Ricardo Sequeira Ferreira

Desenvolvimento, Testes e Qualidade de *Software*

Orientador: Professor Doutor Nuno Manuel Garcia dos Santos

Universidade Lusófona de Humanidades e Tecnologias

Escola de Comunicação, Artes e Tecnologias da Informação

Lisboa

2010

Ricardo Sequeira Ferreira

Desenvolvimento, Testes e Qualidade de *Software*

Dissertação apresentada para a obtenção do Grau de Mestre em Engenharia de *Software* e Sistemas de Informação pela Universidade Lusófona de Humanidades e Tecnologias

Orientador: Prof. Doutor Nuno Manuel Garcia dos Santos

Universidade Lusófona de Humanidades e Tecnologias

Escola de Comunicação, Artes e Tecnologias da Informação

Lisboa

2010

Agradecimentos

Ao concluir a dissertação não posso deixar de expressar uma palavra de agradecimento a todos aqueles que pelo incentivo, sugestões e observações contribuíram para a sua realização.

À família e amigos pelos constantes incentivos, apoio e motivação que manifestaram e que proporcionaram as condições necessárias para que realizasse o mestrado.

Ao meu orientador de mestrado, o professor Nuno Garcia, pelos comentários, sugestões e disponibilidade demonstrada.

A todas as pessoas não mencionadas e que me ajudaram, o meu sincero agradecimento.

Resumo

Hoje em dia o *software* tornou-se num elemento útil na vida das pessoas e das empresas. Existe cada vez mais a necessidade de utilização de aplicações de qualidade, com o objectivo das empresas se diferenciarem no mercado. As empresas produtoras de *software* procuram aumentar a qualidade nos seus processos de desenvolvimento, com o objectivo de garantir a qualidade do produto final.

A dimensão e complexidade do *software* aumentam a probabilidade do aparecimento de não-conformidades nestes produtos, resultando daí o interesse pela actividade de testes de *software* ao longo de todo o seu processo de concepção, desenvolvimento e manutenção.

Muitos projectos de desenvolvimento de *software* são entregues com atraso por se verificar que na data prevista para a sua conclusão não têm um desempenho satisfatório ou por não serem confiáveis, ou ainda por serem difíceis de manter. Um bom planeamento das actividades de produção de *software* significa usualmente um aumento da eficiência de todo o processo produtivo, pois poderá diminuir a quantidade de defeitos e os custos que decorrem da sua correcção, aumentando a confiança na utilização do *software* e a facilidade da sua operação e manutenção.

Assim se reconhece a importância da adopção de boas práticas no desenvolvimento do *software*. Para isso deve-se utilizar uma abordagem sistemática e organizada com o intuito de produzir *software* de qualidade.

Esta tese descreve os principais modelos de desenvolvimento de *software*, a importância da engenharia dos requisitos, os processos de testes e principais validações da qualidade de *software* e como algumas empresas utilizam estes princípios no seu dia-a-dia, com o intuito de produzir um produto final mais fiável. Descreve ainda alguns exemplos como complemento ao contexto da tese.

Palavras-chave: modelos, requisitos, testes, qualidade, *software*.

Abstract

Nowadays the *software* has become a useful element in people's lives and it is increasingly a need for the use of quality applications from companies in order to differentiate in the market. The producers of *software* increase quality in their development processes, in order to ensuring final product quality.

The complexity and size of *software*, increases the probability of the emergence of non-conformities in these products, this reason increases of interest in the business of testing *software* throughout the process design, development and maintenance.

Many *software* development projects are postpone because in the date for delivered it's has not performed satisfactorily, not to be trusted, or because it's harder to maintain. A good planning of *software* production activities, usually means an increase in the efficiency of all production process, because it can decrease the number of defects and the costs of it's correction, increasing the reliability of *software* in use, and make it easy to operate and maintenance.

In this manner, it's recognized the importance of adopting best practices in *software* development. To produce quality *software*, a systematic and organized approach must be used.

This thesis describes the main models of *software* development, the importance of requirements engineering, testing processes and key validation of *software* quality and how some companies use these principles daily, in order to produce a final product more reliable. It also describes some examples in addition to the context of this thesis.

Keywords: models, requirements, testing, quality *software*.

Índice

Figuras

Tabelas

Capítulo 1.
Introdução

A história do *software* é muito recente. Ainda assim, nos últimos anos, a engenharia de *software* enquanto indústria evoluiu significativamente, causando e provocando uma procura pela qualidade do desenvolvimento produzido, verificando-se frequentemente a escassez na disponibilidade de profissionais nas áreas de testes e qualidade. Como consequência da evolução da indústria, os sistemas de informação estão cada vez mais complexos e volumosos, com prazos de desenvolvimento mais curtos e com utilizadores a exigir soluções com maior qualidade.

O mercado de *software* tem exigido dos produtores de *software* um nível de qualidade dos produtos e dos serviços prestados cada vez maior, o que pressupõe uma maior qualidade de todos os actores envolvidos, desde a de engenharia e concepção, passando pela equipa comercial até à equipa técnica de assistência pós-venda. Acrescente-se que com o mercado global, ficou cada vez mais fácil encontrar fornecedores em qualquer parte do mundo, sendo esta uma nova característica do mercado das Tecnologias de Informação, i.e., um fornecedor que ambicione deter uma quota de mercado significativa, quer em volume, quer em especialização, tem que ter também a capacidade de garantir a qualidade de serviço, produção e manutenção do *software* à escala global.

O processo de desenvolvimento de uma aplicação é complexo e envolve uma série de actividades, as quais, apesar das técnicas de controlo usadas, permitem a indesejável existência de erros. Para minimizar a ocorrência de erros, devem usar-se técnicas de teste de *software*, que ao serem postas em prática resultam num aumento da qualidade, eficácia e numa diminuição de custo.

No sector da produção do *software*, com as progressivas exigências do mercado, têm havido várias iniciativas com o intuito da elaboração de normas e padrões internacionais para a melhoria da qualidade do *software*, quer em termos do produto final, quer em termos do processo de desenvolvimento.

Assim, neste contexto, foram criados processos e certificações com o intuito de auxiliar as empresas fornecedoras e os clientes a procurar uma evolução constante na qualidade, unificando todos os interessados em torno do mesmo objectivo, ou seja, numa maior consistência do produto, aumentando a sua produtividade, eficiência do produto e nível serviço a ele associados.

A presente dissertação é uma abordagem ao estudo compreensivo e detalhado dos modelos de desenvolvimento, da engenharia de requisitos, dos processos de testes e de validação da qualidade de *software*, incluído uma descrição de como algumas empresas têm utilizado estes princípios no seu processo de desenvolvimento, esperando que este trabalho contribua para a promoção da importância da prevenção dos erros e da melhoria contínua da qualidade de *software*.

Esta dissertação está organizada da seguinte forma: este capítulo consiste na introdução ao tema e na definição dos objectivos deste trabalho; o Capítulo 2 faz uma apresentação dos principais modelos de desenvolvimento utilizados; o Capítulo 3 está orientado para a importância da engenharia dos requisitos, começando com uma visão geral da engenharia dos requisitos, apresentando as principais actividades, a importância da utilização da engenharia de requisitos e linguagens de especificações, a gestão de requisitos e por fim os requisitos funcionais e não funcionais; o Capítulo 4 está orientado para os testes usados no desenvolvimento de *software* começando com uma abordagem inicial aos testes de *software*, passando pelas principais técnicas usadas e acabando pelo planeamento e ferramentas automáticas de testes; o Capítulo 5 está orientado para a qualidade de *software*, começando com uma retrospectiva histórica e passando pela identificação dos principais conceitos, garantia, confiabilidade, custo, principais normas, métricas e planeamento; no Capítulo 6 apresentam-se alguns casos de estudo, nos quais se pode ver como as empresas abordam a temática dos testes e qualidade do *software*; finalmente no Capítulo 7 são apresentadas as considerações finais para este trabalho.

Capítulo 2.

Processos de testes e qualidade no desenvolvimento de *software*

2.1. Introdução

Um processo de desenvolvimento de *software* é um conjunto actividades que conduzem à criação de um produto de *software* [1]. Os modelos de processos de desenvolvimento de *software* são uma descrição do caminho a seguir com o objectivo de passar os requisitos do cliente [2], englobam actividades como por exemplo a especificação, codificação, implementação, manutenção e testes, caracterizando–se frequentemente pela existência de interacção entre estas actividades.

Nesta secção são analisados os principais modelos ditos clássicos ou não-ágeis de ciclos de vida do desenvolvimento do *software* e embora sejam modelos diferentes, estuda-se ainda a forma como estes podem ser combinados entre si.

2.2. Modelo de desenvolvimento em cascata

Vários autores [1 - 3] fazem referência a este modelo, embora com diferentes nomes, como por exemplo ciclo de vida clássico, modelo de desenvolvimento sequencial linear ou modelo de desenvolvimento em cascata. O nível de detalhe do modelo dependerá de autor para autor, e a Figura 1 mostra uma possível sequência de fases.

Este tipo de modelo implementa o registo da sequência das actividades, seguindo a possibilidade de permitir interacções entre as várias etapas e até, caso seja necessário, desenvolver actividades alternativas [2]. Caso se tenha identificado uma nova funcionalidade ou observado algum erro, devem-se redefinir os requisitos do utilizador.

Segundo Sommerville [1], as principais vantagens do uso deste modelo são a documentação que se produz no final de cada fase, tendo como desvantagens a sua relativa inflexibilidade, resultante da divisão do projecto em etapas distintas. Devem fazer-se compromissos nas etapas ao longo do desenrolar do projecto, com o intuito de se atingirem os objectivos e de não ser posta em causa a conclusão atempada do projecto dentro do orçamento.

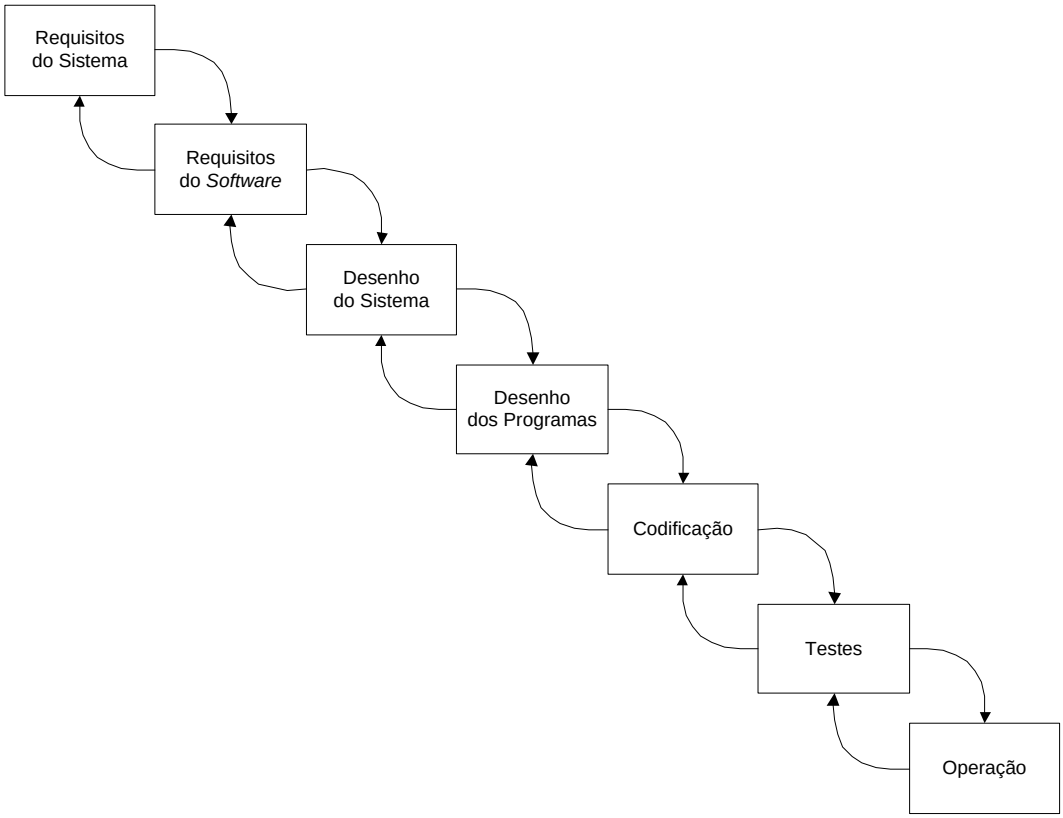

Figura 1 – Modelo de desenvolvimento em cascata

(adaptado de [2]).

