+Pesquisador. Muito obrigado por ter aceito participar da entrevista. Ela tem como objetivo compreender os processos de criação, manutenção e verificação da qualidade do código de teste de unidade. Não há respostas certas ou erradas, desejáveis ​​ou indesejáveis. Dessa forma, senta-se à vontade para dizer o que realmente pensa. Ok?

Entrevistado. Ok.

P. 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 e no relatório final conterá os comentários dos participantes sem nenhuma referência aos indivíduos. Ok?

E. OK

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

E. Eu comecei a começar a trabalhar com testes manuais por volta de 2000, foi quando eu comecei a trabalhar com desenvolvimento de software e durante muito tempo, durante os 5 primeiros anos era o que tinha. O nível de automação era quase inexistente. Por volta de 2005, 2006 que eu comecei a ouvir um pouco a respeito dos testes. Eu comecei de uma maneira muito incipiente e acho que foi por volta de 2009 que nós na empresa na época, começamos a ler um pouco a respeito e a gente começou a automatizar testes de uma maneira mais profissional digamos assim. De maneira sistemática. Como a literatura recomenda. Primeiro contato foi com teste de unidade e o principal truque na época foi ter previsibilidade. Nós tínhamos componentes de cálculo. Na época eu já trabalhava no setor educacional, nós tínhamos uma grande variabilidade de casos de uso de testes. Então tinha centenas de modelos de cálculos de notas diferentes, dados os nossos clientes. Éramos uma software house na ocasião, e a gente tinha regras muito peculiares de cálculo de notas. O que a gente fazia era automatizar esses testes em motor que calculava nota para garantir que qualquer alteração não ia quebrar uma regra existente. Não só naquele momento, mas o histórico também. Desde então a gente vem expandindo o uso de testes tanto na área de testes de unidade, testes integrados, testes de carga, e testes de navegador, que fazem basicamente simulam a resposta de um setor http para ver se o comportamento da aplicação cliente está funcionando conforme esperado.

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

E. Foi por bastante literatura. Na época tive contato com Martin Fowler. Ele tem uma série chamada pragmática em teste que foi um livro bem inspiracional, os livros do Andy Hunt. Da turma que assinou o Jar Manifest foram fontes tantos livros quanto os blogs foram fontes de bastante inspiração e aprendizagem, mas o livro que me inspirou bastante foi o Pragmatic Unit Testing. Esse foi uma fonte muito rica de informações.

P. Em quais linguagens você cria testes?

E. Java, .NET, Kotlin. Acho que essas são as principais. Na verdade dá para usar em qualquer linguagem, mas as que a gente usa no dia a dia geralmente Javascript, Java, .NET e Kotling.

P. Todas 4 você está trabalhando atualmente?

E. Sim.

P. Como é seu processo de criação de teste de unidade?

E. Tem dois principais contextos, pelo menos. O primeiro é quando a gente está criando uma aplicação do zero, então geralmente a gente opta por TDD. A gente está começando uma aplicação do zero a gente não só tenta fazer, como a gente recomenda para os desenvolvedores e parceiros, ou para aplicações existentes, quando a gente quer refatorar a aplicação vigente ou quer fazer uma manutenção no legado, geralmente, dependendo no nível de complexão, um cadastro, a gente assim nesse ponto a gente é um pouco mais pragmático, se for um cadastro que não tem nenhuma regra, você pega dados que estão ali na tela, num sensor, no dispositivo. Existe isso numa base, a gente não pratica testes para 100% do nosso código, mas tem uma regra, começou a ter uma regra, complexidade, a gente quer criar uma rede de proteção para os desenvolvedores no futuro conseguirem manter aquele software ou garantir que aquele software vai continuar funcionando, a gente pratica. Então assim. Respondendo mais diretamente sua pergunta, quando a gente está trabalhando num serviço ou num novo sistema, a gente pratica TDD, quando a gente vai fazer manutenção em serviços existentes que não possuem código, para manter esse código legado a gente primeiro cria os testes, até para já aproveitar para refatorar o código e depois fazer a manutenção.

P. Existe na empresa ou na equipe, em particular, algum fluxograma, template documentado que padronize esse processo de criação de código?

