**Anotações – 01/03/2024**

**Conferência da OTAN (Alemanha, 1968)**

Primeira vez que termo “Engenharia de software” foi utilizado.

“Certos sistemas estão colocando demandas que estão além das nossas capacidades... **Estamos tendo dificuldades com aplicações**”.

Mesmo nessa época, o problema já estava surgindo, mesmo com tecnologias rudimentares, o que só tendia a crescer.

**Definição de Engenharia de Software**

Área da Computação que propõe soluções para desenvolver sistemas de software, - principalmente mais complexos - de forma produtiva e com qualidade.

**O que se estuda em ES?**

1. Engenharia de requisitos
2. Projeto de software
3. Construção de software
4. Testes de software
5. Manutenção de software
6. Gerência de configuração
7. Gerência de projetos
8. Processos de software
9. Modelos de software
10. Qualidade de software

**Não existe bala de prata “tudo depende do contexto”**

**Motivo: Dificuldades Essenciais**

Tornam Engenharia de Software diferente de outras engenharias

Complexidade

Conformidade

Facilidade de Mudanças

Invisibilidade

**Requisitos de Software**

Requisitos: o que sistema deve fazer para atender aos seus clientes com qualidade de serviço.

**Requisitos funcionais e não funcionais**

* Funcional: “o que” um sistema deve fazer – Funcionalidades ou serviços que ele deve implementar.
* Não funcional: “como” um sistema deve operar – sob quais restrições e com qual qualidade de serviço.

**Exemplos de requisitos não funcionais**

* Desempenho: dar o saldo da conta em 5 segundos
* Disponibilidade: estar no ar 99% do tempo
* Capacidade: armazenar dados de 1m de clientes
* Tolerância a falhas: continuar operando se São Paulo cair
* Segurança: criptografar dados trocados com as agências
* Privacidade: não armazenar localização dos usuários
* Interoperabilidade: se integrar com os sitemas do BACEN
* Manutenibilidade: bugs devem corrigidos em 24hr
* Usabilidade: compatibilidade com celulares e tablets

**Testes de Software**

* Verificam se um programa apresenta resultado esperado ao ser executado com casos de teste.
* Podem ser – Manuais ou Automatizados.

**Manutenção de Software**

Tudo que é feito após o lançamento do software é considerado manutenção de software

* Corretiva
* Preventiva
* Adaptativa
* Evolutiva
* Refactoring

**Sistemas legados**

* Sistemas antigos: usando linguagens, SOs, BDs antigos
* Manutenção custosa e arriscada
* Muitas vezes, são muito importantes (legado != irrelevantes)

**COBOL ainda é muito comum em bancos**

* Estima-se que existam em torno de 200 bilhões de LOC em COBOL
* Maioria são sistemas de bancos **–** 95% das transações em ATMs são em COBOL
* Um único banco europeu tem 250 MLOC em COBOL

**Processos de Desenvolvimento de Software**

* Processo de software: define as atividades que devem ser seguidas para construir um sistema de software**.**
* Dois principais modelos: Waterfall (“cascata”) e Ágil.

**Modelo em Cascata**

1. Levantamento de requisitos
2. Análise
3. Projeto
4. Codificação
5. Testes
6. Implantação

**Problemas com Modelo Waterfall**

* Requisitos mudam com frequência
  + Levantamento completos de requisitos demanda tempo
  + Quando ficar pronto, o “mundo já mudou”
  + Clientes ás vezes não sabem o que querem
* Documentações de software são verbosas
* E rapidamente se tornam obsoletas

**Manifesto Ágil (2001)**

* Encontro de 17 engenheiros de software em Utah
* Crítica a modelos sequenciais e pesados
* Novo modelo: incremental e iterativo

**Aspectos Éticos**

* Engenheiros de Software começam a questionar o uso que as empresas fazem do software desenvolvido por eles.

