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. Não há respostas certas ou erradas ou desejáveis ou indesejáveis. Então 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 contigo. Garanto que todos os seus comentários permanecerão confidenciais e que no relatório final conterá os comentários dos participantes sem nenhuma referência aos indivíduos.

Entrevistado. Beleza

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

E. Eu sou desenvolvedor Android já faz cerca de aproximadamente 6 anos e foi onde eu comecei a entrar no mundo dos testes. No início, escrever teste parecia algo de outro mundo, uma coisa estranha, e a gente pensava, principalmente em desenvolvimento mobile, era muito comum teste ser apenas teste funcional, teste manual e eu acabei indo para uma empresa em que tinha a cultura de testes. Era uma empresa de consultoria em que essa empresa atendia grandes clientes, e o público, os clientes dessa empresa, eram clientes do ramo financeiro, como bancos, grandes bancos, como B, C, D, e dentro da empresa existia uma cultura muito forte de teste de software e foi nessa empresa que eu comecei de fato a colocar a mão na massa e começar a ler sobre teste, aprender boas práticas, aprender de fato como testar. Eu tive a oportunidade de trabalhar junto com especialistas de fato em teste. A empresa tinha uma área específica só de QA (Quality Assurance), então foi uma oportunidade muito boa para poder entender como funciona de fato a parte de teste de software e entender a importância, como de fato aprender a colocar aquilo no meu dia a dia e trazer essa parte de teste como algo de valor para o produto que estava desenvolvendo.

P. Quais foram suas fontes de aprendizado sobre código de teste? Basicamente essa empresa, teve outras fontes?

E. Sobre teste de software teve alguns materiais que eu li, teve um livro que eu li sobre JUnit e foi um livro que me ajudou bastante a entender, o nome do livro é “Pragmatic Unit test”. Mas foi um misto de livros, de vídeos, experiências, trabalhar com outras pessoas também que sabiam bastante sobre teste, pessoas que são especialistas na área de testes também; e eu costumo dizer que foi um misto de coisas e que todos os dias a gente está em constante aprendizado. Vários assuntos também. Alguns posts do Martin Fowler conseguiram dar um norte sobre metodologias, sobre tipos de teste, foram bem importantes.

P. Em quais linguagens você cria testes?

E. No meu dia a dia está em torno de duas linguagens: como eu trabalho com Android, hoje a linguagem oficial para desenvolvimento Android é o Kotlin e eu também trabalho com Java. O Java porque ele era a linguagem oficial para desenvolvimento Android, então eu venho dessa época e, um pouco mais de 3, 5 anos, a linguagem oficial se tornou Kotlin, então a gente só fez mudar de linguagem, mas continuamos a usar as mesmas ferramentas. Hoje uma das principais ferramentas é o próprio JUnit, o mesmo JUnit que a gente usava no Java.

