O Programador Pragmático:

De Aprendiz a Mestre

•••

Capítulo 7: Antes do Projeto

Este material foi desenvolvido pelo grupo PET/ADS do IFSP São Carlos

Introdução

"Já teve a sensação de que seu projeto está fadado ao fracasso, mesmo antes de ele começar? Pode ser que realmente esteja, a menos que você ***estabeleça algumas regras básicas antes."***

O Abismo dos Requisitos

A perfeição não é atingida quando não há nada mais a adicionar e sim quando não há nada mais a retirar...”

Antoine de St. Exupery, Vento, areia e estrelas, 1939

Coletar e reunir requisitos não é algo tão simples, uma vez que eles raramente estão na superfície.

Normalmente, os requisitos estão profundamente enterrados sob camadas de suposições, concepções erradas e política de negócio.

Dicas

DICA 51: Não colete requisitos, cave-os.

Cavando Requisitos

Na busca por um requisito real, há duas respostas:

- A simples: um requisito é a declaração direta de algo que precisa ser feito; bons requisitos são claros e específicos.

- A complexa: reconhecer requisitos consiste na compreensão profunda das nuances e contextos envolvidos, como a adaptação às mudanças nas políticas de negócios, a compreensão das ações dos usuários e considerações sobre a implementação.

Separação entre requisito real e política de negócio

Exemplo de um bom requisito:

"*Um registro de funcionário só pode ser visualizado por um grupo de pessoas designadas.* "

Exemplo de um mau requisito:

"Só os supervisores de um funcionário e o departamento de pessoal podem ver os registros desse funcionário. "

O segundo é um mau requisito porque mistura requisitos com política de negócio. É uma sentença que pode mudar futuramente.

Primeira Regra Básica

- Torne o requisito a **declaração geral** e forneça para os desenvolvedores as informações da política (elementos que podem variar).

Essa regra, além de exigir lidar com a profundidade dos requisitos, também consiste em compreender o contexto dos mesmos.

No cenário anterior, provavelmente, o desenvolvedor criará um sistema de controle de acesso. Quando a política mudar (e ela mudará), só os metadados precisarão ser atualizados.

Segunda Regra Básica

- Considere as **necessidades reais** de interface do usuário

Há uma necessidade de documentar as razões por trás dos requisitos.

É fundamental destacar a importância de descobrir a razão real para os usuários solicitarem algo, em vez de apenas conhecer o modo como fazem as coisas atualmente.

Segunda Regra Básica

Exemplo de um requisito real e claro:
“O sistema deve permitir que você selecione um prazo para o empréstimo. “

Exemplo de um requisito dúbio (pode ou não ser um requisito):
“Precisamos de uma caixa de listagem para a seleção do prazo do empréstimo. “

Por que o segundo não é um bom requisito:

Enquanto o primeiro indica a implementação de uma certa funcionalidade se baseando no que o usuário precisa, o segundo exemplo está descrevendo como os usuários usam certa funcionalidade atualmente.

Segunda Regra Básica

É importante **entender por que os usuários precisam de uma determinada funcionalidade** , em vez de apenas saber como eles a utilizam atualmente.

O desenvolvimento tem de **resolver o problema do negócio** e não apenas atender os requisitos declarados.

Compreenda as funcionalidades que os usuários precisam, sem se prender à forma como as utilizam atualmente.

Dicas

DICA 52: Trabalhe com um usuário para pensar como um usuário

Torne-se um Usuário

Para entender o que os usuários precisam, a dica é: torne-se um usuário.

É assim que se conhece as expectativas e esperanças do usuário quanto ao sistema que você está construindo.

Passe um tempo no ambiente do usuário para compreender verdadeiramente como o sistema será usado.

Documentando Requisitos

A maneira mais comumente aceita para documentar requisitos e capturá-los é através do conceito de casos de uso.

No entanto, o autor que propôs o conceito de casos de uso, Ivar Jacobson, não especificou exatamente como ele deve ser.