Como se pode verificar na figura 1, o modelo têm as seguintes actividades, descritas resumidamente:

- **requisitos do sistema** – nesta fase define-se aquilo que o utilizador pretende do sistema;

- **requisitos do *software*** – definem-se os atributos do sistema para os quais o utilizador pretende para a aplicação;

- **desenho do sistema** – define-se a implementação de um sistema com os atributos pretendidos;

- **desenho do programa** – define-se a implementação do programa ou módulo, com cada um dos atributos previamente definidos;

- **codificação** – nesta etapa é criado o código fonte dos programas;

- **testes** – o produto de *software* é testado e caso ocorra algum erro ou não conformidade com as especificações, é removido ou corrigido nesta fase;

- **operação** – depois da aplicação estar desenhada, codificada e testada, é colocado em operação o produto de *software*.

2.3. Modelo de desenvolvimento em protótipo

É frequente o cliente definir um conjunto de objectivos gerais para o *software*. Mas muitas vezes o programador tem dúvidas sobre a eficiência ou da adaptabilidade do sistema, quando implementado. Nestas situações o modelo de protótipo pode ser a melhor abordagem.

Segundo Pressman [3], este modelo prevê a seguinte sequência de etapas, brevemente descritas:

- **definição de requisitos –** nesta etapa os objectivos gerais são definidos de comum acordo entre o programador e o cliente;

- **realização do projecto –** nesta etapa o programador concentra-se nos aspectos que serão visíveis aos clientes e utilizadores;

- **avaliação do protótipo –** as funcionalidades do protótipo são avaliadas pelo utilizador e em seguida refinam-se os requisitos, que serão posteriormente desenvolvidos num novo protótipo ou finalmente, na versão final;

- **conjunto de interacções –** esta etapa consiste num conjunto de interacções durante as quais o *software* é ajustado, com o objectivo de satisfazer as necessidades do cliente e fazer com que o programador entenda o que é preciso ser feito.

Aqui entende-se por cliente o grupo de utilizadores finais da aplicação e o programador o grupo de pessoas encarregues do desenvolvimento do produto de *software*.

Este modelo tem como principal vantagem a de ser visto como um primeiro sistema, ajudando os programadores a convergir mais rapidamente para a solução final. Como desvantagens tem, para o lado do cliente, ser um programa feito com erros imprevistos e funcionalidades incompletas, podendo não ter a qualidade global desejada; para os programadores, este modelo poderá resultar na escolha de soluções menos apropriadas, uma vez que o planeamento e a análise total dos requisitos são sacrificados com o objectivo de permitir rapidamente um protótipo executável.

Apesar de poderem ocorrer vários problemas neste modelo, é bastante importante definir as regras do jogo, isto é, a aplicação deste modelo deverá ser vista como um mecanismo para definição de requisitos, com o objectivo da busca da qualidade e prossecução da solução final.

2.4. Modelo de desenvolvimento em espiral

Este modelo de processo de *software* foi originalmente proposto por Boehm [2]. Como mostra a Figura 2, representa uma sequência de actividades, em cada ciclo da espiral identifica uma fase do processo de desenvolvimento de *software*. A espira mais interior representa a viabilidade do sistema, seguindo-se a definição dos requisitos, passando pela fase de desenho de sistema e assim sucessivamente.

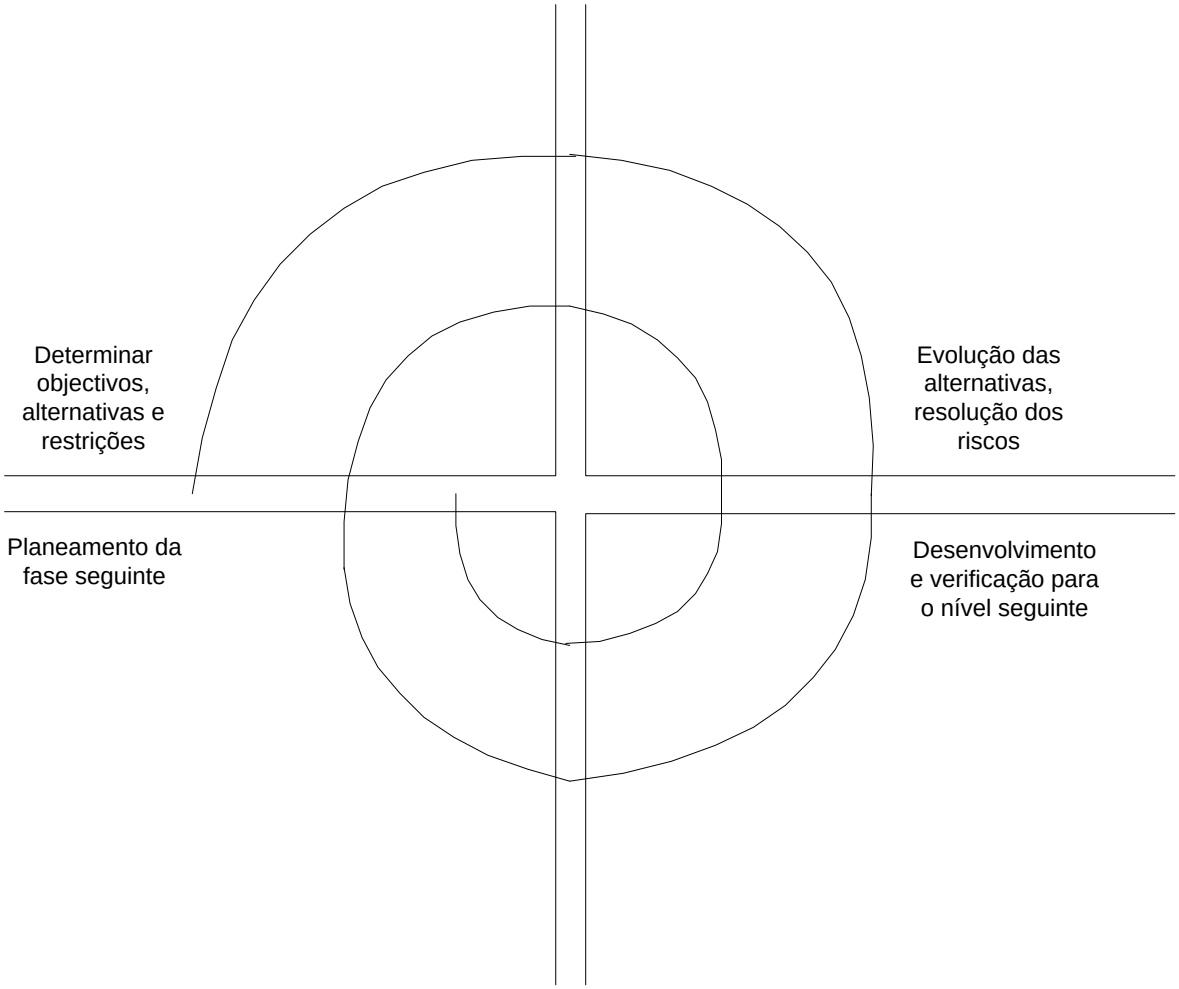

Figura 2 – Modelo de desenvolvimento em espiral

(adaptado de [1]).

Para Somerville [1], este modelo pode-se dividir nas seguintes fases, brevemente descritas como segue:

- **determinar objectivos, alternativas e restrições** – o projecto começa aqui, e nesta fase definem-se os objectivos do projecto, identificam-se as restrições do processo de desenvolvimento e do produto e planeiam-se as alternativas;

- **planeamento da fase seguinte** – nesta fase toma-se a decisão de se se deve continuar para a fase seguinte do projecto;

- **evolução das alternativas, resolução dos riscos** – aqui leva-se a cabo uma análise detalhada dos riscos e definem-se os passos para reduzir estes riscos;

- **desenvolvimento e verificação para nível seguinte** – nesta fase escolhe-se um modelo para desenvolvimento do sistema e implementa-se de forma a atingir os objectivos inicialmente planeados.

A tabela 1 mostra as vantagens e desvantagens do modelo em espiral, de acordo com Miguel [2].

Vantagens	*Desvantagens*
Processo bastante flexível e adaptável a alterações de requisitos	Mais difícil gerir
Pode disponibilizar o produto mais rapidamente no mercado	Mais difícil determinar a situação do projecto
Pode melhorar a qualidade do produto entregue	Risco maior
Melhor ajuste do produto aos requisitos do utilizador	Compromisso mais difícil com o cliente

Tabela 1 – Vantagens e desvantagens do modelo de desenvolvimento em espiral (adaptado de [2]).

2.5. Modelo de desenvolvimento de entrega incremental

Este tipo de modelo têm como objectivo a entrega de funcionalidades parciais, onde cada versão do produto entregue é semelhante à anterior, mas integrando novas funcionalidades [2]. A figura seguinte mostra-nos este modelo em forma de esquema.

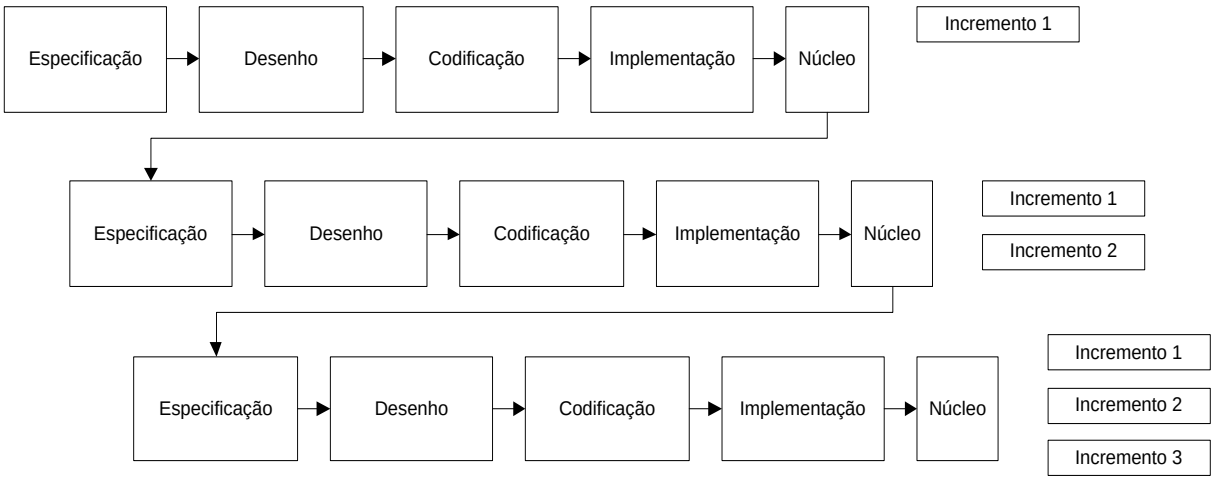

Figura 3 – Modelo de desenvolvimento de entrega incremental

(adaptado de [2]).

