Verificação, Validação e Priorização de Requisitos

Higor Dantas[1][118110808], Ivyna Alves[2][115110057], Matheus Justino[3][118111780] e Thaís Toscano[4][115111596]

[1] Universidade Federal de Campina Grande, Campina Grande - PB, Brasil
higor.dantas@ccc.ufcg.edu.br
[2] ivyna.alves@ccc.ufcg.edu.br
[3] matheus.justino@ccc.ufcg.edu.br
[4] thais.toscano@ccc.ufcg.edu.br

Resumo Uma das dificuldades enfrentadas no desenvolvimento de software é entender bem os requisitos e, em seguida, reproduzi-los no sistema, já que a linguagem natural tem como forte característica a ambiguidade. Sendo assim, a compreensão de um requisito de forma errônea, traz grandes prejuízos para o projeto, já que fará que a equipe aloque mais tempo para modificar o que já foi implementado, perdendo assim, tempo de desenvolvimento. Dessa maneira, a utilização de estratégias de priorização, verificação e validação de requisitos são importantes para o ciclo de vida de desenvolvimento do software, já que tem o objetivo de evitar tais erros. Diante disso, o trabalho a seguir mostrará estratégias para o elaboração do sistema de forma eficiente e segura.

Keywords: Verificação e validação · Requisitos · Software.

1 Introdução

Um dos grandes desafios para engenharia de software é captar corretamente os requisitos de um sistema. Dessa maneira, podemos elencar o quão é importante o desenvolvimento de técnicas para verificação e validação de requisitos.

Tendo em vista a afirmação feita acima, podemos contar com estratégias para esse desafio, como priorizar os requisitos, sempre validar com o cliente os requisitos que estão sendo desenvolvidos e criar um corpo de testes, com o intuito de verificar se as funcionalidades foram implementadas da forma correta.

Este processo trata da validação quanto à consistência, precisão, contextualização de requisitos levantados no processo de identificação, descoberta, análise e negociação de requisitos. Sendo assim, envolve a revisão de todos os requisitos levantados e negociados, assim como uma validação de modelos e teste de requisitos, com um grande peso para a engenharia de requisitos, pois com um documento de requisitos bem definido, permite a correção de incoerências e inconformidades no desenvolvimento de um produto de software. Por isso, a validação permite minimizar o tempo gasto desses problemas.

Na mesma vertente, também temos a verificação onde busca-se avaliar se o que foi planejado realmente foi realizado. Em outras palavras, se os requisitos e funcionalidades documentados foram implementados da maneira correta. Segundo (SOMMERVILLE, 2011)[2], esse processo envolve verificar se o software está de acordo com suas especificações. Logo, é preciso verificar se ele atende aos requisitos funcionais e não funcionais.

Conforme (SANTOS, 2015)[4], o processo verificação e validação, define que cada produto ou artefato elaborado durante o ciclo de desenvolvimento do software seja avaliado, verificado, validado e testado em cada etapa antes de seguir para a próxima etapa, assegurando que os mesmos estejam completos e corretos, seguindo as exigências de verificação e validação.

Diante disso, podemos valorizar tal processo, pois economiza o tempo de implementação e evita erros no desenvolvimento do sistema. Logo, a garantia da qualidade é uma forma de fazer com que o produto e o processo contemplem os aspectos entendidos como qualidade.

2 Verificação

O principal papel dessa fase é observar se todos os requisitos funcionais e não-funcionais elucidados foram bem definidos, ou seja, não possuem ambiguidades, inconsistências e nem omissões. Logo, essa fase deve ser iniciada o quanto antes já que quanto mais adiantado o processo de desenvolvimento maior se torna o custo de realizar as devidas correções.

Para isso, deve-se ao longo dessa fase realizar algumas verificações nas documentações geradas em fases anteriores no processo. São elas:

- **Verificações de validade**: Uma funcionalidade do sistema pode depender de uma outra funcionalidade adicional ou diferente da relatada pela documentação.
- **Verificações de consistência**: Uma funcionalidade não pode ter descrições diferentes ou restrições que se contradizem.
- **Verificações de completude**: O documento deve conter todas as funções, e suas devidas descrições, necessárias aos usuários do sistema.
- **Verificações de realismo**: Essa verificação é importante para passar a garantia de que todas funcionalidades descritas são possíveis de implementar, considerando o orçamento e o cronograma definido para o desenvolvimento do sistema.
- **Verificabilidade**: Essa atividade garante que é possível escrever casos de testes que cubram todos os requisitos definidos para o sistema. Isso ajuda, principalmente, na redução de conflitos entre o cliente e o contratante.