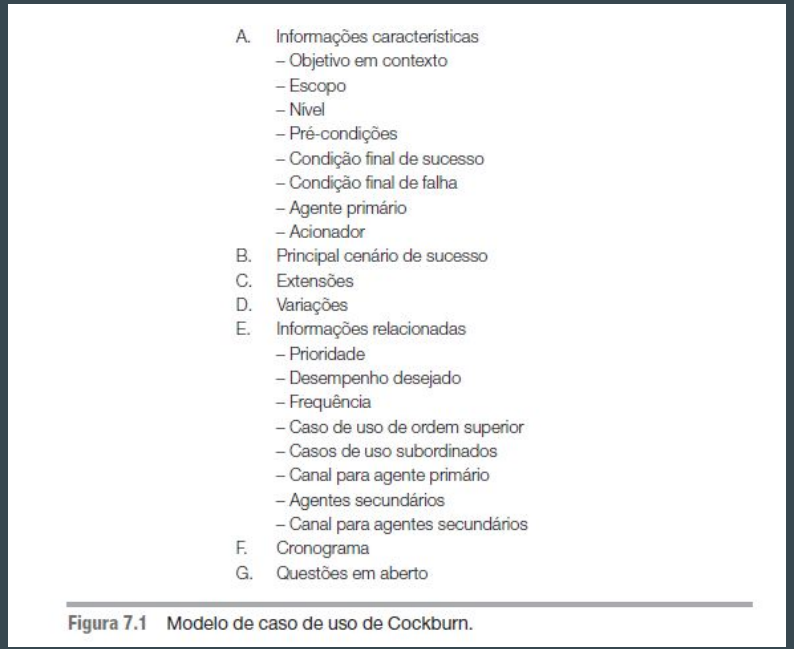
O caso de uso deve ser formal ou informal, prosa simples ou um documento estruturado (como um formulário)? Que nível de detalhe é apropriado (lembre-se de que temos um público amplo)?

Documentando Requisitos

Uma maneira de examinar casos de uso é enfatizar sua natureza direcionada a objetivos.

Esse método leva a um modelo formal que dá suporte à estrutura hierárquica dos casos de uso.

A imagem ao lado apresenta um exemplo:



A. Informações características
   – Objetivo em contexto
   – Escopo
   – Nível
   – Pré-condições
   – Condição final de sucesso
   – Condição final de falha
   – Agente primário
   – Acionador
B. Principal cenário de sucesso
C. Extensões
D. Variações
E. Informações relacionadas
   – Prioridade
   – Desempenho desejado
   – Frequência
   – Caso de uso de ordem superior
   – Casos de uso subordinados
   – Canal para agente primário
   – Agentes secundários
   – Canal para agentes secundários
F. Cronograma
G. Questões em aberto

Figura 7.1 Modelo de caso de uso de Cockburn.

Documentando Requisitos

O exemplo a seguir é mais detalhado, com descrições textuais e modelagem de negócio:

CASO DE USO 5: COMPRA DE MERCADORIAS

A. **Informações características**
- **Objetivo em contexto:** Comprador emite solicitação diretamente para nossa empresa, espera a entrega das mercadorias e a cobrança.
- **Escopo:** Empresa
- **Nível:** Resumo
- **Pré-condições:** Conhecermos o comprador, seu endereço etc.
- **Condição final de sucesso:** O comprador receber as mercadorias, recebermos o dinheiro pelas mercadorias.
- **Condição final de falha:** Não termos enviado as mercadorias, o comprador não ter enviado o dinheiro.
- **Agente primário:** Comprador, qualquer agente (ou computador) atuando pelo cliente.
- **Acionador:** Recebimento do pedido de compra.

B. **Principal cenário de sucesso**
1. Comprador entra em contato com uma solicitação de compra.
2. Empresa verifica o nome do comprador, endereço, mercadorias solicitadas etc.
3. Empresa fornece ao comprador informações sobre mercadorias, preços, datas de entrega etc.
4. Comprador confirma o pedido.
5. Empresa gera o pedido, envia o pedido para o comprador.
6. Empresa envia fatura para o comprador.
7. Comprador paga fatura.

C. **Extensões**

3a. Empresa não tem um dos itens pedidos: renegociação de pedido.

4a. Comprador paga diretamente com cartão de crédito: recebimento de pagamento por cartão de crédito (caso de uso 44).