Para Sommerville [1], este tipo de modelo tem como principais vantagens as seguintes características:

- a entrega incremental do sistema permite obter uma validação da versão anterior, quer pela equipa de desenvolvimento e testes quer, eventualmente, pelo próprio cliente; existe um baixo nível de falha total do projecto, pois as falhas são elas também resultantes do incremento;

- os serviços e funcionalidades com mais prioridade são entregues em primeiro, possibilitando encontrar menos falhas nas partes mais importantes do sistema.

Em contrapartida tem as seguintes desvantagens:

- os incrementos podem ser muitos e pouco relevantes, causando dificuldade na implementação de todos os requisitos do cliente num tempo e orçamento apropriados;

- alguns sistemas podem requerer um conjunto de recursos que utilizam partes diferentes do sistema e os requisitos dessas partes podem não estar detalhados aquando da entrega do incremento.

2.6. Modelo de desenvolvimento em V

O modelo em V foi desenvolvido originalmente na Alemanha pela IABG, destinado a ser utilizado pelo Ministério Federal da Defesa [2], mas depressa passou a ser utilizado a nível internacional. Este tipo de modelo é constituído por dois grandes grupos: o da especificação e o da verificação e validação, como se pode ver na figura seguinte (Figura 4).

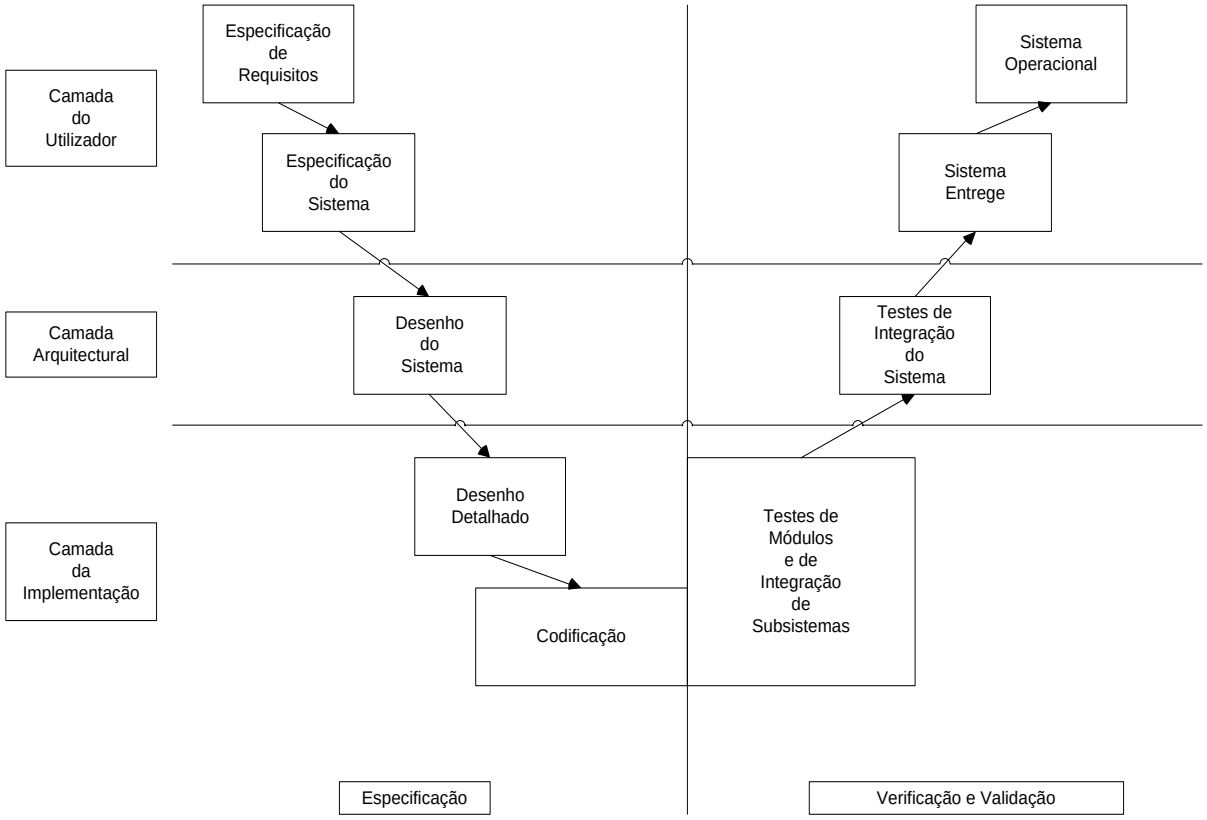

Figura 4 – Modelo de desenvolvimento em V

(adaptado de [2]).

Ao longo da fase da especificação existe um conjunto de etapas progressivas, constituindo um processo de desenho, que vai desde os requisitos à codificação. Na fase de verificação e validação, o sistema deve demonstrar satisfazer as especificações referidas na primeira parte, através de um conjunto de testes.

Como se pode verificar na figura 4, existem várias camadas transversais no modelo, que são as seguintes:

Camada do Utilizador

Esta camada apresenta o sistema do ponto de vista do utilizador, contendo as seguintes fases do projecto:

- **especificação de requisitos** – o utilizador descreve as funções específicas do sistema;
- **especificação do sistema** – descreve o sistema que satisfaz os requisitos da especificação inicial;
- **sistema entregue** – nesta fase o utilizador verifica se o sistema entregue está de acordo com a especificação previamente definida;
- **sistema operacional** – o utilizador valida o sistema de acordo com a especificação dos requisitos.

Camada Arquitectural

Esta camada apresenta a arquitectura do sistema e descreve o modo como a estrutura funcional do sistema se insere no ambiente da infra-estrutura, onde se incluem as seguintes fases:

- **desenho do sistema** – são identificados os principais componentes do sistema, as respectivas relações lógicas e estrutura dos dados que suportam a arquitectura;
- **testes de integração do sistema** – são definidos e realizados os testes de integração do sistema.

Camada da Implementação

Esta camada regista a forma como é implementada a arquitectura, tendo as seguintes fases:

- **desenho detalhado** – define os respectivos atributos funcionais e não funcionais;
- **codificação** – fase onde se faz a escrita do código fonte;
- **testes de módulos e de integração de subsistemas** – nesta fase testam-se cada um dos módulos desenvolvidos e a integração com os outros módulos do sistema.

Segundo Miguel [2], as principais vantagens deste modelo são as de fornecer uma estrutura global do projecto, com uma sequência bem definida de fases. Em relação às desvantagens, há a referir que uma opção de compromisso se pode vir a transformar num risco e desta forma, os respectivos erros virem a ser descobertos demasiado tarde no processo de desenvolvimento.

2.7. Modelo de desenvolvimento baseado em componentes

Nos últimos anos tem aumentado o interesse no desenvolvimento denominado em *software* baseado em componentes. Como o próprio nome sugere, este modelo usa módulos de *software* que foram construídos para outros projectos e promove a sua reutilização no contexto de projecto diferente [1]. Este tipo de *software* é baseado no paradigma de programação orientado aos objectos, cujo desenvolvimento do produto enfatiza a criação de classes que encapsulam os dados e algoritmos a manipular [3].

Segundo Pressman [3] e Sommerville [1], a utilização deste tipo de modelo pode ter como benefícios a redução do prazo de ciclo de desenvolvimento, a redução de custo do projecto e o aumento da produtividade. De acordo com Sommerville as desvantagens deste tipo de modelo têm a ver com o controlo de novas versões, pois havendo várias versões de vários módulos, poderá perder-se o conceito do histórico do desenvolvimento do sistema, ou ainda mais grave, perder funcionalidades causadas por incompatibilidade de funcionamento entre módulos de

diferentes versões, fundamentalmente porque os módulos reutilizados são construídos sem que esse desenvolvimento tenha em consideração a sua aplicação real num projecto em particular.

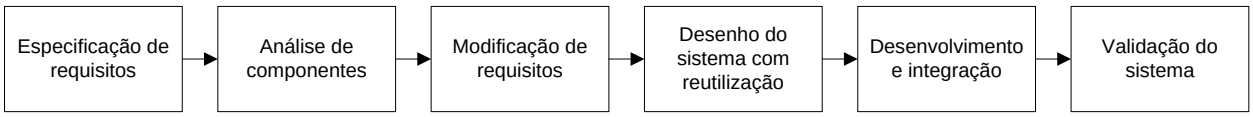

Figura 5 – Modelo de desenvolvimento baseado em componentes

(adaptado de [1]).

Como podemos verificar na figura 5, o modelo baseado em componentes tem as seguintes fases:

- **especificações de requisitos** – nesta fase existe a preocupação com a definição e especificação e requisitos, ou seja, deve-se ter uma visão geral dos processos de negócio;

- **análise de componentes** – através da especificação dos requisitos, faz-se a análise dos componentes que é necessário implementar;

- **modificação de requisitos** – nesta etapa, os requisitos são analisados utilizando a informação acerca dos componentes desenvolvidos, podem alterar-se os requisitos para satisfazer os componentes disponíveis, ou se necessário, desenhar novo *software*;

- **desenho do sistema com reutilização** – nesta fase desenha-se o sistema com base na reutilização dos componentes e caso não existam, desenham-se novos componentes de *software*;

- **desenvolvimento e integração** – nesta fase criam-se os componentes e integram-se os vários módulos no sistema;

- **validação do sistema** – faz-se a verificação de que o sistema está de acordo com o que foi definido na fase de especificações de requisitos.

2.8. Conclusão

Neste capítulo foram descritas as principais características dos modelos de desenvolvimento de *software*, e apresentados os diferentes modelos de desenvolvimento. Adoptando a utilização de um modelo de desenvolvimento de *software*, é possível aumentar a qualidade e produtividade no desenvolvimento, tendo maior controlo sobre este processo, em particular no que respeita a custos e prazos. Como é expectável, os benefícios tendem a aparecer quando os intervenientes têm formação adequada, e tempo para se adaptar e familiarizar com uma dada metodologia.

Capítulo 3.

A importância da engenharia dos requisitos

3.1. Introdução

A engenharia de requisitos é um processo que tem vindo a ser utilizado com o objectivo de melhorar a qualidade de *software*, pois encontravam-se muitas falhas aquando do processo de desenvolvimento do *software*. É frequente recorrer às diferentes actividades de engenharia de requisitos para produzir um documento de requisitos adequado e que posteriormente resulte num *software* de qualidade.