E. Nós temos documentos de referências. A gente tem um documento de referência para code review, para revisão de código. O pull request ele passa pelo processo de revisão, e a gente tem, na verdade 3 conjuntos de documentos. Esse para revisao de código, um que é o guia de referência arquitetural, que diz respeito as tecnologias que a gente utiliza devem ser utilizadas em cada camada e um terceiro que é o nosso padrão de testabilidade. Inclusive na empresa nós temos em cada squad um profissional de testes e nós temos um charter de testes que discutem questões de tendências, novas ferramentas, novas maneiras de efetuar testes.

P. Você falou um pouco sobre a próxima pergunta, que é justamente esse processo de verificação da qualidade do código de teste. Você pode me falar mais um pouco?

E. Posso. Como funciona hoje o nosso. A gente está nesse momento, faz quase 4 anos que eu trabalho para essa empresa atual. Quando nós começamos o processo de transformação digital a uns 2 anos, a área de era inexistente, assim como outras áreas, de usabilidade, de arquitetura de software, e nós tínhamos um departamento de teste mas de uma maneira muito clássica: departamento de testes para testes manuais. Desde então nós dividimos o departamento de tecnologia em squads, cada squad tem ali o seu tester, um analista de testes que tem, ele participa desde o processo da concepção, desde a planning da sprint até o, vai acompanhando a execução do sprint e ali ele já tem as chances de planejar os testes. Hoje a gente trabalha com algumas camadas de teste, como comentei inicialmente, então o desenvolvedor é responsável pelos testes de unidade. O analista de testes ele tem alguns times que praticam TDD, mas ele faz a parte de testes integrados e aqui varia um pouco da natureza da aplicação, se é uma aplicação web, se é uma aplicação mobile, se é uma aplicação predominantemente uma API, tem alguns consults técnicos distintos como a parte de APIs, a gente tem por um outro suite de testes, e ele também assessora de maneira mais integrado nos times, os testes de carga. Boa parte dos testes de unidade integrada são executadas automaticamente pelo Pipe de continuous integration e o processo de continuous delivery, então assim, cada commit roda o processo de build, ou o processo de análise dos scripts se a linguagem não é compilada e roda o conjunto de teste de unidade. Tem alguns squads que já praticam o continuos deployment delivery e aí o pipe de testes integrados e APIs voltam automaticamente, quando isso não acontece a gente tem processo de nighting building, de builds noturnos que fazem o depoymment dos testes e roda uns testes automatizados nessa forma.

P. Você comentou sobre um code review, então quantos desenvolvedores fazem a revisão de um código submetido?

E. Isso varia muito, mas para ter uma resposta absoluta, 1. Geralmente 1. A gente tem alguns contextos, por exemplo pagamentos, é uma parte muito crítica a gente coloca mais de uma pessoa para revisar, mas no geral é uma pessoa.

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

E. Para os testes de unidade a gente usa o JUnit. Para aplicações .Net, o NUnit e no caso de android a gente usa do Mockito.

P. Passando para o processo de manutenção do código de teste de unidade, como ele é realizado?

E. A gente tenta praticar e institucionalizar a cultura de código e teste como códigos de primeira classe. A gente tenta pregar que o código de teste é tão importante quanto o código da aplicação. Então a manutenção dele é, o que nós pregamos e tentamos praticar é que ele seja tão importante quanto o da aplicação. Então isso varia um pouco. Óbvio que temos times que estão mais maduros, menos maduros com relação aos testes, mas geralmente é quando a gente encontra um bug. O que é que a gente recomenda, é que a pessoa primeiro crie um teste para provar que aquele bug realmente acontece, quando ele for corrigido o teste vai passar, não só aquele teste vai passar como os demais testes também deveriam passar. E no caso de você fazer uma grande mudança, sei lá, vai refatorar, vai expandir o sistema de maneira que o isso aí vá quebrar ou descontinuar alguns testes. Tentar fazer com que aqueles testes permaneçam vivos. Desde que façam sentido. Tem um outro ponto que a gente usa como débito técnico. Vamos dizer que você fez um sprint e por alguma necessidade você não conseguiu fazer todos os testes ou cobertura de testes naquele determinado serviço, aquela funcionalidade, não está adequado. O que a gente tenta sugerir e isso é um exercício interessante porque a gente já tem alguns POs que não são técnicos, mas começam a priorizar a parte de testes porque a gente está conseguindo convencer o time de negócio a importância dos testes. Numa manutenção daquele serviço, daquele software, você acaba mantendo a velocidade do time a médio e longo prazo, quando você tem uma boa suite de testes. Um outro cenário onde a gente desenvolve testes é aumentar a cobertura de testes ou pagar débito técnico, a gente considera hoje que quando você entrega um sprint, um novo serviço, uma única série não tem teste aquilo vira um débito técnico e a gente tem dedicando o tempo do time para pagar esses débitos técnicos.

