Formas de Escrever Requisitos de Software

Um guia completo sobre técnicas de especificação de requisitos para o sucesso do seu projeto

A Importância dos Requisitos de Software

Requisitos de software são a base de qualquer projeto bem-sucedido, definindo claramente o que o sistema deve fazer e sob quais condições. Uma especificação adequada de requisitos traz inúmeros benefícios:

Alinhamento de Expectativas
Garante que todos os envolvidos tenham a mesma visão do produto final

Planejamento Adequado
Permite estimativas mais precisas de tempo e recursos necessários

Redução de Retrabalho
Minimiza mudanças e correções durante o desenvolvimento

Qualidade do Produto
Aumenta a chance de entregar um produto que atenda às necessidades reais dos usuários

Redução de Custos
Corrigir problemas na fase de requisitos custa significativamente menos que em fases posteriores

Prevenção de Falhas
Requisitos bem definidos previnem falhas críticas no sistema

Você sabia? Estudos mostram que cerca de 50% dos problemas em projetos de software são originários da fase de especificação de requisitos.

Principais Metodologias de Especificação

Histórias de Usuário

Descrições curtas e simples de uma funcionalidade, escritas do ponto de vista do usuário. Amplamente usadas em metodologias ágeis como Scrum.

Estrutura básica:

```
Como [papel do usuário]
Eu quero [ação/objetivo]
Para que [benefício/valor]
```

Exemplo:

```
Como usuário do aplicativo bancário
Eu quero poder visualizar meu saldo atual
Para que eu possa controlar meus gastos
```

Características principais:

- Foco no valor para o usuário
- Linguagem simples e não técnica
- Baseada nos 3 C's: Cartão, Conversação e Confirmação
- Ideal para desenvolvimento ágil e incremental
- Promove discussões entre a equipe e o cliente

Casos de Uso

Descrições detalhadas de como um usuário (ator) interage com o sistema para alcançar um objetivo específico. Documentam o comportamento do sistema em resposta às ações do usuário.

Estrutura básica:

```
Nome do Caso de Uso: [Verbo + Substantivo]
Ator Principal: [Quem executa]
Pré-condições: [O que deve ser verdadeiro antes]
Fluxo Principal: [Passos numerados]
Fluxos Alternativos: [Exceções]
Pós-condições: [O que deve ser verdadeiro depois]
```

Exemplo simplificado:

```
Nome do Caso de Uso: Transferir Valores
Ator Principal: Cliente do Banco
Fluxo Principal:
1. Autenticar Cliente
2. Cliente informa conta de destino
3. Cliente informa valor
4. Cliente confirma transferência
5. Sistema efetua transferência
Fluxos Alternativos:
3a. Se valor acima do saldo, solicitar novo valor
```

Características principais:

- Foco na interação entre usuário e sistema
- Descrição detalhada dos fluxos possíveis
- Pode incluir diagramas UML de casos de uso
- Utilizada em abordagens tradicionais como RUP
- Documenta comportamentos esperados do sistema

BDD com Gherkin

Behavior-Driven Development (BDD) utiliza a linguagem Gherkin para descrever o comportamento esperado do sistema em linguagem natural estruturada, que também serve como base para testes automatizados.

Estrutura básica Gherkin:

```
Funcionalidade: [Descrição da funcionalidade]
Cenário: [Descrição do cenário]
Dado [contexto inicial/pré-condição]
Quando [evento/ação]
Então [resultado/validação esperada]
```

Exemplo:

```
Funcionalidade: Login no sistema
Cenário: Login com credenciais válidas
Dado que o usuário está na página de login
E possui uma conta ativa no sistema
Quando ele preenche corretamente seu nome de usuário
E preenche corretamente sua senha
E clica no botão de login
Então ele deve ser redirecionado para a página inicial
E deve ver uma mensagem de boas-vindas
```

Características principais:

- Conecta requisitos, desenvolvimento e testes
- Linguagem estruturada e simples
- Facilita a automação de testes
- Foco no comportamento observável do sistema
- Promove colaboração entre técnicos e não técnicos

Especificação IEEE 830

Padrão formal da IEEE para Especificação de Requisitos de Software (SRS). Fornece um modelo estruturado para documentar requisitos de maneira abrangente e detalhada.