Os clientes actuais beneficiam destas metodologias, pois permitem uma diminuição de custos e contribuem para que os engenheiros de requisitos ajudem os utilizadores a um melhor controlo dos requisitos e a reutilização da documentação produzida.

Neste capítulo são descritos as principais características da engenharia de requisitos de *software*. É apresentada uma visão geral da engenharia dos requisitos, as principais actividades da engenharia de requisitos, linguagens de especificação, a gestão dos requisitos, requisitos funcionais e não funcionais e as regras de ouro da engenharia de requisitos do *software*.

3.2. Visão geral da engenharia dos requisitos

A engenharia de requisitos tem como finalidade descobrir os objectivos para os quais o sistema de *software* se destina [4], envolvendo as actividades de descoberta, documentação e manutenção dos requisitos para um determinado sistema. É através dos levantamentos das necessidades de todas as partes envolvidas e da consequente elaboração da documentação dos requisitos que se faz a identificação dos pontos a abordar, das funcionalidades a implementar e até dos testes a realizar no desenvolvimento da aplicação.

Os requisitos são definidos na fase inicial do desenvolvimento do sistema como especificações a ser implementadas, descrevendo o comportamento, as propriedades e os atributos do sistema, reflectindo as necessidades dos *stakeholders*. Entendam-se como *stakeholders* todas as partes interessadas no desenvolvimento da aplicação.

Segundo Turine e Masiero [5], o IEEE (*Institute of Electrical and Electronics Engineers*) define como requisito de *software*:

- condição ou capacidade necessitada por um utilizador para resolver um problema ou atingir um objectivo;
- condição ou capacidade que deve ser satisfeita no sistema ou componente do sistema com o objectivo de satisfazer um contracto, padrão ou especificação;
- representação documentada de uma condição ou capacidade.

3.3. Principais actividades da engenharia dos requisitos

3.3.1. Identificação de requisitos

Nesta fase é efectuado o levantamento dos requisitos para um sistema futuro [4]. O principal objectivo é identificar o domínio da aplicação, identificar os serviços que o sistema deve proporcionar, o desempenho e restrições de uso do sistema. Estes requisitos são obtidos através de: entrevistas, questionários, pesquisas e modelos de processos, documentação, contractos com fornecedores, actas de reuniões e técnicas de desenvolvimento [4], [5].

Na fase de identificação dos requisitos podem ser encontradas várias dificuldades, o que pode ocasionar erros na obtenção dos requisitos devido a:

- ideias imprecisas do utilizadore sobre o que pretende do sistema;
- dificuldades dos utilizadores em descrever o domínio do problema;
- pontos de vista do sistema diferentes, entre analistas e utilizadores;
- aversão do utilizador em utilizar o sistema, não aceitando fazer parte da identificação de requisitos [6].

A identificação de requisitos é uma actividade onde inicialmente se definem os objectivos organizacionais, seguindo-se a aquisição de informação de *background* e de

enquadramento do sistema a desenvolver, de forma a organizar e sistematizar a informação recolhida em estágios anteriores. Numa etapa muito importante deste processo, os *stakeholders* devem ser consultados metodicamente para a descoberta dos requisitos [6].

3.3.2. Análise e negociação de requisitos

Esta fase é muito importante para o sucesso do processo de desenvolvimento do produto de *software*. Nesta fase, o engenheiro de *software* específica as funções, o desempenho e as restrições do sistema.

O principal objectivo da fase de análise de requisitos [5] é avaliar e conferir o documento de requisitos do *software*, obtendo uma especificação de requisitos completa e consistente. Por isso é provável que o documento de requisitos obtido tenha várias inconformidades ou problemas de funcionalidades e o analista deve ser capaz de detectar e resolver as incoerências, melhorando o documento de requisitos para as fases subsequentes.

Esta etapa pode incluir um esforço de negociação para a definição formal dos requisitos do sistema, já que tipicamente, diferentes grupos de *stakeholders* têm expectativas diferentes e frequentemente incompatíveis entre si do que devem ser as estratégias de desenvolvimento e as funcionalidades do sistema a desenvolver. Por exemplo, a administração poderá estar focada em aspectos orçamentais do desenvolvimento, desejando um curto período de desenvolvimento e implementação, enquanto o utilizador final do sistema poderá preferir um ciclo de desenvolvimento alargado e adaptativo.

3.3.3. Modelação de requisitos

Esta fase tem como principal objectivo o desenvolvimento de modelos que descrevem o que o sistema deverá fazer [5], permitindo um maior entendimento da aplicação e fornecendo uma transição para a fase do projecto.

3.3.4. Validação de requisitos

Durante a fase de validação de requisitos, os analistas devem validar se os requisitos das especificações vão ao encontro das necessidades expressas pelos *stakeholders*. Neste contexto são identificados alguns problemas, como por exemplo [7]:

- falta de conformidade com os padrões de qualidade;
- requisitos ambíguos ou mal formulados;
- erros conceptuais no desenho de sistema;
- conflitos entre requisitos não identificados na fase de análise.

3.3.5. Documento de requisitos do *software*

O documento de requisitos do *software* contém os requisitos funcionais e de qualidade do sistema [5]. Este documento é um meio de comunicação entre o analista e o cliente, com o objectivo de estabelecer um acordo sobre o *software* pretendido. Devido à sua importância, é fundamental que esteja organizado de forma a melhorar a compreensão dos requisitos, evitando que surjam erros na fase da implementação da aplicação. No documento de requisitos, deverão ser evitados os seguintes riscos no documento [6]:

- **não incluir requisito crucial** – este erro implica previamente que os requisitos deverão estar classificados por prioridades. Caso haja um erro de classificação (classificando como requisito secundário um requisito primário) ou caso não se inclua, por total lapso, um requisito vital, poderá verificar-se um aumento do trabalho a desenvolver para corrigir o erro. Caso esta omissão seja detectada apenas na fase de testes, pode ser necessário fazer mudanças na estrutura do sistema e nos componentes previstos, o que frequentemente pode significar recomeçar todo o processo;
- **falta de inspecções nos requisitos** – ao fazer regularmente inspecções, podem detectar-se defeitos precocemente, diminuindo o trabalho repetido e os custos a eles associados;

- **representação inadequada de clientes** – se um conjunto de clientes / utilizadores forem representados inadequadamente, é provável que as suas necessidades em relação ao *software* que está a ser desenvolvido não esteja presentes no documento;

- **busca da perfeição nos requisitos antes de começar a construção do *software*** – as alterações dos requisitos são inevitáveis, e, portanto é necessário o controlo das mudanças, devendo o responsável do projecto estar atento às solicitações de alteração e procurando o preenchimento de lacunas antes da implementação estar concluída. Mais ainda, e porque as estruturas de sucesso são as que mostram adaptabilidade, os requisitos devem ser definidos com a inerente assunção da necessidade da sua posterior adaptação à realidade dos *stakeholders*;

- **modelar apenas aspectos funcionais** – a ênfase na identificação de requisitos costuma ser maior em relação a requisitos funcionais e não tanto nos atributos não funcionais, que estão relacionados com a qualidade do *software*;

- **estabelecer requisitos não implementáveis ou não evolutivas –** devem considerar-se as restrições de *hardware* e *software* para o projecto em questão, pois o custo final do produto deve estar de acordo com o mercado ao qual se destina e precisa ser competitivo;

- **definir requisitos incorrectamente** – os requisitos devem ser validados pelos *stakeholders* antes da fase de desenho iniciar, com o objectivo de não registar requisitos incorrectos.

3.4. Linguagens de especificação

Devido à complexidade elevada que o *software* possui, existe a necessidade de definir com rigor todos os seus componentes, com a finalidade de produzir aplicações sem falhas [8]. As linguagens de especificação usadas na área de análise de requisitos são o instrumento que permitem criar aplicações com maior fiabilidade.

3.4.1. Vantagens

As principais vantagens na utilização de linguagens de especificações são as seguintes [8]:

- melhor compreensão dos requisitos do sistema;
- provar que as especificações do *software* foram cumpridas;
- identificar e definir casos de testes mais relevantes;
- identificar e corrigir erros no inicio do projecto, diminuindo os custos do projecto.

3.4.2. Desvantagens

As principais desvantagens no uso de especificações são as seguintes [8]:

- dificuldade em demonstrar que com as especificações, o custo total do projecto será mais baixo;
- dificuldade em desenvolver as especificações formais, devido à falta de experiência do analista;
- o cliente pode não estar familiarizado com a linguagem de especificação;
- dificuldade em especificar componentes interactivos.

3.4.3. Desmistificação de convicções relativas aos métodos formais

Apresentar-se-ão de seguida algumas evidências relacionadas com os métodos formais, que são habitualmente interpretadas de um modo incorrecto, pelo qual se tentará clarificar essas evidências [8]:

- o *software* pode incluir erros e omissões, ao contrário da pretensão dos métodos formais da garantia da infalibilidade do *software*;

- os métodos formais para além de serem usados como prova do programa, são usados também para escrever a especificação formal e proporcionam uma abstracção da especificação;

- os métodos formais reduzem o custo total, sendo usados em todos os sistemas de segurança e não apenas nos sistemas de segurança críticos, como pretende o mito de que os métodos formais são apenas utéis a sistemas críticos, devido ao seu elevado custo;

- o mito de que os métodos formais só podem ser implementados por matemáticos não é verdadeira, porque a matemática utilizada é simples e a sua única dificuldade reside na abtracção do mundo real;

- os métodos formais apesar de terem um custo inicial maior, diminuem o custo total do projecto, porque dispendem mais tempo de especificação e menos tempo nas fases seguintes, pelo que não aumentam o custo de desenvolvimento como pretende o mito;

- que os métodos formais são inaceitáveis pelos utilizadores é uma convicção relativa aos métodos formais; pelo contrário, os métodos formais ajudam os utilizadores, porque as especificações capturam não só os que os utilizadores querem antes de o produto ser desenvolvido, como o projecto pode ser compreendido pelo cliente, se for traduzido para linguagem natural, através da criação de protótipo que exemplifique a especificação;

- os métodos formais são utilizados, não apenas, para sistemas simples, mas também para sistemas críticos.

3.5. Gestão de requisitos de *software*

Os requisitos são entidades que podem ter prazos de validade muito curtos, devido a diversos factores como por exemplo mudanças externas ao ambiente (alteração de legislação, alterações de mercado ou alteração de posicionamento estratégico da empresa), erros ocorridos

no processo de requisitos, variação no estado do mercado, entre outras. Por estas razões deve-se ter em consideração as seguintes actividades na gestão de requisitos [9]:

- **controlo de mudança –** uma das maneiras utilizadas para organizar as mudanças é utilizando uma biblioteca de versões, o que permite ajudar no controlo de versões;

- **gestão de configuração –** tem como objectivo definir critérios que permitam realizar modificações de configuração, mantendo a consistência e integridade do *software* com as especificações, minimizando os problemas relacionados com o processo de desenvolvimento, através do controlo sobre as modificações;

- **rastreabilidade –** tem como objectivo a capacidade de acompanhar a vida de um requisito em ambas as direcções do processo de *software* e durante todo o seu ciclo, podendo ser de dois tipos: pré-rastreabilidade (rasto que fundamenta a criação do requisito) e pós-rastreabilidade (rasto que se forma a partir do requisito criado, pela mesma alteração).