P. O que você conhece sobre test smell?

E. Não conheço test smell, mas se for análogo ao code smell, aí me corrija se eu estiver errado, são características que demonstram que aquele teste não está seguido alguns padrões ou tem alguns problemas estruturais.

P. Isso. Exatamete. Test smells eles descendem dos code smells, apesar de ter uma suite específica de tipos de test smells que diferem dos code smells.

E. Eu não conhecia a expressão não

P. Eu vou te passar o conceito especificamente, apesar de você já ter feito essa analogia com code smell. Caso você tenha alguma pergunta para me fazer a respeito do conceito a gente conversa, porque as perguntas a partir de agora vão estar relacionadas a esse tópico, ok?

E. Pode ser. Vamos lá

P. “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.”

E. Tá.

P. Como você tinha comentado mesmo, em relação a estrutura e são coisas que ocorrem na estrutura que acaba tendo impacto negativo na atividade de teste. Devido a complexidade, devido a dependência, a problemas de repetibilidade e assim por diante.

E. Mais ou menos como code smell, só que no contexto dos testes.

P. Exato.

E. Legal

P. Eu vou te mandar 2 exemplos de test smells, só para você ter uma ideia a respeito disso, mas tem a ver com o que a gente está conversando. Um chamado Empty Test smell e o outro é o Conditional Test Logic smell. O primeiro que chegou é o Empty, eles são testes que eles não vão ter nenhum script executável. Então ele não vai estar executando aquilo que ele foi pensado para ser executado.

E. Está tudo comentado o código, não é?

P. Exato. Então as vezes para identificar porque que ele está comentado ao invés de ter sido excluído ou estar sendo executado. Isso também dificulta na manutenção dessa atividade de teste. O outro exemplo é o chamado Conditional Test Logic smell. Ele é considerado um smell por ter estruturas de condição e de repetição dentro dele, o que ao depender do desenvolvedor pode dificultar a compreensão desse teste, como também uma vez que esse teste falhe ele vai dificultar a saber exatamente em que momento da execução desse teste ele falhou.

E. Legal. Você tem desvios condicionais aqui dentro que dependendo do conteúdo do seu map, ele vai ou não executar... entendi! Legal.

P. Então assim, existem test smells relacionados a asserts, relacionados a dependência, por exemplo, ao depender da ordem que você execute os testes um vai passar outro vai falhar por conta da dependência de recursos. Tem smell relacionado a duplicidade, seja de um código em particular ou até da própria classe inteira, então são testes que impactam na sustentabilidade da atividade de teste devido a sua estrutura que acaba se tornando mais complexa, difícil de compreender ou de realizar alguma alteração. Ok?

E. Legal. É bem análogo ao conceito de code smells, mas aplicado ao universo de testes. Mas bacana.

P. Tem alguma pergunta, alguma dúvida sobre isso?

E. Não. Tranquilo. Eu coloquei aqui para depois pesquisar mais. Eu fiquei interessado nesse assunto. Eu vou atrás de uma lista. Assim como code smell você tem guia de referência, uma lista, eu vou atrás desses tests smells para conhecer um pouco mais. Eu li um livro, o Pragmatic Unit Testing, mas eu lembro de ter visto algumas dicas de testar funções de fronteira, testar usabilidade, testar dependência. Evitar ao máximo dependência de test file system, rede, http, mas eu não lembro, possa ser que ele mencione isso no livro mas a minha memória não esteja funcionando tão bem.