3 Validação

Segundo (SOMMERVILLE, 2011)[2], finalidade da validação é assegurar que o sistema de software atenda às expectativas do cliente. Dessa maneira, esse processo representa a atividade em que obtemos o aceite do cliente sob determinado

artefato. No cenário de engenharia de requisitos, esta atividade significa aprovar junto ao cliente os requisitos que foram especificados.

4 Técnicas de Verificação e Validação

As formas de realizar as validações e verificações de requisitos são divididas em duas categorias: estáticas e dinâmicas, de modo que esta última se faz necessária ter o código em execução, diferentemente das estáticas, que não possui a dita funcionalidade. No tocante às atividades acima descritas, as referidas podem ser realizadas mediante a utilização de algumas técnicas, quais sejam as inspeções, os walkthroughs e os checklists.

4.1 Verificação e Validação Estática

Inspeções Esse processo possui o objetivo de observar defeitos antes que a fase de testes tenha início, porém existem diversas formas de ser executada, a forma tradicional foi criada por Fagan, em 1986. Contudo, o processo passou por modernizações que fizeram com que esta técnica se adaptasse melhor às formas de desenvolvimento atuais. Logo, atualmente se trabalha com pelo menos 4 pessoas que podem desempenhar mais de um papel.

Os papéis são:

- Autor: responsável pela descoberta de defeitos e suas correções.
- Inspetor: deve encontrar erros ou inconsistências nos artefatos separados para inspeção.
- Relator: registra os resultados das reuniões.
- Moderador: papel que realiza o gerenciamento e busca facilitar a realização das inspeções, além de planejar todas as atividades.

Nas versões mais atuais costuma ter 6 fases basicamente, são elas:

1. **Planejamento**: O moderador define o contexto que ocorrerá a inspeção, seleciona os usuários que desempenharão o papel de inspetor e distribui o material que passará pela inspeção.
2. **Detecção de defeitos**: Os inspetores selecionados anteriormente devem realizar uma atividade de detecção de defeitos, onde será produzido uma lista de desconformidades.
3. **Coleção de Defeitos**: O moderador reúne todas as listas produzidas pelos inspetores no passo anterior e elimina as discrepâncias repetidas, já que podem ser relatadas por mais de um inspetor.
4. **Discriminação de defeitos**: O moderador, o autor do documento e os inspetores realizam discussões das discrepâncias encontradas. Durante esse passo cada problema relatado receberá uma classificação entre falso positivo e defeito. Daí, os reais defeitos são repassados ao autor, que realizará as devidas correções.
5. **Retrabalho**: O autor realiza as correções dos problemas relatados no passo anterior.

6. **Continuação**: O autor repassa o material corrigido para o moderador, que irá realizar uma avaliação da necessidade de uma nova inspeção ou não.

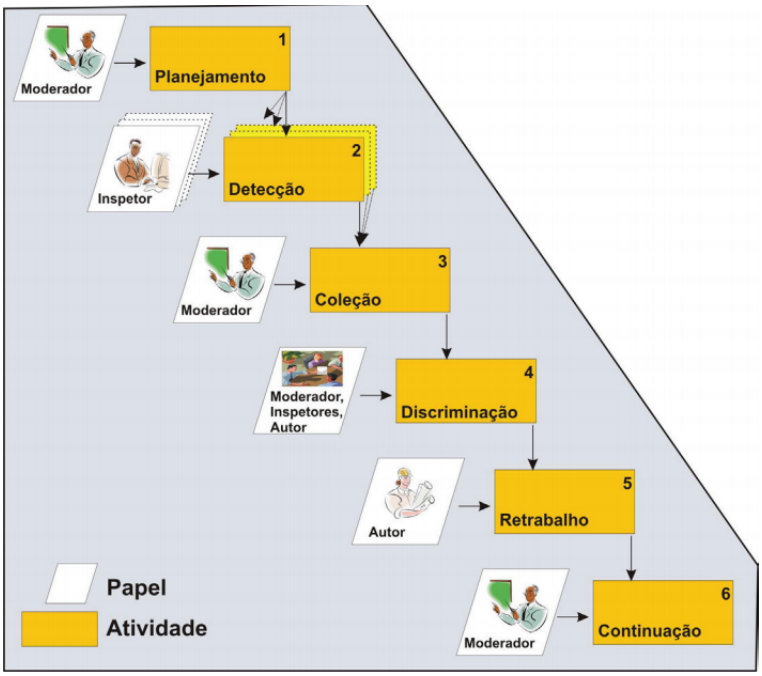

Figura 1. Processo de Inspeção.
Fonte: [3].

