobrir, os frameworks de injeção de dependência, não sei se você tem conhecimento disso, eles permitem você injetar as dependências via atributo. Você cria um atributo lá na classe, bota um object, isso em Java. Isso eu fiz muito no passado, porque é massa porque você chega lá, cria um atributo, bota um object e quando você vai usar o atributo ele vai lá e quando você cria a classe ele cria um atributo para você. Mas isso é péssimo para seus testes. Quando você precisa testar, digamos você tem uma classe de serviço que injeta uma classe que acessa no banco de dados. Quando você vai testar, como você faz? Você não tem como testar isso aí. Não tem como mochar. Hoje já se sabe que o correto é injetar isso no construtor. Você cria um construtor que recebe a dependência como parâmetro e você injeta. A injeção de dependência ela foi criada pensando nisso, na verdade. Para você injetar no construtor porque aí no seu teste você consegue injetar dependência também; mas se a pessoa já fez a classe sem pensar nisso, na hora de testar ela não vai conseguir. Aí ele vai ter que pensar em outras coisas. Eu fico pensando será que se talvez alguma ferramenta que no momento que o cara faça isso diga "isso aí não vai facilitar seu teste". Não é um test smell, mas é um indicativo que isso vai deixar seu teste difícil de testar. Não sei se sairia do escopo do seu trabalho

P. Um pouco, mas se o objetivo final for atingido, porque não? Não sei se daria tempo no mestrado, mas se eu estender para o doutorado, quem sabe?

E. Beleza

P. Muito obrigado. Suas informações foram muito importantes.