P. Eu vou passar para a próxima pergunta, ok?

E. Vamos lá.

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

E. O que a gente tenta, não sei se eu vou conseguir responder diretamente, até porque o conceito é relativamente novo para mim, mas o que a gente tenta fazer é garantir que o teste o comportamento de uma classe e não um método em si, e que ele tenha lá um pequeno número de asserts, essas questões dos Ifs, do teste estar testando mais de um comportamento a gente tenta sugerir que isso seja quebrado em mais de um teste. A gente evita de fazer essa questão por exemplo dos comentários. A gente tenta não fazer porque a gente sugere que "se o teste não faz mais sentido, apague" porque ele não deveria estar ali. E com relação aos ifs que você colocou, que eu lembro que tem dois ifs condicionais, a gente pede para que isso seja evitado para garantir previsibilidade dos testes e não depender de agentes externos, sejam dados, os mocks que você cria.

P. Pensando mais na questão conceitual, por exemplo, evitar dependência, manter uma repetibilidade dos testes, existe algum tipo de cuidado durante a criação dos testes para que isso ocorra?

E. Existe. Assim, cuidado técnico. O código de teste ele passa pelos nossos processos de SonarQube, de revisão, assim como qualquer outro código. A gente tenta disseminar a cultura de que o código de teste é tão importante quanto o código da aplicação. Então assim, se nessa regra que eu tenho lá no SonarQube para o código da aplicação ela se aplica ao código de teste de unidade. Isso a gente tenta pregar e eventualmente nos code reviews a gente pode pegar isso. Mas se existe uma maneira sistemática digamos assim, hoje nós não temos.

P. Por curiosidade, você mencionou o Sonar. Durante a verificação do Sonar, ele faz alguma verificação de smell?

E. Eu não lembro de cabeça. Eu acho que sim, mas eu não lembro. Eu sei que algumas questões do nosso padrão de nomenclatura, ela está contemplada no SonarQube, mas eu não lembro se tem alguma questão do code smell. Eu posso ver depois e te falar, mas eu não tenho de cabeça isso.

P. Se você puder fazer isso eu agradeço.

E. Posso sim. Depois você me manda uma mensagem só para ter certeza que eu não vou esquecer e aí eu respondo para você. Acho que alguma coisa existe. Acho que vi uma coisa parecida na confirguração do SonarQube.

P. Esse cuidado que vocês têm em relação a criação do código, em relação a smell, existe também durante a manutenção? Ou seja, se por algum motivo vocês encontrem algum teste com smell? Vocês têm essa verificação, se tem alguma dificuldade de compreensão, se existe algum tipo de complexidade. Isso é tratado na manutenção do código de teste?

E. Sim. Tanto no código de teste quanto no código da aplicação. É que no código da aplicação isso de um certo ponto já é mais natural, né? Existe a tendência que as pessoas, as empresas, on indivíduos acabam tratando o código de teste não como um código importante, mas a gente preza, vou dizer novamente, é a terceira vez que estou falando sobre isso, que o código do teste é importante, então numa eventual manutenção de produção ou se vai criar mais testes para um determinado serviço porque você está expandindo aquele serviço aumentando a cobertura de testes você percebe que aquele código não está legal seja por um test smell ou a nomenclatura, é feita a manutenção sim.

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

E. Eu acho que assim, prevenir ou evitar que esse código seja criado ou um sinal ou alerta para o desenvolvedor ou para o analista de testes de que ali tem um problema?

P. Se você puder pensar nas duas formas ajuda bastante.

E. Eu acho que um plugin a lá no Visual Studio ou Android Studio, na IDE Java que o desenvolvedor esteja utilizando, ou mesmo o Visual Studio Code, um plugin seria um bom warning sense, um sinal de alerta. E regras no NCover, para quem não trabalha com SonarQube ainda ou o próprio SonarQube que sinalize isso acho que seria uma boa maneira de sinalizar que tem um problema de manutenção futura.