P. Qual plataforma do seu projeto atual?

E. Kotlin e Java

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

E. Normalmente, eu vou tentar ilustrar um pouco do fluxo de desenvolvimento que eu acho que fica mais tranquilo da gente chegar onde começa os testes e até o momento: De fato, a gente executa e valida se os testes estão ok ou se não estão. Basicamente no meu dia a dia, se eu preciso desenvolver uma feature A, por exemplo, antes do desenvolvimento, na fase de planejamento, por trabalhar com mobile, tem certas coisas que sempre tem que estar envolvidas, como por exemplo tela. É muito comum a gente estar desenvolvendo aplicativo... normalmente está tudo baseado em tela e quando você começa o desenvolvimento, você já tem basicamente predefinido o que você precisa fazer, seja um fluxo ou alguma alteração de uma feature já existente, e aí na fase de planejamento a gente tenta analisar quais são os possíveis cenários, possíveis casos de uso que no dia a dia o usuário ele irá fazer. Por exemplo: se tem uma tela de login, o que o usuário vai fazer nessa tela de login. Quais são os possíveis caminhos. Ele vai digitar usuário e senha e clicar no botão, só que dentro desse clique ele pode inserir uma senha errada, ele pode deixar um campo em branco, e em cima disso a gente começa a montar os cenários de teste. Obviamente quando você começa a desenvolver, é uma prática que eu tenho, tem muita gente que escreve todo o código depois escreve todo o código de testes, tem gente que aplica o TDD e primeiro escreve o teste e depois começa a desenvolver, eu costumo fazer meio que em paralelo, eu desenvolvo um pequeno pedaço e escrevo um teste para aquilo que estou desenvolvendo. Sejam esses testes, testes de unidade, quanto teste final, que é um pouco mais complexo, falando do mundo Android, o teste de tela, que a gente chama de teste instrumentado. Esses são testes que são um pouco mais custosos porque eles dependem do device, porque o código que a gente precisa rodar, ele depende do framework do Android e aí tem um certo grau de complexidade e um certo custo para a gente executar um teste. Normalmente esse é um teste que acaba ficando por último porque ele acaba validando a integração da minha tela com alguns outros componentes, então, só resumindo... eu preciso, se eu tiver um fluxo com três telas, eu acabo desenvolvendo a primeira tela, o teste para aquela tela, a segunda tela, faço teste para a segunda tela... e de fato no final eu consigo validar a integração das três.

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

E. JUnit. Os testes que a gente tem com o JUnit atuam em todas as camadas da arquitetura. Normalmente a gente costuma seguir uma divisão de três grandes níveis da aplicação: Uma camada de apresentação, onde de fato tem as dependências do framework do Android, a parte de renderização, a parte de views, e a gente tem outras 2 camadas: tem a camada de domínio, que é a parte mais isolada onde tem as regras de negócio e a camada de dados. Nessas duas últimas camadas, normalmente os testes que a gente escreve é em cima do JUnit, porque são testes rápidos, a gente não depende do framework do Android, que é algo custoso para executar os testes. Já na camada de apresentação, que tem componentes visuais, onde depende de fato do framework do Android, aí você precisa ter ou um dispositivo físico ou um emulador para conseguir rodar aquele tipo de teste, aí a gente usa uma ferramenta chamada “Expresso” que é uma ferramenta que a Google criou para a gente conseguir realizar teste de tela no Android e essa é a principal ferramenta que está no nosso dia a dia para conseguir escrever esse tipo de teste além do JUnit.

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

E. Hoje no Android, não é bem um padrão, não chega ser um modelo acadêmico, alguma coisa assim do tipo, mas a gente costuma seguir um padrão criado por um cara chamado Jake Wharton e é um cara que ele era da Google. Ele foi responsável pela criação de várias coisas que a gente usa do Android hoje e ele criou um padrão chamado robot pattern. É uma forma de organizar o teste instrumentado no teste de UI, onde você consegue deixar seu teste mais organizado e desacoplado, separando "o quê" do "como". “O quê” é basicamente a classe de teste que você vai ler, consegue ler e sabe o que aquele teste está fazendo, só que você não sabe os detalhes da implementação. Por exemplo, seu teste tem um método que diz inserir um nome de um usuário, ou clicar num botão. Mas só tem a chamada do método, você não sabe o que precisa ser feito a nível de implementação para clicar naquele botão na hora do teste. A gente utiliza, eu falo em nome do time Android da empresa, a gente tem esse padrão como um guia para escrever os testes instrumentados e a gente tem algumas documentações e algumas ferramentas que a gente criou dentro da empresa para deixar a escrita de algumas coisas mais simples. Por exemplo, essa ferramenta Expresso, você consegue fazer qualquer tipo de teste: de tela, desde clicar num botão do menu até clicar no botão de back, fazer scroll, fazer swipe, você consegue fazer um monte de coisa. Só que a forma que você utiliza essa biblioteca ela é muito verbosa e pelo fato da gente estar utilizando Kotlin, o Kotlin tem várias ferramentas que nos ajudam a escrever as coisas de uma maneira mais simples. A gente criou uma biblioteca aqui na empresa que consegue abstrair toda a complexidade da escrita do expresso e a gente fornece maneiras mais simples para que os desenvolvedores possam utilizar essa nossa biblioteca nos testes e conseguir escrever os testes de maneira mais simples, escrevendo menos código e conseguindo manter os testes concisos, digamos assim. Escrever também em menos tempo, porque uma das coisas que a gente vê é comentário tipo "escrever teste gasta tempo" então a nossa ideia é construir ferramentas para que a escrita dos testes seja feita de forma mais simples.