7a. Comprador devolve mercadorias: manipulação de mercadorias devolvidas (caso de uso 105).

D. **Variações**
1. Comprador pode usar telefone, fax, formulário de pedido na Web, intercâmbio eletrônico.
7. Comprador pode pagar em dinheiro, ordem de pagamento, cheque ou cartão de crédito.

E. **Informações relacionadas**
- **Prioridade:** Máxima
- **Desempenho desejado:** 5 minutos por pedido, 45 dias até pagamento
- **Frequência:** 200/dia
- **Caso de uso de ordem superior**: Gerenciamento do relacionamento com o cliente (caso de uso 2).
- **Casos de uso subordinados:** Geração de pedido (15). Recebimento de pagamento por cartão de crédito (44). Manipulação de mercadorias devolvidas (105).
- **Canal para ator primário:** Pode ser telefone, arquivo ou interativo
- **Atores secundários:** Empresa de cartão de crédito, banco, serviço de entrega

F. **Cronograma**
- **Data de vencimento:** Versão 1.0

G. **Questões em aberto**
- O que acontece se tivermos parte do pedido?
- O que acontece se o cartão de crédito for roubado?

Figura 7.2 Um exemplo de caso de uso.

Documentando Requisitos

Há também diagramas de casos de uso:

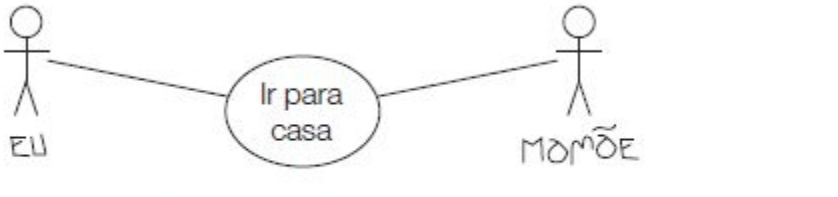


Figura 7.3 Casos de uso UML – tão simples que uma criança poderia criá-lo!

Esse é um exemplo um tanto sarcástico, mostrando um diagrama muito simples que até mesmo uma criança poderia criar.

Documentando Requisitos

Use qualquer método que melhor comunique ao seu público, seguindo as seguintes dicas:

- **Adaptação ao público-alvo** : adapte a forma como os requisitos são documentados para atender ao entendimento e às expectativas de cada público.
- **Métodos variados** : utilize uma variedade de métodos de documentação, como casos de uso, protótipos, diagramas ou até mesmo narrativas descritivas, dependendo da situação.
- **Feedback Contínuo** : estabeleça um processo de feedback contínuo com as partes interessadas. Isso ajuda a refinar requisitos à medida que o projeto avança e fortalece a colaboração e a compreensão mútua.

Documentando Requisitos: O que evitar

- Especificar em excesso

Evite entrar em detalhes excessivos que possam prejudicar a flexibilidade do projeto. A especificação excessiva pode resultar em documentação complexa e difícil de manter.

Enfatize a simplicidade. Um documento claro e conciso é mais propenso a ser compreendido e seguido.

Bons documentos de requisitos permanecem abstratos. No entanto, isso não significa que possam ser vagos. Eles devem focar em aspectos essenciais que devem ser comunicados.

Dicas

DICA 53: Abstrações têm vida mais longa do que detalhes.

Documentando Requisitos: O que evitar

- Prever o futuro

Exemplo:

“O sistema faz uso ativo de uma abstração de DATAs. Ele implementará os serviços de DATAs, como na formatação, armazenamento e operações matemáticas, consistente e universalmente. “

Não tente prever detalhes específicos do futuro. É desnecessário especular sobre implementações futuras, como no exemplo da abstração de DATAs.

Os requisitos devem se concentrar nas necessidades e funcionalidades desejadas, sem amarrar o sistema a tecnologias específicas.

Documentando Requisitos: O que evitar

- Falta de rastreamento ativo dos requisitos

Rastrear ativamente os requisitos é essencial para uma gestão eficaz do projeto.

Muitos projetos não têm informações sobre alterações no escopo, quem solicitou um recurso, quem o aprovou, entre outros detalhes.