Requisitos que são mal interpretados pelo utilizador frequentemente causam falhas graves no projecto de *software*. Muitas organizações de desenvolvimento estão a melhorar nos métodos que usam na recolha, análise, documentação e gestão dos requisitos. Tradicionalmente existem as seguintes limitações na documentação de requisitos de *software* [10]:

- dificuldade em gerir a documentação (por exemplo, documentação em excesso ou demasiado extensa, ou inversamente, documentação escassa ou com falhas no conteúdo, e ainda, documentação escrita em linguagem que não é facilmente compreensível, entre outros);

- dificuldade em comunicar mudanças aos membros afectos ao projecto;

- dificuldade em armazenar informações complementares sobre cada requisito estando esta dificuldade sobretudo relacionada com a importância da escolha da plataforma de suporte à informação do processo de desenvolvimento;

- dificuldade em definir ligações entre requisitos funcionais e casos de uso, código, testes e tarefas de projecto.

Por estas razões é importante o uso de ferramentas de gestão de requisitos que armazenem os requisitos e disponibilizem uma solução robusta. Estas ferramentas fornecem funções para manipular e visualizar o conteúdo de base de dados, importação e exportação de requisitos, definir ligações entre requisitos e ligações entre requisitos e outras ferramentas de desenvolvimento de *software*.

Em seguida apresentam-se algumas tarefas de gestão de requisitos que essas ferramentas podem ajudar a executar [10]:

- **gestão de versões e mudanças –** o projecto deve definir uma linha base de requisitos, um conjunto específico de requisitos que a versão deverá conter e a história das mudanças feitas de cada requisito o que irá explicar as decisões anteriores e permitir reverter para uma versão anterior;

- **armazenamento dos atributos dos requisitos –** deve armazenar-se uma variedade de informação ou atributos sobre cada requisito, para que qualquer pessoa que trabalhe no projecto possa ser capaz de ver os atributos, seleccionar-los e modificar-los individualmente. Cada uma das ferramentas de gestão de requisitos deverá gerar vários atributos definidos, como data de criação, número de versão e atributos de adicionais como autor, pessoa responsável, número de lançamentos, prioridade, custo e risco;

- **ligação dos requisitos a outros elementos do sistema –** o rastreamento individual dos requisitos para outros componentes do sistema, permite que a equipa não negligencie inadvertidamente os requisitos durante a implementação. Podem definir-se ligações entre diferentes tipos de requisitos e os vários subsistemas, e por fim, ao analisar o impacto de uma mudança de um requisito específico, as ligações de rastreabilidade devem revelar os outros elementos do sistema que podem afectar a mudança;

- **estado do rasto –** o acompanhamento do rasto de cada requisito durante o desenvolvimento ajuda a controlar o estado geral do projecto;

- **ver subconjuntos de requisitos –** classificar, filtrar ou consultar a base de dados para ver subconjuntos de requisitos e os valores dos atributos específicos;

- **controlo de acesso –** definição de permissões para utilizadores e grupos de utilizadores, podendo assim a partilha de informações de requisitos com todos os membros da equipa, mesmo que estejam geograficamente separados;

- **comunicação entre as partes interessadas –** a maior parte das ferramentas de gestão de requisitos permitem que a equipa discuta algumas ideias sobre os requisitos, utilizando *mails* e ferramentas *on-line*.

Em seguida apresenta-se um conjunto de ferramentas de gestão de requisitos [11]:

Accompa	Accept 360	Active!Focus
AnalystPro	Caliber-RM	Clariys
CORE	Cradle	DOORS & DOORS Require IT
Enterprise Architect	GatherSpace	GMARC
IRqA	Jama Contour	Leap SE
Lighthouse RM	Mac A&D and Win A&D	MKS Requirements
objectiF	Open Source RM	Optimal Trace
PACE	PixRef Pro	Polarion Reqs
Projectricity	Rally	RaQuest

Reconcile	Reqtify	Requirement Tracing System
Requisite Pro	RMTrak	RTM Workshop
SoftREQ	Teamcenter	TestTrackRM
TopTeam Analyst	Tracer	TRUEreq
XTie-Requirements Tracer		

Tabela 2 – Ferramentas de gestão de requisitos

(adaptado de [11]).

3.6. Requisitos funcionais

Os requisitos funcionais são, por definição, as descrições das funções e restricções do sistema [9].

É através da engenharia de requisitos que os engenheiros de *software* conseguem compreender melhor o problema que terão que resolver. Os requisitos descrevem as funcionalidades do produto final, de uma forma completa e consistente, sendo aquilo que o utilizador espera que o sistema ofereça.

Uma especificação completa contém em si todas as condições que se aplicam ao requisito, expressando uma ideia completa. Deve ser consistente, não apresentando contradições ou conflitos, quanto à ambiguidade da interpretação dos requisitos, à implementação do requisito, cumprimento dos prazos e orçamentos estabelecidos [9].

3.7. Requisitos não funcionais

Requisitos não funcionais são os atributos que não estão relacionados com a funcionalidade do sistema (também chamados por atributos de qualidade) [9], sendo restrições sobre serviços ou funções oferecidas pelo sistema. Tem um papel importante durante o

desenvolvimento do sistema, podendo ser usados como critérios de selecção, estilo arquitectural e forma de implementação.

3.8. Regras de ouro dos requisitos de *software*

Neste capítulo serão apresentadas as regras de ouro que a engenharia de requisitos de *software* deve ter em conta [12]:

- **medir os benefícios da exigência de teste com base nos testes de desenvolvimento –** constituir um conjunto de métricas com o objectivo de medir os benefícios retirados das necessidades de teste com base em comparação com o teste completo, iniciado a seguir à fase de desenvolvimento e onde esta análise vai certamente reflectir grandes resultados em termos de tempo, esforços e dinheiro;

- **novo teste para confirmar –** nova análise dos casos de teste com os requisitos do negócio é tão importante como a nova verificação de casos de teste construída;

- **começar os testes assim que uma unidade esteja pronta –** no final de cada módulo, deve ser feito o teste de unidade e teste de integração;

- **construir um repositório de casos de teste –** construir um repositório de casos de teste permite economia de tempo, para cenários de requisitos semelhantes;

- **alteração dos casos de teste com a mudança de requisitos –** para efeitos de requisitos, deve-se alterar os casos de teste após a compreensão das mudanças exigidas (documentado e assinado);

- **confirmar a cobertura completa dos requisitos em casos de teste –** assegurar que as coberturas dos requisitos são completas e adequadas com a necessidade do negócio em relação ao produto construído, podendo resultar em desastre, caso a cobertura não seja completa;

- **o conhecimento do negócio é tão importante quanto o conhecimento de testes –** a regra dos 50:50 é aplicável para ambos os casos, ou seja, 50% dos conhecimentos deve ser técnicos e os outros 50% de conhecimentos de negócio;

- **começar a construir os casos de teste conjuntamente com o desenvolvimento –** ao mesmo tempo que o gestor do projecto efectua a gestão das tarefas da equipa de desenvolvimento, a equipa de testes deve construir os casos de teste com base nos requisitos de *software*;

- **requisito reservado não significa não haver mudanças –** não se deve fechar as portas para mudanças nos requisitos de negócio após a fase inicial de estudo, pois as regras do negócio e as empresas mudam;

- **documentação com requisitos claros e completos -** os requisitos de teste são baseados em requisitos estabelecidos pelo cliente ou patrocinador do *software* e por isso é importante a documentação dos requisitos ao nível dos utilizadores e ao nível da gestão, pois qualquer falta ou requisito menos claro pode causar problemas na execução e implementação do projecto.

3.9. Conclusão

Neste capítulo apresenta-se a importância do processo da engenharia de requisitos, com o objectivo da melhoria da qualidade do processo de desenvolvimento. Nenhuma das propostas se ajusta na perfeição no projecto e por isso não há obrigação de seguir rigorosamente os modelos apresentados.

Neste capítulo foram igualmente descritas as principais características da engenharia dos requisitos de *software*, onde foi apresentado uma visão geral da engenharia dos requisitos, as principais actividades da engenharia dos requisitos, linguagens de especificação, a gestão dos requisitos, requisitos funcionais e não funcionais e no final apresentaram-se as regras de ouro da engenharia de requisitos do *software*.

Capítulo 4.

Testes de *software*

4.1. Introdução

Os testes de qualidade são de grande importância para a qualidade de *software*, pois têm como propósito a garantia da qualidade de *software* e a validação de que as funcionalidades estão de acordo com os requisitos. Têm como principal objectivo o de identificar problemas no funcionamento do *software*, fornecer diagnósticos para que os erros sejam corrigidos e manter o foco na prevenção do erro durante o ciclo de vida do *software*.

Depois de codificar o programa, o passo seguinte é testá-lo, existindo para tal vários tipos de abordagens. Quando se procede à acção do teste, devem–se ter projectado casos de testes, com o intuito de aplicar na fase de testes de *software* e fazer a respectiva correcção dos defeitos existentes. Nesta fase, os testes do *software* também devem ter em conta os requisitos funcionais e técnicos, pois são estes que especificam e caracterizam o produto que está a ser construído.

Como esta actividade consome bastante tempo e recursos, esta fase é muitas vezes negligenciada em função de limitações de prazo e orçamento e por isso o *software* é entregue ao cliente com defeitos não revelados que provocarão falhas no quotidiano. Por esta razão, a eficácia dos testes é um dos factos importante para garantir a qualidade e para a redução de custos.

Neste capítulo apresentar-se-ão os principais pontos na fase de teste para ajudar a perceber melhor a área de testes de *software*, como por exemplo, os objectivos e princípios dos testes, os critérios de cobertura de testes, a abordagem estratégica aos testes, os tipos de teste mais conhecidos, a depuração de programas, os defeitos e falhas na aplicação, o planeamento e as ferramentas automáticas de teste.

4.2. Principais objectivos e princípios de teste

Neste capítulo é apontado os principais objectivos e princípios por que se devem reger os testes de *software*, fornecendo indicações de fiabilidade e qualidade de todo o *software*.

Em relação aos principais objectivos dos testes de *software* são de referir os seguintes pontos [3]:

- têm como finalidade encontrar erros;
- os bons casos de teste são aqueles que têm alta probabilidade de encontrar erros ainda não encontrados;
- testes bem sucedidos são aqueles que encontrarão erros ainda não descobertos.

Entre os principais princípios que devem reger os testes de *software* temos os seguintes pontos [3]:

- devem estar relacionados com o cliente e com as suas expectativas quanto ao produto;
- os testes devem ser planeados com a devida antecedência, devendo faze-los assim que o modelo de requisitos esteja completo;
- os testes devem considerar a aplicação do princípio de Pareto [3], que sugere que 80% dos erros estarão relacionados com 20% dos componentes;
- os primeiros testes concentram-se nos componentes e depois deslocam-se para os testes de sistema;
- o teste completo é impossível.