Walkthrough É um processo menos rigoroso que o apresentado acima, onde costuma possuir de entre 3 e 5 participantes, com a possibilidade de desempenhar papel de autor, de líder de revisões ou de revisores.

O processo inicia selecionando uma parte do sistema que deverá passar por uma avaliação de no máximo 2 horas, onde cada revisor simula a execução passo a passo no papel para que se detecte algum erro. Porém, é uma técnica bastante desestruturada e para ser bem sucedida é necessário que a equipe tenha bastante dedicação e eficiência.

4.2 Verificação e Validação Dinâmica

Sua principal forma é por meio da execução de testes, onde se executa o programa com entradas específicas e verifica se o comportamento tá dentro do esperado.

A forma dinâmica busca responder, basicamente, três perguntas:

1. *Quando testar?* Onde se está relacionado basicamente à fase de desenvolvimento, que são testes que incluem todas as atividades de testes efetuadas pelos desenvolvedores do sistema.
   - Teste de unidade: Consiste em testar a menor unidade.
   - Teste de integração: Verifica se os componentes funcionam quando estão em conjunto.
   - Teste de sistema: Avalia o software de forma integrada, como um usuário final.
   - Teste de aceitação: São testes que verificam funcionalidades e verificam se as mesmas estão de acordo com o esperado.
   - Teste de regressão: A cada nova versão do software deve-se executar todos os testes já realizados anteriormente. Isso deve ser feito, pois ajuda a verificar se alguma modificação recente causou algum efeito indesejado
2. *O que testar?* Está relacionado com o tipo de teste que será realizado.
   - Teste de funcionalidade: Testa os requisitos funcionais, considerando o documentado gerado inicialmente.
   - Teste de interface: Verifica se a navegabilidade e os objetivos da tela funcionam como esperado.
   - Teste de desempenho: Testa a capacidade de resposta do sistema a uma determinada solicitação.
   - Teste de performance: Analisa o funcionamento da aplicação diante três situações: o sistema sob condições normais de uso (teste de carga), o sistema diante de uma grande quantidade de transações e usuário simultaneamente (teste de stress), e ainda verifica se após um tempo o sistema ainda se mantém de forma satisfatória (teste de estabilidade).
   - Teste de usabilidade: Testa as questões que interfere na relação do sistema com o usuário (facilidade no uso, por exemplo).
   - Teste de volume: Tem a função de testar a quantidade de dados envolvidos em operações normais. Para isso, o teste envolve o banco de dados durante um longo período de tempo.
   - Teste de segurança: Testa a segurança do sistema diante de diversas formas de acessos ilegais.
3. *Como testar?* Utilizado quando se está preocupado apenas com as saídas ou com as estruturas internas.
   - Teste funcional da caixa-preta: Usado quando a preocupação não é com o código produzido, e sim com a saída esperada e obtida.
   - Teste estrutural caixa-branca: É o oposto do anterior, pois representa uma preocupação com a estrutura interna do sistema.

5 Priorização de Requisitos

5.1 Definição

A priorização é uma forma de lidar com demandas conflitantes por recursos limitados. Estabelecer a prioridade relativa de cada capacidade de produto permite planejar a construção para fornecer o maior valor com o menor custo, com

isso o mesmo ajuda o projeto a fornecer o máximo valor comercial o mais rápido possível dentro das restrições do projeto.

Até mesmo um projeto de tamanho médio pode ter dezenas de requisitos do usuário e centenas de requisitos funcionais. Para mantê-lo gerenciável, escolha um nível apropriado de abstração para a priorização - recursos, casos de uso, histórias de usuários ou requisitos funcionais. Dentro de um caso de uso, alguns fluxos alternativos podem ter uma prioridade mais alta que outros.

