Módulo 8 - Atividade Individual

Validação e Verificação

Wellington M. Espindula

Maio de 2021

1. Considerando abordagens de Verificação e Validação (V&V), qual a diferença entre abordagens estáticas e dinâmicas? Escolha duas abordagens de V&V (uma estática e uma dinâmica) e indique como cada abordagem poderia ser incorporada em um processo de desenvolvimento de software incremental.

Resposta: As abordagens de V&V **estáticas envolvem a validação e inspeção dos artefatos sem a execução do produto**; dessa forma, nas **análises estáticas se fazem inspeções de software** (usando por exemplo *linters* ou Analisadores Estáticos de Programa) . Já na análise V&V **dinâmica se faz necessária a execução do código** (ou trechos de código) de forma a testá-lo **almejando descobrir novos erros, falhas ou *bugs* em nível de execução**, para tanto pode-se utilizar testes de software de diferentes níveis.

Em processos incrementais, faz-se imprescindível o uso de V&V ao longo do desenvolvimento bem como no fim do desenvolvimento de cada um dos incrementos. Para tanto, abordagens estáticas como inspeção de software através de *code reviews* juntamente com análise estática de código com uso de ferramentas como *linters* podem e devem ser utilizadas ao longo do processo de desenvolvimento de cada incremento. Dito isso, **a revisão moderna de código é um processo informal a ser utilizado a fim de fazer revisão regular do código, o aperfeiçoando e compartilhando conhecimento entre os desenvolvedores; sendo assim, é uma abordagem que se encaixa bem ao processo de desenvolvimento incremental.**

Já abordagens dinâmicas como **testes automatizados também são bem-vindas no modelo incremental, dado que podem ser incrementados ao passo que funcionalidades são implementadas.** Testes unitários, por exemplo, podem ser realizados juntamente ao processo de desenvolvimento de cada módulo e serve também como documentação. Portanto, ao longo do processo do desenvolvimento de um incremento é importante também focar no desenvolvimento de testes unitários e de integração. Já no fim de uma iteração, faz-se necessário testes de sistema e aceitação a fim de homologar e validar de acordo com seus princípios e requisitos dos usuários.

Por fim, existem diversas ferramentas (CI/CD) que existem para auxiliar desenvolvedores nesses casos, como, por exemplo, o Gitlab que fornece a possibilidade de integrar os processos estáticos e dinâmicos de V&V e realizar *code reviews* em códigos, controle de versionamento, análises estáticas de código, rodar testes automatizados, construir os projetos e afins.

2. Quais são as dimensões do teste de software e o que caracteriza cada uma? Cite um exemplo de cada dimensão.

Resposta: As dimensões de teste são as seguintes:

- **Técnicas de Teste:** Versará sobre como os aspectos do software ou componentes serão testados; Dessa forma, temos **(*i*) Teste Caixa-Preta**: Verificará a funcionalidade, sendo assim, se para as entradas do módulo a saída gerará os resultados esperados; **(*ii*) Teste Caixa-Branca**: Fará asserções acerca da estrutura do software.
- **Níveis de Teste:** São os estágios ou estratégias que estão sendo utilizados para testar o software. São estes: **(*i*) Teste de Unidade**: Fará asserções sobre comportamentos e estruturas de um determinado módulo; **(*ii*) Teste de Integração**: Fará asserções sobre a interligação entre os módulos; **(*iii*) Teste de Sistema**: Testa todos os componentes do sistema e replicação em tempo de execução; **(*iv*) Teste de Aceitação**: Verifica aceitação dos requisitos do usuário.
- **Dimensão de qualidade:** Irá determinar os atributos que estão sendo o foco do teste. Alguns exemplos destes são: **(*i*) Teste Funcional**: Este irá testará se as funcionalidades estarão funcionando conforme os requisitos; **(*ii*) Teste de Segurança**: Verificará potenciais falhas de segurança que ponham em risco os usuários do sistema, seus dados sensíveis, o sistema em si, entre outros; **(*iii*) Teste de Stress**: Verificará potenciais máximos ; **(*iv*) Teste de Desempenho**: Versará sobre tempo de execução de tarefas; **(*iv*) Teste de Usabilidade**: Versará sobre os testes qualidade de uso do ponto de vista do usuário.

3. Considere uma aplicação de software com ampla cobertura de testes unitários automatizados. Nesse caso, é necessário realizar testes de integração, uma vez que grande parte das unidades do sistema já foram testadas? Por que os testes de integração são necessários?

Resposta: Sim, mesmo um código tendo alta cobertura em termos de testes unitário, isso **significa apenas que cada componente individual está com alta cobertura de testes**, isto é, **não significa que as interações entre os componentes do software estarão bem testadas**. Dessa forma, os testes de integração são necessários para fazer justamente essa testagem inter-modular.

4. Quando o desenvolvimento de software adota a prática de TDD, são necessários testes de integração e testes de sistema? Por que?

Resposta: Mesmo no TDD ainda faz-se necessário testes de integração tendo em vista que mesmo que os componentes e seus testes tenham sido previamente bem pensados e executados, nada garante que na orquestração dos componentes não houve um erro não previsto em cada unidade modular.

5. Por que mudanças são inevitáveis em sistemas de software? Indique uma forma de diminuir a degradação através da mudança.

Resposta: A vida muda, o mundo muda e o que os usuários precisam do software também muda. Para Heráclito, a mudança é inerente à experiência humana. Dado que o software é volátil, é de se esperar que ele tenha que mudar e se adaptar às novas expectativas dos usuários sobre eles. Portanto, ele deve portar novas *features*, remover outras, corrigir erros e falhas, se portar à novos dispositivos, ser compatível com outros sistemas/dispositivos, se adaptar à novas legislações. Ou seja, para um software se manter ativo, faz-se necessário que ele continue buscando se aprimorar e amadurecer. Por conseguinte, a fim de diminuir a degradação, é de suma importância manter uma rotina responsável de manutenção

através abordagens de V&V constantes em conjunto à refatoração e redesenho de trechos do sistema que estejam tendendo a se tornarem legados e obsoletos. Para, desse modo, manter o software com uma arquitetura limpa e extensível (com baixo acoplamento e alta coesão) e mantendo os melhores padrões de implementação.