4.3. Critério de cobertura de teste

O critério de cobertura de teste é uma regra pela qual como se deve seleccionar os testes e quando se deve parar os mesmos testes. Trata-se de condições que devem ser preenchidas pelo teste, a qual seleccionam determinados caminhos que visam cobrir o código implementado.

Segundo Gustafman [13] os principais pontos de coberturas são os seguintes: teste funcional, teste estrutural e matrizes de testes.

Teste funcional

Neste tipo de teste é utilizada a especificação do *software* para identificar subdomínios que devem ser testados. Um dos primeiros passos é criar casos de testes para cada tipo de saída da aplicação (como exemplo, cada mensagem de erro deveria ser gerada), seguindo todos os casos especiais que devem ter casos de teste. Todas as situações de excepção, erros e enganos devem ser testados. No final temos como resultado um conjunto de casos de teste que irá testar totalmente a aplicação quando implementado. Esse conjunto de casos de teste também tem como objectivo ajudar a identificar alguns comportamentos esperados do *software*.

Teste estrutural

Este tipo de teste é baseado no fluxo de dados do programa, onde podemos encontrar os seguintes critérios de teste:

- **c0 – cobertura de todos os comandos –** todos os comandos do código-fonte devem ser executados por algum caso de teste;
- **c1 – teste de todos os ramos –** este tipo de teste tem como objectivo passar por todos os caminhos e em todas as decisões;
- **teste de todo o caminho –** este tipo de teste tem como finalidade verificar se uma determinada sequência de nós de programas é executado por um caso de teste, podendo ter um número infinito de caminhos e por isso não é atingido o final do caminho;
- **cobertura de condição múltipla –** para este tipo de critério é requerido que cada condição de primitiva seja avaliada como verdadeira e falsa;
- **teste de subdomínio –** a ideia principal para este tipo de teste é o de dividir o domínio de entrada em subdomínios, não restringido como estes subdomínios são seleccionados.

Matrizes de teste

Uma maneira de formalizar a identificação de subdomínios é através de matrizes de teste, identificando assim todas as combinações possíveis como sendo verdadeiras ou falsas.

Um exemplo da matriz de teste é a tabela seguinte com as condições do problema de um triângulo: são colocadas 4 condições nas linhas da matriz. As colunas da matriz representam um subdomínio, sendo colocado um V quando a condição é verdadeira e um F quando a condição é falsa. Podem ser usadas linhas adicionais para as condições e as respectivas saídas esperadas para cada subdomínio.

Casos	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
a =b ou a=c ou b=c	V	V	V	V	V	F	F	F
a=b=c	V	V	F	F	F	F	F	F
a<b+c ou b<a+c ou c < a+b	V	F	V	V	F	V	V	F
a,b,c>0	V	F	V	F	F	V	F	F
Exemplo de Caso de teste	0,0,0	3,3,3	0,4,0	3,8,3	5,8,5	0,5,6	3,4,8	3,4,5
Saída esperada	inválido	equilátero	inválido	não é triângulo	isósceles	inválido	não é triângulo	escaleno

Tabela 3 – Matriz de teste

(adaptado de [13]).

4.4. Abordagem estratégica ao teste de *software*

Existem várias estratégias de teste de *software*. O principal objectivo da estratégia de testes de *software* é o de fornecer as directrizes para a verificação, validação, organização e estratégia dos testes de *software*, tendo os seguintes pontos principais [3]:

Verificação e validação

Os testes têm como objectivo garantir a qualidade de *software*, permitindo a verificação e validação:

- verificação tem como objectivo garantir que o programa está a ser implementado correctamente ("Estamos a construir bem o produto?");
- validação tem como objectivo garantir que o programa corresponde aos requisitos ("Estamos a construir o produto correcto?").

Em seguida serão apresentados os erros mais comuns, que dificultam o processo de desenvolvimento [14]:

- incapacidade de compreensão das necessidades do utilizador;
- engano na gestão do tamanho do projecto;
- falta de planeamento de projecto;
- falta de testes do produto final;
- execução errada do produto;
- forma inadequada de lidar com as coisas;
- não permitir que um profissional possa planear o projecto de desenvolvimento de *software*;

- não incentivar a equipa a dar o melhor;
- atraso do plano do projecto;
- instalação de um pacote de *software* insuficiente;
- mudança de ferramentas de *software* no início do projecto;
- permitir adicionar coisas desnecessárias;
- por vezes os programadores tentam eliminar tarefas necessárias a fim de encurtar o plano do projecto;
- incapacidade de gestão do projecto;
- falta de patrocínio do mais alto nível empresarial;
- não aumento da equipa de projecto para acelerar o projecto;
- falta de teste de unidade;
- esgotamento dos programadores de *software*;
- falta de tratamento de erros;
- um erro leva a outro e por isso deve-se verificar cuidadosamente os erros;
- erros por parte de um processo de desenvolvimento de *software*;
- implementação incorrecta de *hardware*;
- inexistência de convenções de códigos;
- aplicar sempre variáveis globais;
- falta de coordenação do processo de desenvolvimento de *software*;

- incapacidade de comentar o código;
- manter as informações para si próprio;
- executar operações de base de dados na camada de aplicação, em vez na camada de base de dados;
- invalidar os dados;
- falta de teste de carga.

Na tabela seguinte apresentam-se alguns exemplos de erros que aconteceram em empresas [15]:

Ano	**Organização** (quando não indicado, a organização está sedeada nos EUA)	**Resultados**
2004	Avis Europe PLC (RU)	Sistema de ERP (*Enterprise Resource Planning*) cancelado, depois de gastos 54.5 milhões de dólares
2004	Ford Motor Co.	Sistema de compras abandonado depois do desenvolvimento ter custado aproximadamente 400 milhões de dólares
2004	Hewlett-Packard Co.	Problemas com o sistema de ERP contribuíram para a perda 160 milhões de dólares
2001	Nike Inc.	Problemas com o sistema de gestão da cadeia de abastecimento contribuíram para a perda de 100 milhões de euros

Tabela 4 - Resultados de erros de empresas

(adaptado de [15]).

Organização do teste de *software*

Deverá haver um trabalho contínuo no projecto em conjunto entre os vários grupos de trabalho (programadores, grupo de teste, etc.), com o objectivo de garantir a qualidade de *software*.

O programador deverá ser responsável por efectuar os testes de unidade dos módulos do programa, garantido que cada um dos módulos está a funcionar de acordo para o fim qual foi concebido. O grupo independente de teste deverá remover os problemas associados à aplicação. O engenheiro de *software* deverá trabalhar com os programadores e o grupo de teste, para garantir que estam a ser efectuados testes rigorosos.

Estratégia de teste de *software*

Devem testar-se individualmente os programas que integram a aplicação, com o objectivo de verificar que cada um está a funcionar adequadamente. Em seguida deve-se reunir os diversos componentes para formar o pacote de *software*, permitindo fazer os respectivos testes de integração e validar se os requisitos estão a funcionar adequadamente. Por fim deve ser testado com outros elementos do sistema e verificar se o desempenho global é conseguido.

Critérios para completar o teste

Não existe limite para o término dos testes. Ao longo do tempo, existe a probabilidade de decréscimo de erros à medida que o teste progride, não havendo nunca a certeza de inexistência de mais erros. Um exemplo desta situação é o Microsoft Windows 7, onde participaram no desenvolvimento três mil engenheiros, cinquenta mil parceiros e oito milhões de testadores [16].

4.5. Teste caixa-branca

O teste de caixa-branca [3] é um método que utiliza a estrutura de controlo do projecto procedimental para derivar casos de testes. Esta técnica utiliza directamente o código-fonte do

componente para avaliar diversos aspectos, tendo como principais objectivos os seguintes pontos [3]:

- garantir que todos os fluxos de execução corram pelo menos uma vez;
- garantir que todas as decisões lógicas corram os lados verdadeiros e falsos;
- garantir que os ciclos corram nos seus limites e intervalos operacionais;
- garantir que as corram as estruturas de dados internas com objectivo de garantir sua validade.

Para além dos objectivos anteriormente referidos, também se deve ter em conta o tempo e a energia gastos com este tipo de teste, devido às seguintes razões [3]:

- erros lógicos e pressupostos incorrectos encontrados no fluxo do programa;
- o fluxo de controlo de um programa pode levar a cometer erros de projecto;
- erros de digitação ao traduzir um programa num código-fonte.

4.6. Teste caixa-preta

O teste de caixa-preta [3] tem como objectivo focar-se nos requisitos de *software*, funcionando como complemento ao teste de caixa-branca. A principal consideração que se deve ter neste tipo de teste é a do domínio da informação. As principais categorias em que este tipo de teste se foca são as seguintes [3]:

- funções incorrectas;
- erros de interface;
- erros de estrutura de dados e acesso a base de dados;
- erros de comportamento ou desempenho;

- erros de inicialização e finalização.

Existem várias técnicas que podem ser usadas [3]:

- **métodos baseados em grafos** – entender os objectos de dados modelados e suas relações, definindo–se uma série de testes que verifiquem relações esperadas entre eles e por final cria-se uma colecção de nós que representam os objectos;

- **método da partição equivalente** – o domínio de entrada de um programa é dividido em classes de dados, em que os casos de teste podem ser derivados, descobrindo classes de erro e reduzindo o número total de casos de teste;

- **método da análise de valores limites** – alguns erros ocorrem na fronteira do domínio de entrada e este tipo de técnica é usada para a análise de valores de limite;

- **testes de comparação** – é utilizada para minimizar a possibilidade de erro usadas em sistemas redundantes;

- **testes de matriz ortogonal** – é normalmente utilizada quando o domínio de entrada é pequeno, mas que exigem muitas provas exaustivas e onde é bastante útil para encontrar erros de lógica associados a erros de lógica num componente de *software*. Também possibilita obter boa cobertura de teste com menos casos de testes que a estratégia exaustiva.

4.7. Teste caixa-cinzenta

Enquanto o teste de caixa-preta se concentra na funcionalidade do programa e sua especificação e o teste de caixa-branca se concentra na estrutura da lógica, o teste de caixa-cinzento é a combinação dos dois testes anteriores [17]. O responsável pelos testes de *software* estuda as especificações dos requisitos e comunica com o programador para entender a estrutura do sistema. O principal objectivo é acabar com especificações ambíguas e construir testes apropriados. Um exemplo deste tipo de teste é quando a pessoa que está a testar uma certa funcionalidade se apercebe que uma parte dessa funcionalidade está a ser reutilizada em toda a aplicação, entrando em contacto com o programador com o objectivo de compreender a

arquitectura interna e assim eliminar muitos testes, pois poderá testar a funcionalidade apenas uma única vez.

4.8. Teste de unidade

Também conhecido por teste unitário ou teste de módulo, o principal objectivo é o de testar em pequenas partes ou unidades do *software* desenvolvido (componentes / módulos) e assim encontrar falhas de funcionamento isoladas do todo o sistema [18].