(DAVIS, 2005) indica que a priorização bem sucedida requer uma compreensão de seis questões:

1. As necessidades dos clientes;
2. A importância relativa dos requisitos para os clientes;
3. O momento em que as capacidades precisam ser entregues;
4. Requisitos que servem como predecessores para outros requisitos e outros relacionamentos entre requisitos;
5. Quais requisitos devem ser implementados como um grupo;
6. O custo para satisfazer cada requisito.

Há requisitos que devem ser implementados juntos ou em uma sequência específica, pois não fará sentido implementar algo que não possa ser desfeito e/ou atualizado, por exemplo, você pode acabar escrevendo código para, digamos, aceitar pagamentos com cartão de crédito sem verificar se o cartão é válido, rejeitando cartões que foram relatados como roubados ou manipulando outras exceções.

5.2 Técnicas

Priorizar requisitos não é algo simples e devido a necessidade cada vez maior do uso de metodologias ágeis no desenvolvimento de softwares surgiram algumas técnicas que auxiliam nesse processo.

In or Out É caracterizado pela formação de um grupo que tem interesse no projeto, o mesmo produz uma lista de requisitos e vão tomando decisões binárias: *Este requisito está dentro ou fora?*

Pairwise comparison and rank ordering O grupo envolvido no projeto atribui uma lista com uma sequência de prioridades exclusivas para cada requisito. Após concluída, faz-se comparações entre pares dentre todos eles, para que no final seja percebido qual membro de cada par tem maior prioridade.

Three-level scale Essa abordagem de priorização agrupa os requisitos em três categorias: alta, média e baixa. Há alguns fatores para que sejam estabelecidos qual nível os requisitos estão. Os requisitos de alta prioridade são importantes e urgentes. Alternativamente, obrigações contratuais ou de conformidade podem

ditar que um requisito específico deve ser incluído, ou pode haver razões comerciais convincentes para implementá-lo prontamente. Requisitos de prioridade média são importantes mas não são urgentes. Os requisitos de baixa prioridade não são importantes, nem urgentes, e ainda há uma quarta categoria onde diz que os requisitos parecem urgentes para alguns interessados, mas eles não são realmente importantes para alcançar os objetivos de negócios.

Ao executar uma análise de priorização com a escala de três níveis, você precisa estar ciente das dependências de requisitos. Você terá problemas se um requisito de alta prioridade depender de outro classificado com prioridade mais baixa e, portanto, planejado para implementação posteriormente.

MoSCoW As quatro letras maiúsculas representam quatro possíveis classificações de prioridade para os requisitos em um conjunto (IIBA 2009):

- Must: O requisito deve ser satisfeito para que a solução seja considerada um sucesso;
- Should: O requisito é importante e deve ser incluído na solução, se possível, mas não é obrigatório para o sucesso;
- Could: É uma capacidade desejável, mas que pode ser adiada ou eliminada. Implemente-o somente se o tempo e os recursos permitirem;
- Won't: isso indica um requisito que não será implementado no momento, mas pode ser incluído em uma versão futura.

Esse esquema altera a escala de três níveis para uma escala de quatro, ele não oferece nenhuma justificativa para tomar a decisão sobre como classificar a prioridade de um determinado requisito em comparação a outros. Ele também possui ambiguidade quanto ao tempo, particularmente quando se trata da classificação "Won't". "Não"pode significar "não no próximo lançamento"ou "nunca", com isso, esta técnica não é recomendada.

$100 (hundred-dollar) Esta técnica consiste em dar à equipe de priorização 100 dólares imaginários para trabalhar, os mesmos alocam esses dólares para "comprar" itens que eles gostariam de ver implementados do conjunto completo de requisitos do candidato. Eles ponderam os requisitos de prioridade mais alta alocando mais dólares a eles. Se um requisito for três vezes mais importante para um interessado do que outro requisito, ela atribui talvez nove dólares ao primeiro requisito e três dólares ao segundo, desta forma faz com que os participantes realizem suas próprias alocações de dólares e, em seguida, somem o número total de dólares atribuídos a cada requisito para ver quais deles, coletivamente, têm a prioridade mais alta.