**Tipos ABC de sistemas**

* Existem 3 tipos de sistemas
* Classificação proposta por Bertrand Meyer
* Sistemas C (casuais)
  + Tipo muito comum de sistema
  + Sistemas pequenos, sem muita importância
  + Podem ter bugs
* Sistemas B (Business)
  + Sistemas importantes para uma organização
  + Risco: não usarem técnicas de ES e se tornarem um passivo
* Sistemas A (Acute)
  + Sistema onde nada pode dar errado, pois o custo é imenso, em termos de vidas humanas
  + Também chamados sistemas de missão crítica
  + Requerem certificações

**Exercícios**

1. Os custos com manutenção podem alcançar 80% dos custos totais alocados a um projeto de software durante o seu ciclo de vida. Por que esse valor é tão alto?

R: É claro que em custos muito altos, podemos associar esse valor gasto com más decisões de negócios, despreparação da equipe ou outros fatores do tipo.

Mas, quando falamos em manutenção de um software, logo pensamos em correções de bugs ou apenas alguma implementação nova. Porém, precisamos ter a ciência que manutenção de software envolve fatores corretivos, preventivos, adaptativos, evolutivos e os refactorings (revisão e reescrita de trechos do código SEM ALTERAR SUA FINALIDADE) e todos esses processos são o que mantém um software durante sua vida útil, onde seria lógico todos esses trabalhos alcançarem cerca de até 80% do seu custo.

Ou seja, a manutenção de um software é o que corrige, adapta, implementa e deixa tudo da maneira mais completa e desejável para o cliente.

2. Refactoring é uma transformação de código que preserva comportamento. Qual o significado da expressão preservar comportamento?

R: Seria basicamente a revisão e reescrita de trechos do código sem que alterem o seu comportamento final.

3. Em testes, existe uma frase muito famosa, de autoria de Edsger W. Dijkstra, que diz que:

**“testes de software mostram a presença de bugs, mas não a sua ausência.”**

Por que testes são incapazes de mostrar a ausência de bugs?

R: Os testes de software não conseguem mostrar a ausência de bugs, devido a vários fatores, como: complexidade do software desenvolvido, limitações nos casos de teste, condições de execução e até mesmo a complexidade desses bugs. Mesmo com testes abrangentes, sempre há espaço de que certos cenários não sejam cobertos, deixando espaço para bugs passarem despercebidos. Portanto, por mais que testes tem a finalidade de mostrar falhas, ele não determina a ausência completa de bugs.

4. Em gerência de projetos de software, existe uma lei empírica muito famosa, chamada Lei de Brooks, que diz que

**“incluir novos devs em um projeto que está atrasado, vai deixá-lo mais atrasado ainda.”**

Por que essa lei tende a ser verdadeira?

R: Adicionar novos desenvolvedores a um projeto já atrasado pode resultar em mais atrasos devido ao tempo necessário para se alinharem com o projeto e a equipe existente. Isso inclui processos de integração, treinamento e orientação, o que pode interromper o fluxo de trabalho de desenvolvedores atuais. Além disso, gerenciar uma equipe maior, pode ser mais complexo e demandar mais tempo, contribuindo para o prolongamento do atraso do projeto.

5. Em 2015, descobriu-se que o software instalado em milhões de carros da Volkswagen comparta-se de forma diferente quando testado em um laboratório de certificação. Nessas situações, o carro emitia poluentes dentro das normas. Fora do laboratório, ele emitia mais poluentes... Ou seja, o código incluía um “if” como o seguinte (meramente ilustrativo). O que você faria se seu chefe pedisse para escrever um if como esse?

If “Carro sendo testado em um laboratório”

“Emita poluentes dentro das normas”

Else

“Emita poluentes fora das normas”

R: Esse contexto entra muito no aspecto de ética em engenharia de software. Porque leva em conta o princípio de que o algoritmo foi desenvolvido de forma enganosa e despretensiosa. Com isso, acredito que eu tentaria conversar com o meu chefe de uma maneira sútil e objetiva, mostrando os sérios riscos ao fazer tal “if” no código.