4.9. Teste de integração

Nesta fase o principal objectivo é a integração dos vários módulos / componentes do sistema, que são combinados e testados em grupo. Por vezes, quando combinados, podem não efectuar a função principal para a qual se desejaria, sendo por isso necessário fazer a construção dos vários programas independentes. Em seguida serão discutidas algumas abordagens dos testes de integração [3]:

- **teste *bottom-up*** - começa a construção de testes com os componentes individualmente, ou seja, nos níveis mais baixos da estrutura do programa, e são integrados de baixo para cima;

- **teste *top-down*** - é uma abordagem incremental para a construção da estrutura do programa. Os vários módulos são integrados, começando com o módulo principal, movendo-se descendentemente pela hierarquia de controlo. Poderá haver problemas de processamento nos níveis mais baixos da hierarquia quando é necessário testar adequadamente os níveis superiores.

- **teste de regressão** - cada vez que um módulo é adicionado ao programa, o *software* modifica-se. Estas modificações podem trazer problemas a funções que anteriormente funcionavam perfeitamente. Neste contexto, o teste de regressão é a reexecução de parte de testes, para garantir que essas modificações não se propagam com efeitos indesejáveis;

- **teste cortina de fumo** - este tipo de teste é usado frequentemente quando os produtos de *software* estão a ser desenvolvidos. São testes que procuram erros ao sistema todo, de ponta a ponta. Embora não exaustivo, deverá ser suficientemente rigoroso para garantir a estabilidade do sistema.

- **documentação do teste de integração** - o documento de testes de integração deverá conter um plano de testes e procedimentos de teste. O plano de testes deverá descrever a estratégia global de integração, devendo os testes serem divididos por fases e construções que tratam as características funcionais e comportamentais específicos do *software*. Deverão seguir os seguintes critérios aplicados à fase de teste:

    - **integridade de interface** – são testados os interfaces internos e externos à medida que cada módulo é integrado na estrutura;

    - **validade funcional** – o objectivo é descobrir erros funcionais, isto é, verificar se o produto corresponde aos requisitos dos utilizadores;

    - **conteúdo informacional** – serve para descobrir erros relacionados com estruturas de dados locais ou globais;

    - **desempenho** – testes associados à verificação do desempenho estabelecidos pelo *software*.

4.10. Teste de validação

Segundo Pressman [3], no final dos testes de integração e depois de completar o pacote final de *software* e corrigidos os erros que foram descobertos, podem começar os testes de validação.

A validação do *software* torna-se bem sucedida quando o próprio *software* funciona segundo as expectativas do cliente. Esta é conseguida através de um conjunto de testes que demonstram conformidade com os requisitos, através de um plano de testes que confirma que todos os requisitos estão completos. O plano e os procedimentos dos testes são projectados com o

objectivo de garantir que os requisitos funcionais são satisfeitos. Caso sejam descobertos desvios ao planeado, será necessário estabelecer métodos de resolução das deficiências existentes, sendo preciso negociar com o cliente um método para corrigir as deficiências encontradas.

Outro elemento importante do processo de validação é a revisão da configuração, que têm como objectivo garantir que todos os elementos da configuração foram adequadamente desenvolvidos, catalogados e com os detalhes necessários para apoiar a fase de suporte do ciclo de vida do *software*.

4.11. Teste de sistema

O teste de sistema é uma fase do processo que integra testes cuja finalidade principal é a preocupação com os aspectos gerais do sistema. Não se limita a testar os requisitos funcionais, mas também os não funcionais. Em seguida, mostram–se alguns testes de sistemas baseados em *software* [3].

4.11.1. Teste de recuperação

Muitos sistemas devem recuperar de falhas e retomar o processamento em tempo útil, sendo em alguns casos tolerante a falhas e em outros casos deverão ser corrigidos em tempo específico. Este tipo de teste têm como objectivo forçar o *software* a falhar em diversos modos, com o objectivo de verificar se a recuperação é adequadamente realizada.

4.11.2. Teste de segurança

O teste de segurança tem como objectivo verificar se os mecanismos de protecção incorporados no sistema estão de facto a proteger de invasão imprópria, quanto às vulnerabilidades do sistema.

4.11.3. Teste de carga

No teste de carga, procura–se testar o sistema quando sujeito em quantidade, frequência ou volume de dados anormais. Como o principal objectivo é verificar o limite de dados processados pela aplicação até não conseguir processar, permite a melhoria do *software* ao reduzir custos, optimização de carga e eventuais estrangulamentos.

4.11.4. Teste de desempenho

Tem como objectivo testar o desempenho de *software* durante a execução, ocorrendo ao longo do contexto do sistema integrado. Por vezes são utilizados instrumentos para monitorizar intervalos de execução, para descobrir situações que possam levar à degradação do desempenho do *software* e falha do sistema.

4.12. Testes de ambiente, arquitecturas e aplicações especializadas

À medida que os programas se tornam mais complexos, são usadas abordagens de testes mais especializadas. Nesta secção apresentam–se alguns desses testes [3].

4.12.1. Testes de interfaces gráficas

É através da interface gráfica que a maioria dos programas comunica com os utilizadores, cujo GUI (*Graphical User Interface*) representa uma grande parcela de código e consequentemente maior probabilidade de defeitos de aplicação. Ao longo do tempo a criação de interfaces para o utilizador tornou–se mais precisa e menos demorada, devido ao uso de componentes reutilizáveis, mas ao mesmo tempo aumentou a complexidade dos casos de teste.

Devido à permutação de operações GUI, existem várias ferramentas automáticas de testes que ajudam a realizar esses mesmos testes, em oposição à abordagem manual, que está sujeita a falhas humanas, dura bastante mais tempo e é mais dispendiosa [3].

4.12.2. Testes de arquitecturas cliente / servidor

Devido à natureza distribuída desta arquitectura, torna–se mais difícil fazer testes do que em relação aos aplicações isoladas. Existe um aumento de custo e de tempo para o desenvolvimento deste tipo de arquitectura, devido à natureza distribuída do ambiente cliente / servidor, aspectos de desempenho associado ao processamento de transacções, a diferentes tipos de plataformas de *hardware*, complexidades de redes de comunicação, etc.

4.12.3. Testes de documentação

O teste de documentação deve ser abordado de duas maneiras:

- verificação da clareza do documento;
- utilização da documentação em conjunto com o programa de *software* com o objectivo de fazer testes de coerência.

Sugere-se haver um utilizador que deverá validar a documentação. Todas as discrepâncias devem ser anotadas e quando existe ambiguidade ou deficiência no documento, este erro deve ser corrigido.

4.12.4. Testes de sistemas em tempo real

Este tipo de testes têm o objectivo avaliar o correcto desempenho a um estímulo e sua resposta em tempo útil, podendo ter os seguintes casos de teste [3]:

- **teste de tarefa –** neste tipo de teste verifica-se cada tarefa individualmente, onde o objectivo é descobrir erros de lógica e de função;
- **teste comportamental –** o principal objectivo é examinar o comportamento do sistema e verificar a sua conformidade com o esperado;
- **teste intertarefas –** depois das tarefas serem testadas individualmente e examinado o comportamento do sistema, as tarefas são avaliadas em conjunto com a finalidade de descobrir erros;
- **teste de sistema –** o *software* e o *hardware* são integrados com o objectivo de verificar a eficiência do sistema.

4.13. Testes Alfa e Beta

Por norma as aplicações passam por duas fases de testes, antes de serem consideradas como produto final [19]:

- **teste Alfa –** testes efectuados pelos utilizadores da própria empresa;

- **testes Beta –** testes efectuados que envolve o uso limitado de utilizadores externos à empresa que desenvolve a aplicação, com o objectivo de encontrar erros e serem corrigidos antes na versão final.

4.14. Depuração de *software*

A depuração de *software*, normalmente conhecido por *debugging*, não é propriamente um teste e ocorre como consequência dos testes efectuados. Quando o resultado da avaliação de um teste não tem correspondência entre o esperado e aquilo que efectivamente é obtida, é efectuado o *debug* ao programa com o objectivo de encontrar a causa e levar à correcção do erro de *software*.

Em geral temos três categorias de abordagens de *debug* [3]:

- **força bruta** – é o método mais comum e o menos eficiente; normalmente utilizado quando as outras abordagens falham; o programa é carregado com mensagens tendo como objectivo encontrar uma pista, que nos leve a encontrar o erro;

- **comportamento** – método utilizado quando se têm programas pequenos; o código-fonte é verificado até encontrar a causa do erro;

- **eliminação da causa** – os dados relacionados com a ocorrência do erro é organizada para isolar as causas, refinadas numa tentativa de isolar o erro.

4.15. Defeitos e falhas no *software*

Devido a um grande número de situações, a presença de defeitos é função não só do *software*, mas também das expectativas dos utilizadores e clientes. O *software* falha quando a expectativa que esperamos do programa não está de acordo com os requisitos, resultado de vários factores, como por exemplo [20]:

- falta ou falha de requisito na especificação, onde a especificação pode não ter sido manifestada de modo claro pelo cliente;

- requisito da especificação difícil ou impossível de implementar, considerando o *software* e o *hardware* (violação da regra de ouro);

- defeitos no projecto do sistema, onde o algoritmo implementado pode não corresponder ao esperado ou está incompleto, quer em termos de funcionalidade, quer em termos de desempenho.

Quando é concluída a fase de codificação e se entra na fase de teste, espera-se que as especificações reflectidas no programa tenham uma implementação correcta. Através da fase de testes consegue-se identificar o defeito e efectuar posteriormente a correcção do defeito.

4.16. Planeamento de testes

O planeamento caracteriza-se pela definição de proposta de testes baseadas nas especificações do cliente, considerando prazos, custos e qualidade esperados, ajudado a projectar e a organizar os testes. Cada etapa do processo deve ser planeada, devendo seguir os seguintes pontos [20]:

- estabelecer os objectivos do teste;

- definir os casos de teste;

- escrever casos de teste;

- testar os casos de testes;

- executar os casos de testes;

- avaliar os resultados dos testes.

Normalmente usa-se um documento, chamado plano de teste, que consiste numa modelação detalhada do fluxo de trabalho durante o processo. Este plano de testes é um dos oito

documentos pertencentes à norma IEEE 829, onde estão descritas as linhas orientadoras para a estrutura do plano.

4.17. Ferramentas automáticas de testes

A execução manual dos testes de *software* pode ser rápida e efectiva, embora seja uma tarefa dispendiosa e cansativa [21]. Por isso é normal que ao longo do tempo as condições dos testes não se verifiquem novamente e que os erros de *software* aconteçam, trazendo prejuízo para as equipas de desenvolvimento que perdem tempo a identificar e a corrigir os erros e também para os clientes que sofrem com os atrasos nos prazos e tempo com a entrega de *software* de qualidade duvidosa.

Os testes automáticos são programas ou *scripts* simples, cujo principal objectivo é avaliar as funcionalidades do sistema, fazendo essas verificações de uma forma automática [21]. A principal vantagem deste tipo de abordagem é que todos os casos de teste podem ser rápida e facilmente repetidos a qualquer momento e com pouco esforço. Existem várias ferramentas automatizadas que nos podem ajudar nos testes de *software*, sendo os principais pontos indicados seguidamente [20]:

Ferramentas de análise de código

Existem duas categorias de ferramentas de análise de código: a **análise estática** (que é utilizada quando o programa não está a ser executado) e a **análise dinâmica** (possibilita relatar o comportamento do programa em tempo real);

- **análise estática** – existem diversas ferramentas que analisam o código-fonte do programa antes que seja executado, onde se investiga a correcção da aplicação e que inclui as seguintes tarefas:
    - **analisador de código**: avalia a sintaxe dos componentes;
    - **verificadores de estrutura**: avalia a fluxo lógico, verificando problemas estruturais;

- **analisador de dados**: avalia a estrutura de dados;
- **verificador de sequência**: avalia a sequência de eventos.