P. Isso vale também para os testes de unidade ou vocês utilizam algum outro tipo de padrão para isso?

E. Para os testes unitários a gente não tem nenhum guia da empresa, a gente tenta manter a organização dos testes, seguindo os padrões de “Arrange, Action e Assert” ou “Give, When, Then”, como tem algumas recomendações do próprio Martin Fowler de como organizar os testes, organizar as features e tudo mais. E o que a gente tenta nos testes unitários, a gente olha sempre para o que tem de referência a nível de boas práticas. O que é recomendado de boas práticas para fazer teste, como um teste deve ser organizado. A gente costuma olhar para esses pontos no “code review”. Todo o código que sobe em qualquer projeto, a gente tem um fluxo de code review e é nos code reviews que a gente costuma garantir que um código ele não vai subir se ele tiver sendo escrito de uma maneira que foge do padrão; seja esse padrão algo que nós definimos dentro do time ou, por exemplo, um código que não está validando nada, é um teste que nunca vai falhar e a pessoa não percebeu e está subindo um código ali. Essa parte de olhar para a qualidade do teste, ela costuma ser feita dentro próprio time em um code review. A gente não tem ninguém externo que olha para aquilo ali. O próprio time que tenta cuidar do código de teste, tentando manter todo o código que é “mergeado”, aquele código tem que atender aos padrões predefinidos.

P. Como é feita a verificação de qualidade do código de teste criado, por exemplo: verificação de escrita, alguém revisar o teste depois de feito, verificar compatibilidade com o requisito ou classe de produção...?

E. Temos vários times hoje, o time Android ele está chegando a 40 desenvolvedores Android e é o que a gente chama de "skill android". Esses 40 “devs” compõem a skill android. Dentro da empresa, cada desenvolvedor Android atua em uma squad diferente. Normalmente tem 2 desenvolvedores na mesma squad, mas são várias pessoas que trabalham em coisas diferentes, só que na hora de “mergear” qualquer código, a responsabilidade é no nível de skill. Então qualquer código que é “mergeado” no code base é de responsabilidade da skill fazer os code reviews. A gente tem uma ferramenta que roda no git hub... toda vez que alguém abre o pull request, essa ferramenta tem um algoritmo que olha dentro do time e randomiza quem vai fazer a revisão do pull request. Normalmente é alguém que não escreveu o próprio código e a gente consegue dessa maneira selecionar 3 ou 4 pessoas para fazer a revisão do mesmo código e enquanto essas pessoas não aprovam, o merge fica bloqueado. A gente tem uma documentação, a nossa wiki com várias guidelines, tanto para escrita de código de produção, quanto a parte de testes, e basicamente é o que as pessoas precisam olhar, se aquele código que está subindo atende aos padrões estabelecidos pelo time, que são padrões que estão escritos na nossa wiki, quanto padrões pré-estabelecidos no mercado e na academia. Por exemplo, tem coisas que são muito básicas e que não precisam ser escritas na nossa wiki. Um exemplo disso é manter o código no padrão da formatação recomendada pelo Kotlin. São coisas que a gente não escreve na nossa wiki, mas são coisas que são cobradas no code review, por exemplo. Dessa forma a gente consegue manter que a skill como um todo consiga revisar o código de todo mundo. Eu trabalhei em empresas que tinham uma ou duas pessoas em que eram centralizadas as revisões de código, trabalhei em empresas que tinha um time específico que cuidava da revisão de código, mas eu confesso para você que hoje o cenário que estou tem funcionado bem melhor, em que todo o time revisa o código de todo mundo. A gente tem pessoas de nível júnior até nível sênior e o júnior pode simplesmente analisar o código do sênior e fazer comentários que o cara por ser sênior não significa que ele vai estar escrevendo o código da melhor maneira e o júnior pode achar um problema que o sênior escreveu. É basicamente essa ideia que a gente tem dentro da empresa como esse espírito de time. Todo mundo é dono do produto e todo mundo precisa participar de tudo porque o trabalho é do time e não é da pessoa ou do time A ou do time B, o trabalho é de todos. Isso tem ajudado a manter a qualidade do nosso código e até a disseminar conhecimento porque uma das coisas interessantes do code review é disseminar conhecimento. Hoje a gente tem um time que é bastante participativo, a gente tem toda as segundas feiras uma reunião do skill android em que a gente compartilha o que a gente está fazendo no nosso dia a dia porque, como é um time grande e cada um está em uma frente diferente apesar de trabalhar no mesmo produto, você acaba não sabendo níveis de detalhes que as pessoas estão trabalhando; muita das vezes nem conhecendo também features novas que a empresa vai lançar. E é interessante para fazer esse alinhamento, quanto alinhamentos técnicos também. A gente tem trazido não em todas as semanas, mas de vez em quando, uma palestra que alguém do próprio time traz sobre arquitetura, sobre boas práticas, sobre teste, sobre code review, sobre vários temas que são interessantes para o nosso dia a dia e muita das vezes e só um pequeno grupo do time conhece.

