Software: Natureza e Qualidades

Princípios da Eng. de Software

Prof. Márcio Lopes Cornélio

Software

- Produto da Engenharia de Software
- Diferentemente de produtos de outras Engenharias, software é *maleável*, pode ser modificado
  - Mudança deveria ser vista como mudança no *design* e não no código

Classificação sobre qualidade de software

- Qualidade externa
  - Visível ao usuário do sistema
- Qualidade interna
  - Diz respeito aos desenvolvedores do sistema
- Fatores internos ajudam a obter fatores externos de qualidade
  - Verificação x Confiabilidade

Qualidades do produto e do processo

- Qualidade do processo está relacionada com qualidade do produto
  - Ex: processo que exige dados de teste antes do projeto e codificação melhora confiabilidade do produto
  - Produto inclui todos os artefatos produzidos ao longo do processo

Correção

- Um programa *funcionalmente correto* se comporta de acordo com sua especificação
- Especificação disponível, não-ambígua, sendo possível determinar se o programa está de acordo com a especificação
- Propriedade matemática que estabelece equivalência entre um programa e sua especificação
- Pode ser aferida por: testes, verificação formal de correção

Confiabilidade (fidedigno)

- Comportamento estatístico (probabilidade de um software operar como esperado em um intervalo de tempo
- Correção é absoluta (desvio dos requisitos torna o sistema incorreto),
- Confiabilidade é relativo (se a consequência de um erro não é séria o software pode ainda ser confiável
- Ao contrário de produtos de outras engenharias, erros de projeto de software têm sido tratados como inevitáveis

Correção x Confiabilidade

Robustez

- Capacidade de comportamento “razoável”, mesmo em circunstâncias não especificadas
  - Ex: Entrada de dados incorreta
- Dificuldade de definir o que é “razoável”
- Presença de requisito na especificação, realização é questão de *correção*; ausência é questão de robustez

Desempenho

- Em Eng. de Sw é comum “desempenho” e eficiência são tratados da mesma forma
- Software eficiente usa recursos computacionais economicamente
- Afeta escalabilidade (depende do tamanho da entrada de um algoritmo
- Formas de medir
  - Complexidade de algoritmos (caso médio e pior caso)
  - Medição, análise e simulação

Verificabilidade

- Um software é verificável se suas propriedades podem ser facilmente verificadas
  - Exemplos: correção ou desempenho
  - Pode-se utilizar monitores de software

Manutenção

- Em geral, pensa-se em correção de defeitos
- Termo não muito adequado, pois envolve várias atividades que modificam partes de um software para melhoria
- Em essência é “evolução de software”
- Três categorias
  - Corretiva: remoção de erros
  - Adaptativa: adaptação a mudanças no ambiente
  - Perfectiva: mudança para melhoria de qualidade

Portabilidade

- Diz respeito à capacidade de executar um software em ambientes diferentes
  - Ambiente: hardware ou ambiente de software
- Exemplo de software para máquinas específicas
  - Sistema Operacional
  - Ainda é possível obter algum grau de portabilidade

Compreensível

- Fator interno de qualidade
- Ajuda a alcançar outros fatores como capacidades de evolução e verificação

Interoperabilidade

- Capacidade de coexistir e cooperar com outros sistemas
  - Exemplo: editor de texto que utiliza um gráfico feito por outra ferramenta
- Pode ser obtida por padronização de interfaces
- Conceito relacionado: sistema aberto
  - Coleção extensível de aplicação escritas independentemente e que cooperam para funcionar como um sistema integrado
  - Permitem a adição de nova funcionalidade por organizações independentes

Requisitos de qualidade em áreas de aplicação distintas

- Sistemas de Informação
  - Integridade de dados
    - Sob que circunstâncias os dados serão corrompidos devido ao mal funcionamento do sistema?
  - Segurança
    - Até que ponto o sistema protege os dados de acesso não-autorizado?
  - Disponibilidade de dados
    - Sob que circunstâncias os dados se tornarão indisponíveis e por quanto tempo?
  - Desempenho de transações
    - Quantidade de transações por unidade de tempo

Requisitos de qualidade em áreas de aplicação distintas

- Sistemas de tempo real
  - Respondem a eventos dentro de um período de tempo predeterminado
  - Noção de tempo de resposta é algo que pode ser especificado e verificado
  - Tempo de resposta é um critério de correção e não de desempenho
    - Exemplo: operação crítica como monitoramento de pacientes

Princípios de Engenharia de Software

- Lidam tanto com o *processo* de engenharia quanto com o *produto* final
  - Processo correto ajuda a obter um produto correto
- Princípios gerais
  - Aplicáveis ao processo de construção e gerenciamento

Rigor e formalidade

- Abordagem rigorosa permite obter produtos mais confiáveis, com controle de custos
- Formalidade
  - Mais alto nível de rigor
  - Processo de software dirigido e avaliado por leis matemáticas
  - Métodos e técnicas aplicados podem ser combinação de resultados teóricos, ajustes empíricos ou experiência anterior
  - Não há necessidade de ser sempre formal, mas deve-se perceber quando for necessário
- Sistemático

Separation of concerns

- Permite lidar com diferentes aspectos individuais de um problema, de forma que nos concentremos em cada um separadamente
  - Exemplo
    - Decisão (permuta de dados da memória para disco)
    - Depende do tamanho da memória
    - Pode afetar a política de recuperação de erro
- Para lidar com complexidade de um projeto
  - Separar as diferentes preocupações

Separation of concerns

- Como obter
  - Separação no tempo
    - Exemplos: horário pessoal de atividades, atividades de um processo de software
  - Em termos de qualidade
    - Exemplo: lidar separadamente com eficiência e correção de um programa
  - Por *visões* diferente*s do software*
    - Exemplo: a partir da análise de requisitos pode-se concentrar em fluxo de dados ou fluxo de controle
  - Partes do sistema
    - Separação em termos do tamanho

Modularidade

- Consiste em dividir um software complexo em partes menores (módulos)
- Duas estratégias
  - *Bottom-up*
  - *Top-down*
- Decomposição, composição e entendimento
  - Exigem alta coesão e baixo acoplamento
- Decomposição: dividir para conquistar

Modularidade

- Capacidade de entendimento x modificação
- Para obter composição, decomposição e entendimento modulares precisamos que módulos sejam
  - Altamente coesos
    - Todos os elementos são fortemente conectados
  - Fracamente acoplados
    - Acoplamento mede interdependência entre módulos

Abstração

- Processo que identifica o que é importante e ignora os detalhes
- Modelos são abstrações da realidade
- Linguagens de programação são abstração sobre hardware
  - Comando de atribuição

Antecipação à mudança

- Software muda constantemente
- Antecipação diferencia software de outros tipos de produção industrial
- Novos requisitos surgem ou antigos são atualizados

Generalidade

- Soluções gerais
  - Exemplo: merge de dois arquivos com registros ordenados
    - Programa que faça *merge* mesmo que registros diferentes que tenham chaves iguais