5.3 Priorização baseada em valor, custo e risco

Uma maneira definitiva e rigorosa de relacionar o valor do cliente aos recursos do produto proposto é com uma técnica chamada Implantação da Função

de Qualidade, ou QFD (Cohen 1995). Os participantes típicos no processo de priorização incluem o gerente de projetos ou analista de negócios, que lidera o processo, arbitra conflitos e ajusta os dados de priorização recebidos dos outros participantes, se necessário, e os representantes do cliente que fornecem as classificações de benefício e penalidade.

Representantes de desenvolvimento, que fornecem as classificações de custo e risco:

1. Listar em uma planilha todos os recursos, casos de uso, fluxos de casos de uso, histórias de usuários ou requisitos funcionais que você deseja priorizar uns contra os outros. Usamos recursos no exemplo. Todos os itens devem estar no mesmo nível de abstração: não misture requisitos funcionais com recursos, casos de uso ou histórias de usuários.
2. Classificar de 1 a 9 os benefícios relativos ao cliente de cada requisito (1 indica que ninguém consideraria útil; 9 significa que seria extremamente valioso).
3. Classificar de 1 a 9 as penalidades caso aquele requisito não seja incluído no projeto (1 significa que ninguém ficará chateado se estiver ausente; 9 indica uma desvantagem séria).
4. A planilha calcula o valor total de cada característica como a soma de suas pontuações de benefício e penalidade. A planilha soma os valores de todos os recursos e calcula a porcentagem do valor total que vem de cada um dos recursos (a coluna Valor%).
5. Classificar de 1 a 9 os custos relativos da implementação, pelos desenvolvedores, (1 – rápido e fácil – a 9 – demorado e caro). Os desenvolvedores estimam as classificações de custo com base na complexidade do recurso, na extensão do trabalho da interface do usuário necessária, na capacidade potencial de reutilizar o código existente, na quantidade de testes necessária e assim por diante.
6. Classificar de 1 a 9 o risco técnico relativo. O risco técnico é a probabilidade de não obter o recurso logo na primeira tentativa. Por exemplo, uma classificação de grau 1 significa que o risco é baixíssimo, enquanto que grau 9 indica sérias preocupações sobre a viabilidade, a falta de conhecimento necessário sobre a equipe, o uso de ferramentas e tecnologias desconhecidas, ou a preocupação com a quantidade de complexidade escondida dentro do requisito.
7. Depois de inserir todas as estimativas na planilha, deve-se calcular um valor de prioridade para cada recurso usando a seguinte fórmula:

$$priority = value\%/(cost\% + risk\%) \tag{1}$$

8. Por fim, deve-se classificar a lista de recursos em ordem decrescente por prioridade calculada, a coluna mais à direita. Os recursos no topo da lista têm o balanço de valor, custo e risco mais favorável e, portanto, todos os outros fatores sendo iguais devem ter a prioridade mais alta.

A utilidade desse modelo de prioridade é limitada pela capacidade da equipe de estimar o benefício, a penalidade, o custo e o risco de cada item.

6 Resultados

Diante do que foi exposto nos tópicos acima, podemos observar que as técnicas de verificação e validação andam lado a lado com o desenvolvimento saudável e a qualidade do sistema, pois em todo o ciclo de vida desse processo, os requisitos são testados e validados com cliente. Além disso, também temos como aliado as técnicas de priorização de requisitos, que por sua vez auxilia na escolha da melhor forma de utilizar o tempo disponível. Portanto, podemos concluir que esse tópico de engenharia de requisitos assegura a corretude do sistema e permite que as etapas do desenvolvimento do software seja entregue com qualidade.

Referências

1. Wiegers, K., Beatty, J.: Software Requirements. 3ª ed. (2013).
2. Sommerville, Ian. Engenharia de software. 9ª ed. (2011).
3. Engenharia de Software Magazine. Ano 1, 1ª ed (2007), 68 – 74. Acesso em: bluehttps://profandreluisbelini.files.wordpress.com/2016/02/revista-engenharia-de-software-ano-1-1c2ba-edic3a7c3a3o.pdf
4. Dos Santos, C. W.: PROCESSO DE VV APLICADO AO DESENVOLVIMENTO DE SOFTWARE DO NTIC. Alegrete (2015). Acesso em: bluehttp://dspace.unipampa.edu.br/bitstream/riu/859/1/Processo%20de%20V%26V%20aplicado%20ao%20desenvolvimento%20de%20software%20do%20NTIC.pdf