Estrutura básica:

```
1. Introdução
  1.1 Propósito
  1.2 Escopo
  1.3 Definições, acrônimos e abreviações
  1.4 Referências
  1.5 Visão geral
2. Descrição Geral
  2.1 Perspectiva do produto
  2.2 Funções do produto
  2.3 Características do usuário
  2.4 Restrições
  2.5 Premissas e dependências
3. Requisitos Específicos
  3.1 Requisitos funcionais
  3.2 Requisitos não funcionais
  3.3 Requisitos de interface
```

Exemplo de um requisito funcional:

```
RF001: O sistema deve permitir ao usuário realizar login utilizando nome de usuário e senha.
Descrição: O sistema deve autenticar usuários registrados através de suas credenciais.
Entrada: Nome de usuário e senha.
Processo: Validar credenciais contra banco de dados de usuários.
Saída: Acesso ao sistema ou mensagem de erro.
Prioridade: Alta
```

Características principais:

- Abordagem formal e estruturada
- Documentação completa e detalhada
- Separação clara entre requisitos funcionais e não funcionais
- Ideal para sistemas críticos ou regulamentados
- Fornece rastreabilidade de requisitos

FURPS+

Modelo para classificação de requisitos, onde o acrônimo representa categorias de atributos de qualidade de software. Desenvolvido pela Hewlett-Packard e posteriormente adotado pela Rational Software.

Categorias FURPS+:

```
F - Functionality (Funcionalidade): O que o software faz
U - Usability (Usabilidade): Facilidade de uso
R - Reliability (Confiabilidade): Disponibilidade, recuperação de falhas
P - Performance (Desempenho): Tempo de resposta, throughput
S - Supportability (Suportabilidade): Testabilidade, manutenibilidade
+ - Requisitos adicionais: Design, implementação, interface, físicos
```

Exemplo de aplicação:

```
F - Funcionalidade: O sistema deve permitir que usuários criem, editem e excluam contas
U - Usabilidade: A interface deve seguir padrões de acessibilidade WCAG 2.1 nível AA
R - Confiabilidade: O sistema deve ter uma disponibilidade de 99,9% durante horário comercial
P - Performance: O tempo de resposta para consultas não deve exceder 1 segundo
S - Suportabilidade: O sistema deve permitir atualizações sem tempo de inatividade
+ - Design: A arquitetura deve seguir o padrão MVC
```

Características principais:

- Abordagem abrangente para classificação de requisitos
- Equilíbrio requisitos funcionais e não funcionais
- Útil para checklist de verificação de completude
- Ajuda a garantir que todos os aspectos do sistema sejam considerados
- Facilita a organização de requisitos em categorias lógicas

Comparação entre Abordagens

Metodologia	Detalhamento	Complexidade	Metodologia Favorecida	Foco Principal
Histórias de Usuário	Baixo a Médio	Baixa	Ágil (Scrum, XP)	Valor para o usuário
Casos de Uso	Alto	Média a Alta	Tradicional (RUP, Waterfall)	Interações sistema-usuário
BDD com Gherkin	Médio	Média	Ágil com automação	Comportamento observável
IEEE 830	Muito Alto	Alta	Waterfall, V-Model	Completude e precisão
FURPS+	Médio a Alto	Média	RUP, adaptável	Atributos de qualidade

Pontos Fortes e Fracos

Histórias de Usuário

Pontos Fortes:
- Simplicidade e facilidade de entendimento
- Foco no valor para o negócio
- Flexibilidade para mudanças

Pontos Fracos:
- Falta de detalhamento técnico
- Pode deixar lacunas em requisitos complexos
- Pouca formalidade para sistemas críticos

Casos de Uso

Pontos Fortes:
- Detalhamento das interações
- Boa visualização de fluxos alternativos
- Eficaz para requisitos complexos

Pontos Fracos:
- Pode se tornar extenso e burocrático
- Resistente a mudanças frequentes
- Tempo considerável para elaboração

BDD com Gherkin

Pontos Fortes:
- Integração direta com testes
- Linguagem estruturada mas compreensível
- Facilita colaboração entre times

Pontos Fracos:
- Curva de aprendizado para estruturação adequada
- Pode se tornar repetitivo
- Foco em comportamento exige disciplina

IEEE 830

Pontos Fortes:
- Documentação exaustiva e completa
- Estrutura formal bem definida
- Ideal para sistemas críticos/regulamentados

Pontos Fracos:
- Processo lento e trabalhoso
- Resistente a mudanças
- Pode gerar documentação excessiva

FURPS+

Pontos Fortes:
- Abordagem holística para qualidade
- Cobertura ampla de aspectos do sistema
- Organização clara em categorias