P. A ferramenta que vocês utilizam, que faz o compartilhamento quando o código é submetido com outras pessoas avaliarem é uma ferramenta desenvolvida internamente pelo skill?

E. Não, é uma ferramenta chamada “Pull Panda”, é uma ferramenta que o Git hub comprou recentemente e o papel dela é toda vez que alguém faz um pull request, ela olha uma lista de pessoas e escolhe aleatoriamente pessoas para revisar aquele código; a gente coloca alguns critérios para aquele código seja “mergeado”. Toda vez que é feito um pull request, é disparado para uma plataforma que a gente tem que pega aquele código que a pessoa quer “mergear”... é o nosso pipeline de integração contínua. Ele pega aquele código e roda todos testes. Se todos os testes passaram, ele habilita um dos “checks” do Git hub, que é como se estivesse ok para “mergear” o código, pois não quebrou nada. Se quebrou alguns testes, automaticamente aquele pull request é bloqueado. Assim que o código passou, foi tudo ok, ele é integrado. Depois que ele é integrado, a gente roda novamente esse pipeline de testes numa esteira mais completa, rodando testes unitários e testes instrumentados. Hoje, a gente não está na realidade de conseguir rodar, por exemplo, os testes de tela todas as vezes que alguém abre o pull request, mas é algo que a gente está trabalhando ultimamente para conseguir fazer isso com mais frequência. A gente faz isso duas vezes ao dia e, se algum teste quebra (a gente usa o slack), a gente já recebe uma mensagem no slack dizendo que "a parte tal do código que foi integrada fez quebrar um teste tal". A gente consegue ter esse relatório e o time já é mobilizado para poder subir uma correção. Uma pequena alteração leva em torno de 40 minutos para conseguir rodar o pipeline inteiro e a gente tem um time de 40 desenvolvedores. A gente tem uma infraestrutura que suporta toda essa parte de integração continua e é muito oneroso e demorado. Dentro do nosso time, é algo que a gente tem difundido é que antes de subir qualquer código, abrir qualquer pull request, que os próprios desenvolvedores rodem os testes unitários e rodem também os testes de tela na própria máquina deles antes de subir um pull request.