Garantir um sistema eficaz de rastreamento de requisitos é fundamental para o gerenciamento adequado do crescimento do escopo.

Rastrear requisitos evitam problemas relacionados ao aumento não controlado de funcionalidades ao longo do projeto.

Documentando Requisitos: Conclusão

“Os requisitos não são a arquitetura. Os requisitos não são o projeto, nem a interface de usuário. Os requisitos são necessidades. “

Mantenha um Glossário

Crie e mantenha um glossário do projeto: um local que defina todos os termos e o vocabulário específicos usados em um projeto.

Exemplo: usuários podem fazer a distinção entre um “cliente” e um “freguês”. Seria inapropriado usar as duas palavras casualmente no sistema.

Manter um glossário é bom para assegurar a consistência.

É muito difícil ser bem-sucedido em um projeto em que os usuários e desenvolvedores se referem à mesma coisa usando nomes diferentes ou, ainda pior, referem-se a coisas diferentes usando o mesmo nome.

Dicas

DICA 54: Use um glossário do projeto

Divulgue o que ficou definido

Para divulgar a documentação dos requisitos, a publicação de documentos de projeto em sites internos da Web para fácil acesso pode ser útil.

Apresentando os requisitos em um documento de hipertexto, podemos atender melhor às necessidades de um público variado.

Os patrocinadores do projeto podem navegar usando um alto nível de abstração para verificar se os objetivos do negócio estão sendo atendidos.

Os programadores podem usar hiperlinks para descer a níveis maiores de detalhe.

Enfim, o projeto: resolvendo problemas impossíveis

“De vez em quando, nos vemos envolvidos no meio de um projeto quando surge um enigma realmente difícil: alguma parte de engenharia com a qual não sabemos lidar ou talvez algum trecho de código que está se mostrando muito mais difícil de escrever do que pensávamos. Pode parecer impossível. Mas é mesmo tão difícil quanto parece? ”

Identificando as restrições reais

O segredo para resolver o enigma é identificar as **restrições reais** .

Restrições reais são aquelas não imaginadas, que existem de fato. Restrições podem ser técnicas, podem envolver tempo e prazos, podem abranger recursos, entre outros fatores.

Algumas restrições são absolutas e devem ser respeitadas. Outras restrições podem não ser reais, sendo apenas noções preconcebidas que podem ser superadas.

> *“A abordagem pragmática envolve entender e lidar efetivamente com essas restrições para encontrar soluções viáveis para problemas aparentemente impossíveis. “*

Graus de liberdade

Muitos diriam que para sair desse problema impossível o necessário é que se pense fora da caixa.

No entanto, se a “caixa” for o limite das restrições e condições, o truque será encontrar a caixa, que pode ser consideravelmente maior do que você pensa.

- A chave para resolver enigmas é reconhecer as restrições impostas a você e os graus de liberdade que você realmente tem, porque neles estará sua solução.

Dicas

DICA 55: Não pense fora da caixa – encontre a caixa.

Enumere todas as saídas possíveis

Diante de problemas desafiadores, explore todas as possíveis soluções, sem descartar nenhuma inicialmente.

Analise cada opção considerando por que ela pode não ser viável, e questione a certeza dessa conclusão.

Categorize e priorize as restrições, identificando as mais limitantes primeiro para orientar o desenvolvimento e encaixar as restrições restantes de forma apropriada.

Tem de haver um caminho mais fácil!

"Há situações em que nos vemos trabalhando em um problema que parece muito mais difícil do que achamos que seria. Você pode ter a impressão de estar tomando o caminho errado – de que deve haver um caminho mais fácil do que esse! (...) É aí que deve fazer uma pausa e se perguntar: "

• Há um caminho mais fácil?

• Você está tentando resolver o problema certo ou sua atenção foi desviada por um assunto técnico periférico?

• Por que esse fato é um problema?

• O que está tornando isso tão difícil de resolver?

• Precisa ser feito dessa maneira?

• Afinal, precisa mesmo ser feito?

Questione o problema

Nos questionando sobre o problema, tentando responder essas questões, podemos chegar a conclusões surpreendentes.