Pontos Fracos:
- Pode ser complexo para equipes iniciantes
- Requer conhecimento amplo de atributos de qualidade
- Menos focado em necessidades específicas do usuário

Boas Práticas para Escrita de Requisitos

Clareza e Precisão

Use linguagem clara e objetiva
Evite termos ambíguos, jargões técnicos desnecessários e expressões vagas

Especifique o "o que", não o "como"
Foque no resultado desejado, não na implementação técnica

Seja específico e mensurável
Substitua "rápido" por "tempo de resposta máximo de 3 segundos"

Use voz ativa
"O sistema deve validar os dados" em vez de "Os dados devem ser validados"

Estrutura e Organização

Um requisito por vez
Evite combinar múltiplos requisitos em um único item

Use identificadores únicos
Numere requisitos para facilitar rastreabilidade (ex: REQ-001)

Agrupe logicamente
Organize requisitos por funcionalidade, módulo ou perfil de usuário

Priorize requisitos
Use classificações como "Essencial", "Importante", "Desejável"

O que Evitar

Requisitos Conflitantes
Verifique se há contradições entre requisitos diferentes

Exemplo ruim: "O sistema deve armazenar todos os dados históricos indefinidamente" vs "O sistema deve ocupar no máximo 10GB de espaço em disco"

Requisitos Não Verificáveis
Evite termos subjetivos que não podem ser testados objetivamente

Exemplo ruim: "O sistema deve ser amigável ao usuário" (como verificar?)

Melhor: "90% dos usuários devem completar a tarefa X em menos de 2 minutos sem ajuda"

Requisitos Ambíguos
Evite termos que podem ter múltiplas interpretações

Exemplo ruim: "O sistema deve responder rapidamente às consultas"

Melhor: "O sistema deve responder a consultas em menos de 1 segundo para 95% das requisições"

Requisitos Inatingíveis
Não especifique requisitos impossíveis ou irrealistas

Exemplo ruim: "O sistema deve ter 100% de disponibilidade, sem exceções"

Melhor: "O sistema deve ter disponibilidade de 99,9%, com tempo máximo de inatividade de 8 horas por ano"

Quando Usar Cada Metodologia

Histórias de Usuário

Ideal para:

- Projetos ágeis com desenvolvimento incremental
- Quando há necessidade de flexibilidade para mudanças
- Equipes pequenas ou médias com comunicação frequente
- Produtos com alta grau de interação com usuários
- Quando o valor para o usuário é o foco principal

Casos de Uso

Ideal para:

- Sistemas com fluxos de interação complexos
- Projetos com processos mais tradicionais
- Quando há necessidade de documentação detalhada
- Sistemas com múltiplos atores e cenários
- Quando é importante documentar exceções e fluxos alternativos

BDD com Gherkin

Ideal para:

- Projetos com forte integração entre requisitos e testes
- Equipes que praticam automação de testes
- Quando há uma equipe multidisciplinar colaborando
- Para reduzir mal-entendidos entre técnicos e não técnicos
- Quando o comportamento do sistema é o foco principal

IEEE 830

Ideal para:

- Sistemas críticos ou de missão crítica
- Projetos em ambientes regulamentados
- Quando a documentação formal é um requisito
- Sistemas com alto risco ou complexidade
- Contratos e licitações que exigem especificação detalhada

FURPS+

Ideal para:

- Quando é necessário um foco em atributos de qualidade
- Para complementar outras técnicas de especificação
- Para garantir que todos os aspectos do sistema sejam considerados
- Projetos onde qualidade é crítica
- Quando é importante classificar requisitos em categorias lógicas

Abordagem Híbrida

Muitas vezes, a melhor abordagem é combinar diferentes técnicas de especificação de requisitos, adequando-as ao contexto do projeto. Algumas combinações comuns incluem:

- **Histórias de Usuário + BDD**: Para equipes ágeis que desejam automação de testes
- **Casos de Uso + FURPS+**: Para equilibrar interações do usuário com atributos de qualidade
- **Histórias de Usuário + Casos de Uso**: Histórias para planejamento ágil e casos de uso para detalhamento
- **IEEE 830 + FURPS+**: Para sistemas críticos com ênfase em qualidade

Dica: Adapte a abordagem ao contexto do projeto, considerando fatores como cultura organizacional, complexidade do sistema, criticidade, restrições regulatórias e maturidade da equipe.