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

E. Não. Eu gostei bastante da entrevista. Assim me alegra muito, eu acompanho o desenvolvimento dessa área de testes a bastante tempo. Quando testes não eram um tópico acadêmico ainda, dos cursos de graduação em relação a esse conceito, a necessidade e importância dos testes. E hoje eu vejo isso, eu vejo isso com muito bons olhos. Eu sou da época que eu tive que convencer muita gente da necessidade de testes da necessidade de continuous integration, de enfim, agilidade e fico muito contente sabendo que tem bastante gente estudando. Eu tenho um amigo que estuda testes a muito tempo e eu fico muito feliz em saber que tem bastante gente estudando e expandindo essa área que é muito importante, que já passou por tempos difíceis de convencimento, principalmente dentro das empresas.

P. Muito obrigado. Como tinha mencionado antes da gente iniciar a entrevista, eu já tenho experiência de mercado na área de testes, mas numa outra área, não em teste de unidade. E agora com o mestrado estou tendo a oportunidade de abordar essa outra área que está sendo bastante interessante e tem agregado muito em termos de conhecimento. Então essa pesquisa ela tem me aberto várias possibilidades, não só acadêmicas, mas de pesquisa da indústria em si e os resultados da pesquisa só tem a agregar não só academicamente, mas o meu pensamento quando voltei meus olhos para a indústria para que esse pensamento não fique só na academia, que isso consiga chegar a indústria de alguma forma.

E. Bacana. Muito bom. Qual a sua área de teste especificamente?

P. Minha pesquisa ela é focada em test smell. Por isso que da metade para o final das perguntas elas são focadas em test smell. A gente não está focando especificamente em uma linguagem porque de certa forma como a indústria existem profissionais de diferentes áreas e que o conhecimento sobre o assunto, por mais que não conheça o tema, pode ser tratado de diferentes formas independente da linguagem, a gente quer pegar esse cenário da forma mais global possível e numa próxima etapa talvez fechar em uma linguagem específica por exemplo, mas a princípio o pensamento é no macro para ver se a gente consegue desenvolver algo que seja de certa forma genérico para todas mas que consiga atingir o mesmo resultado.

E. Legal. É bem promissora, é muito promissora. Porque esses livros de padrões e alguns deles que eu nem mencionei, eles são atemporais. Tem alguns deles que são bem antigos e continuam valendo até hoje. Novas linguagens surgiram de um tempo para cá e eles continuam valendo. Tem mais um livro que eu li que eu lembro que gostei muito, que foi do... eu ouvi o podcast dele esses dias... é que eles não estão aqui, eu tenho boa parte deles fisicamente mas estão na empresa. Eu não vou achar ele aqui.

P. Não tem problema. As informações que você me passou elas já ajudam bastante. Foram bem detalhadas e com certeza elas vão contribuir bastante com minha pesquisa. O que eu vou te pedir, e que vou te dar um lembrete amanhã é em relação ao Sonar e também, se você puder, isso não é fundamental para pesquisa mas para mim seria interessante caso eu tivesse acesso... Você falou que tem um documento de referência para o code review. Se for possível fazer esse compartilhamento comigo seria bem interessante para mim.

E. Pode deixar. Se não tiver nenhum impeditivo junto a empresa e eu acho que não, eu compartilho com você. Mas eu te respondo amanhã. Talvez esse demore um pouco mais para verificar. Fazer a verificação do SonarQube é rápida, mas a verificação desse documento, não creio que tenha algum problema, mas eu posso eventualmente remover alguma informação que identifique a empresa e te passo, mas eu vou averiguar.

P. Tranquilo. Como eu falei, não é algo mandatório para minha pesquisa, é algo complementar, então seria muito interessante se eu tivesse acesso e óbvio, se você puder compartilhar. Quanto mais a empresa investe em tecnologia ela investe em confidencialidade. Se você não puder compartilhar, não for possível, eu obviamente compreendo, até a empresa que eu trabalhei eu não posso nem citar o cliente ao qual trabalhava durante um tempo, quem dirá documento, informações. Então eu compreendo bastante esse processo.

E. Tranquilo.

P. Muito obrigado. Eu vou finalizar a gravação.