Em muitos casos, uma reinterpretação dos requisitos pode fazer um grupo inteiro de problemas desaparecer.

“Você só precisa das restrições reais, das restrições enganosas e da sabedoria para saber a diferença. “

Não antes de você estar pronto

“Aquele que hesita às vezes é salvo.”

James Thurber, The glass in the field

Profissionais de sucesso compartilham uma peculiaridade: sabem quando começar e quando esperar. Tem de escutar a voz que sussurra “espere”.

Se sentar para começar a digitar e houver alguma dúvida insistente em sua mente, dê atenção a ela.

Dicas

DICA 56: Só comece quando estiver pronto.

Não antes de você estar pronto

- Confie nos seus instintos

Há um certo instinto de cada programador, que é adquirido na prática, acumulando experiência testando o que funciona ou não.

O desenvolvimento de software não é uma ciência exata, e confiar nos instintos pode aprimorar o desempenho.

Ao notar dúvidas persistentes ou relutância em uma tarefa, sugere fazer uma pausa para que as dúvidas se tornem mais claras.

Bom senso ou procrastinação?

- Diferencie o bom senso da procrastinação.

Certamente, começar um novo projeto (ou um novo módulo em um projeto existente) pode ser uma experiência difícil e estressante. A dica é: comece um protótipo.

Se sentir tédio logo no início, pode indicar que estava procrastinando. Nesse caso, abandone o protótipo e comece o desenvolvimento real.

Se o protótipo progredir, seus instintos estavam corretos, identificando que as premissas básicas estavam erradas. Abandonar o protótipo nesse momento economiza tempo e esforço.

A armadilha das especificações

Especificar um programa é o processo de pegar um requisito e resumi-lo até o ponto em que a habilidade do programador possa assumir o controle.

Além de dar informações para o desenvolvedor que executará a implementação inicial, a especificação é um **registro** para futuras gerações de programadores que editarão e aperfeiçoarão o código.

A especificação também é um **acordo** com o usuário – uma codificação de suas necessidades e um contrato implícito que estabelece que o sistema final respeitará os requisitos.

A armadilha das especificações

Escrever uma especificação é uma responsabilidade e tanto ...

Porém, o problema se encontra no fato que muitos projetistas acham difícil parar. Não param até cada pequeno detalhe minucioso seja resumido.

Há dois problemas principais nisso:

- Problema comum: é ingênuo pensar que uma especificação capturará todos os detalhes de um sistema.
- Limitação linguística: o idioma natural é limitado na expressão de operações.

Dicas

DICA 57: Algumas coisas são fáceis de fazer, mas não de descrever.

Efeito camisa de força

No looping indefinido de especificações, um projeto que não deixa espaço de interpretação para o codificador elimina do esforço de programação qualquer habilidade e arte.

Especificações detalhadas podem ser necessárias, mas é crucial não permitir que se tornem obstáculos para a codificação.

Evite cair no ciclo vicioso da especificação e, eventualmente, comece a codificar.

Círculos e setas

Existem inúmeros métodos de engenharia de software. Modelo em espiral, modelo cascata, diagramas ER, Método Jackson, etc. Cada método atraiu seus discípulos e teve sua popularidade.

O problema está em alguns desenvolvedores que agarram à última moda, que ficam reféns de tais modelos.

“(...) adotar cegamente qualquer técnica sem trazê-la para o contexto de suas práticas e capacidade de desenvolvimento é uma receita para a frustração. “

Dicas

DICA 58: Não seja escravo dos processos.

Defeitos graves dos processos

1- Captura de Requisitos:

- Uso de diagramas e linguagem técnica, muitas vezes irrelevantes para usuários finais.
- Falta de verificação formal pelos usuários, dependendo das interpretações dos projetistas.
- Proposta de preferência por protótipos demonstráveis.

Na captura de requisitos, é importante evitar linguagem técnica excessiva e diagramas muito complexos. Opte por protótipos mais compreensíveis pelos usuários finais.

Defeitos graves dos processos

2- Especialização e Comunicação:

- Incentivo à especialização em modelos específicos leva a comunicação deficiente.
- Tendência à mentalidade "nós versus eles" entre projetistas e codificadores.
- Abordagem favorável ao entendimento total do sistema.