P. Passando para o processo de manutenção do código de teste, me conte como este processo é feito na empresa?

E. A parte de manutenção do código de teste ela está sempre associada a manutenção de um código de produção, porque normalmente a gente costuma voltar em um código (de teste) já escrito se você está atuando em algo no código de produção que aquele teste está validando. Isso ocorre quando a gente precisa corrigir algum bug, para corrigir um bug X, a gente volta nos testes e executa aqueles testes, as vezes precisa voltar naquele teste, refatorar o teste se alguma coisa quebrar ou mudar, mas fora esse processo, não existe um processo definido que a gente fique olhando para manutenção especificamente do código de teste. Normalmente o que acontece é o seguinte: se tem alguma alteração no código de produção, eu olho o código de teste. O cenário que a gente olha para os testes que já estão escritos é só se mexer em algum código de produção que o teste começou a quebrar, aí você olha. Do contrário, só testes novos são incluídos.

P. Me conte o que você conhece sobre test smell.

E. Eu conheço algumas coisas. Eu considero que são más práticas, más escolhas que você faz no teu código de teste que ele acaba dificultando a manutenção e a evolução daquele código de teste.

P. Como você conheceu este assunto?

E. Através de uma pessoa que pesquisa sobre o assunto. É algo que é bem recente para mim e as pessoas que trabalham comigo a maioria não tem conhecimento sobre test smells.

P. Você possui alguma dúvida sobre esse tema?

E. Não, mas talvez eu não tenha conhecimento suficiente para ter alguma dúvida.

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

E. Pela minha experiência e pela minha realidade não são tratados. É muito raro a gente escrever o código de teste pensando em test smell. Eu acho que o principal fator é falta de conhecimento de fato. Eu conheço alguns test smells. Muitas das vezes só depois que eu escrevo o código de testes que eu vou olhar "será que eu preciso refatorar porque tem esse problema?", mas no momento da escrita não. Eu acho que se você perguntar a 90% ou mais do time, a maioria, vai falar que não conhece test smells.

P. Me conte como test smells são tratados no processo de manutenção do código de teste.

E. Pessoalmente, normalmente é no término da escrita ou no término da escrita da classe, porque eu sempre vou conferindo, por exemplo: eu tenho uma classe que tem que validar o comportamento da unidade classe X, eu tenho um cenário A, B e C. Eu volto conferindo se realmente os cenários que eu preciso estão sendo atendidos pelo que escrevi, e é nessa etapa em que eu consigo identificar os smells no meu código.

P. Você tem esse cuidado quando faz essa verificação em code review ou quando está realizando a manutenção do código de teste?

E. Nos code reviews, eu posso falar que bem recentemente, eu comecei a comentar "isso seria interessante fazer dessa maneira porque você está inserindo um test smell tal" e coloco um link para a pessoa ver porque que tem um problema ali e tudo mais. Na manutenção tenho esse cuidado sim, acaba caindo no mesmo cenário em que eu reviso algo e acaba sendo nessa revisão.

P. Para você, como seria possível prevenir/evitar a inserção de test smell no código de teste no momento da criação do código de teste?

E. Eu posso fazer uma comparação com code smells. Para code smells, existe uma ferramenta que a gente usa que ajuda muito no momento da escrita. A ferramenta, para linguagem Kotlin, se chama "Detekt". É uma ferramenta que você consegue utilizar ela no Android Studio, e tem o plugin que você consegue habilitar essa detecção de code smells no momento que você está escrevendo. Se tivesse uma ferramenta que fizesse isso no momento que você está escrevendo aparecesse um balãozinho, um warning: "isso aqui tem um test smell", ajudaria bastante. E outro fator também é a falta de conhecimento. Code smells eu acho que mais pessoas conhecem, mas test smells, eu acho que poucas pessoas conhecem, apesar de ser um problema no mesmo nível de code smells.

P. Você tem alguma pergunta para mim ou quer acrescentar alguma informação ou ainda tem alguma sugestão para melhorar nossa entrevista?

E. Não sei, eu acho que não.