É essencial buscar uma compreensão global do sistema para evitar conflitos entre os programadores e os projetistas (aqueles que especificaram).

Promova uma comunicação eficaz entre ambos.

Defeitos graves dos processos

3- Adaptabilidade e Dinamismo:

- Dificuldade em criar sistemas adaptáveis e dinâmicos.
- Métodos atuais favorecem modelos estáticos, incapazes de ilustrar dinamismo desejado.
- Risco de definir relacionamentos estáticos em vez de conexões dinâmicas entre objetos.

Para garantir adaptabilidade e dinamismo nos sistemas, é necessário superar a tendência aos modelos estáticos. Esses processos têm dificuldade de criar sistemas adaptáveis e dinâmicos.

Devemos usar processos?

Certamente, mas lembre-se sempre de que métodos de desenvolvimento formais são apenas mais uma ferramenta da caixa de ferramentas.

“Programadores pragmáticos consideram as metodologias criticamente e então extraem o melhor de cada uma e as combinam em um conjunto de práticas de trabalho que fica melhor a cada mês. “

Cuidado ao encontrar abordagens e filosofias extremas.

Considere o custo de ferramentas: não se deixe influenciar pelo custo de ferramentas; a qualidade do projeto não está diretamente ligada ao seu preço.

Dicas

DICA 59: Ferramentas caras não produzem projetos melhores.

Considerações finais

Especificação Consciente: evite detalhamentos excessivos na especificação, reconhecendo suas limitações e permitindo espaço para interpretação.

Pragmatismo na Escolha: avalie criticamente métodos formais, extrair o melhor de cada um e combiná-los em práticas de trabalho evolutivas. Cuidado com abordagens extremas.

Valor sobre Custo: o preço de ferramentas não garante a qualidade do projeto, avalie o valor proporcionado independentemente do custo.

Sobre os autores

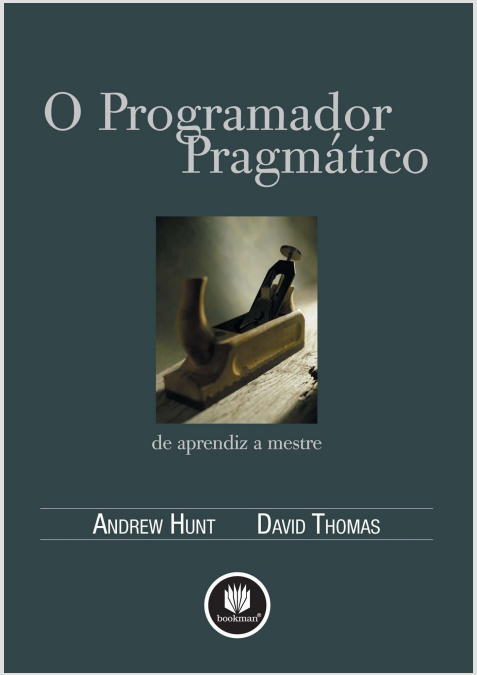
Andrew Hunt trabalhou em diversas áreas, como telecomunicações, serviços financeiros, artes gráficas etc. Hunt se especializou em combinar técnicas já consolidadas com tecnologias de ponta, criando soluções novas e práticas. Ele administra sua empresa de consultoria em Raleigh, Carolina do Norte.

Em 1994 , David Thomas, fundou na Inglaterra uma empresa de criação de software certificada pela ISO9001 que distribuiu mundialmente projetos sofisticados e personalizados. Hoje, Thomas é consultor independente e vive em Dallas, Texas.

Atualmente, David e Andrew trabalham juntos em The Pragmatic Programmers, L.L.C

Este conjunto de slides foi elaborado a partir da obra:

HUNT, Andrew ; THOMAS, David. **O Programador Pragmático:** de aprendiz a mestre.Bookman, 2010.



A equipe

Bianca Antonietti, Autora

Aluna e bolsista do PET/ADS desde dezembro de 2022.
LinkedIn

Lucas Oliveira, Revisor

Professor de Computação, é tutor do PET/ADS desde janeiro de 2023.
LinkedIn