- **análise dinâmica** – muitas vezes os sistemas são difíceis de testar, pois diversas operações são executadas em paralelo, especialmente nos sistemas de tempo-real. É aqui que os testes com ferramentas automáticas possibilitam que a equipa de desenvolvimento obtenha os estados dos eventos durante a execução da aplicação, através de monitorização de programas que permitem avaliar o desempenho do sistema.

Ferramentas de execução de testes

Este tipo de testes não tem foco no código, mas sim na execução dos testes de *software*, devido ao tamanho e à complexidade dos sistemas actuais. Nos últimos anos, este tipo de ferramentas têm ajudado os programadores a aumentar a produtividade no desenvolvimento de *software*, aumentando também a execução de testes na validação do produto em menor tempo. Assim temos as seguintes ferramentas que podem ajudar na execução dos testes:

- **captura e repetição –** quando os testes são planeados, a equipa deve especificar nos casos de teste, as entradas fornecidas e qual o resultado esperado das acções testadas e é aqui que as ferramentas de captura e repetição são úteis, pois assim são capturadas as entradas e respostas dos testes que estão a ser executados e descobrir discrepâncias e em seguida corrigir os defeitos que foram encontrados;

- ***stubs* e *drivers* –** este tipo de testes têm como objectivo testar um componente específico com um caso de teste, existindo ferramentas comerciais com o objectivo de auxiliar na geração de *stubs* e *drivers*, podendo o *driver*:

  - definir todas as variáveis de estado à preparação de casos de teste;
  - simular a inserção de dados e outras condições relacionadas com dados;
  - comparar os resultados com a saída esperada;

- rastrear os caminhos durante a execução;
- redefinir as variáveis para novo caso de teste;
- interagir com pacotes de depuração, com o objectivo de rastrear e corrigir os erros.

- **ambientes automáticos de teste –** a automação dos testes deve ser integrado com outras, afim de formar um amplo ambiente de testes e têm como objectivo o auxilio nos testes do *software* (como por exemplo ferramentas conectadas a base de dados de teste, ferramentas de medição, ferramentas de análise de código-fonte, ferramentas de simulação e modelação, etc.), pois os testes sempre envolverão esforço manual para rastrear o defeito até a causa original e por isso a automação auxilia os testes, não substituindo a função humana.

Geradores de casos de teste

Este tipo de ferramenta baseia–se na estrutura do código fonte, fluxo de dados, teste funcional ou estados de cada variável com o objectivo de criar casos de testes. Como os testes de *software* resultam da definição cuidadosa e completa dos casos de testes, é bastante vantajoso automatizar parte dos casos de teste, com o objectivo de garantir que os testes cubram todos os casos possíveis.

4.18. Conclusão

Os testes de *software* são um elemento útil e que têm como objectivo o aumento da confiabilidade e garantia dos requisitos do produto. Podem ajudar o processo de desenvolvimento e consequentemente a qualidade da aplicação, dentro dos custos e prazos.

A implementação de métodos de teste, ajuda a minimizar desperdícios e estimular a equipa de desenvolvimento a melhorar o seu procedimento de trabalho, através da incorporação de procedimentos, filosofias e documentação que incorporam factores de qualidade do *software*.

Neste capítulo foram descritas as principais características dos testes de *software*, começando com uma introdução aos testes de *software*, quais os principais objectivos dos testes, o critério de cobertura de teste, a abordagem estratégica aos testes de *software*, os principais testes de *software*, a depuração *software*, defeitos e falhas no *software*, planeamento de testes e as ferramentas automática de testes.

Capítulo 5.

Qualidade de *software*

5.1. Introdução

A qualidade de *software* é uma área de conhecimento da Engenharia de *Software* que tem como objectivo garantir a qualidade do *software* através da definição e normalização de processos de desenvolvimento. A qualidade hoje em dia é um pré-requisito que as empresas devem conquistar para conseguir colocar os seus produtos no Mercado Global.

A garantia do controlo da qualidade está relacionada com as actividades de verificação e validação. Em algumas organizações não existe diferença entre estas actividades, embora devam ser distintas, sendo também designadas por actividades de controlo da qualidade.

Neste capítulo serão apresentados a visão histórica da qualidade, os principais conceitos de qualidade, as principais garantias e custos, as normas de qualidade, as métricas e o planeamento da qualidade do *software*.

5.2. Visão histórica

A história do desenvolvimento da qualidade [22] como sistema têm origem do modelo científico de administração de F. Taylor em 1911, publicado em seu livro "Princípios da Administração Cientifica", onde citava: o aumento da eficiência, a racionalização dos métodos de trabalho, a crença no homem económico, a divisão e a hierarquização do trabalho, a relevância da organização formal.

Durante os anos 30, Shewhart [23] causou uma revolução na teoria científica da administração, ao propor um método voltado para gestão das organizações, conhecido como "Controlo da Qualidade" que se baseava na aplicação de gráficos de controlo e na inspecção por amostragem.

Deming [23] defendia nos Estados Unidos o modelo da produção com qualidade e como não lhe deram muita atenção, o Japão convidou-o para fazer uma série de palestras. Ensinou o

seu método e aperfeiçoou-o, desenvolvendo uma nova forma de gestão, tirando proveito, não só dos conhecimentos e habilidades dos funcionários, como das sugestões dos clientes.

Para além de Deming, os Estados Unidos têm outros grandes idealistas do processo de qualidade: J.M. Juran, Philip Crosby, são famosos por serem conhecidos como pensadores da qualidade e que começaram a efectuar consultoria nos Estados Unidos sobre qualidade total ou liderança pela qualidade.

5.3. Principais conceitos de qualidade de *software*

Nesta secção apresentam-se os principais conceitos sobre qualidade de *software*, preocupações e atributos a ter em conta durante o processo de desenvolvimento do produto.

Existem várias maneiras de definir qualidade de *software*, não sendo nenhuma perfeita, pois é um conceito relativo e dinâmico. Algumas das definições de qualidade são [22]:

- a qualidade deve cumprir com as exigências dos clientes;
- a qualidade deve andiantar e agradar as vontades dos clientes;
- a qualidade deve escrever que se faz e fazer o que é escrito.

A qualidade é indispensável para a sobrevivência das empresas no mercado de *software* e que se está a efectuar de uma forma global. Uma organização só sobressairá no mercado global quando produzir *software* e serviços de boa qualidade, reconhecido como tal pelos seus clientes.

Existem muitas motivações que devem ser levadas em conta [22]:

- **a qualidade é competitividade** – uma das maneiras de diferenciar o produto de outras empresas competidoras, é através da qualidade do *software* e do suporte fornecido aos seus clientes. Os utilizadores querem que as empresas demonstrem efectivamente a sua qualidade;

- **a qualidade é indispensável para a sobrevivência** – os clientes estão a exigir qualidade do *software*. Se a empresa quiser sobreviver num mercado altamente competitivo, tem que alinhar-se com o mercado. As empresas estão a reduzir o número de fornecedores e a escolher as que têm certificações de qualidade;

- **a qualidade é indispensável para o mercado global** – o mercado de *software* está, cada vez mais, tornar-se global. As empresas que mostrarem qualidade, eventualmente colocam–se em situação privilegiada no mercado global;

- **a qualidade melhora a relação custo / benefício** – um sistema de qualidade permite o aumento da produtividade e diminuição de custos. As empresas sabem que efectuar a correcção de defeitos posteriormente ao desenvolvimento do *software* é mais dispendioso do que corrigi-los ao longo do processo de desenvolvimento;

- **a qualidade fideliza clientes e aumenta os lucros** – a pouca qualidade normalmente custa muito tempo e recurso. Os clientes não toleraram a falta de qualidade e por isso quanto maior a qualidade, maior será a satisfação dos clientes.

Existe um conjunto de atributos que qualquer produto com qualidade deve ter, independentemente de estarem ordenados por prioridade, sendo bastante a sua definição no início do desenvolvimento [24]:

- **funcionalidade** – conjunto de atributos que o produto deve atender com o objectivo de satisfazer os requisitos dos utilizadores, onde os principais funcionalidades são: adequação, exactidão ou rigor, interoperabilidade, concordância e segurança;

- **fiabilidade** – conjunto de atributos que expressa a capacidade do produto em manter o seu nível de desempenho ao longo do tempo, onde os principais atributos são: maturidade, tolerância às falhas e facilidade de recuperação ou recuperabilidade;

- **facilidade de utilização** – conjunto de atributos que relaciona o esforço do utilizador com a utilização do produto, onde os principais atributos são: facilidade de compreensão, facilidade de aprendizagem e facilidade de operação;

- **eficiência** – conjunto de atributos que relaciona o nível de desempenho e a quantidade de recursos utilizados, onde os principais atributos são: comportamento face ao tempo e face aos recursos;

- **facilidade de manutenção –** conjunto de atributos que traduz no esforço necessário para alteração do produto de *software*, onde os principais atributos são: facilidade de análise, facilidade de alteração, estabilidade e facilidade de testes;

- **portabilidade –** conjunto de atributos que possibilita a transferência de um produto de *software* para outro ambiente, onde os principais atributos são: facilidade de adaptação, facilidade de instalação, conformidade, facilidade de substituição.

5.4. Garantia de qualidade de *software*

A principal função da garantia de qualidade de *software* (SQA – *Software Quality Assurance*) é fornecer à equipa de gestão a eficácia do processo de qualidade [25] e ajudar a equipa de *software* a conseguir um produto com qualidade [3], através de um conjunto de actividades seguidamente apresentadas [25]:

- preparar o plano de SQA para o projecto de *software*;

- execução de actividades de acordo com o plano de SQA;

- revisões do plano do projecto de *software*, dos procedimentos e dos padrões definidos;

- revisão das actividades de engenharia de *software* para verificar o cumprimento do processo de *software* definido;

- realizar auditoria de produtos de *software* com o objectivo de verificar os desvios ao projecto;

- relatar resultados ao grupo de engenharia de *software*;

- documentar e tratar os desvios identificados nas actividades e nos produtos de *software*, de acordo com um procedimento documentado;

- revisão das actividades do grupo de SQA.

5.5. Fiabilidade do *software*

A fiabilidade de um programa [20] é bastante importante, pois se o programa falha frequentemente no desempenho, deixa uma desconfiança do produto em relação ao utilizador.

A fiabilidade do *software* [26] também pode ser definida como a probabilidade de que o *software* não falhar durante um determinado período de tempo. Este atributo é um factor chave da qualidade de *software*, uma vez que pode quantificar as falhas do *software* em unidade de tempo.

Um exemplo da fiabilidade no *software* é a utilização de *software* médico, através da utilização de um dos seus ramos (Inteligência Artificial), que vêm desenvolvendo programas capazes de elaborar decisões na área do diagnóstico, prognóstico e terapia médica, onde têm capacidade de raciocínio do tipo dedutivo semelhante ao que o médico utiliza para identificar e tratar problemas dos pacientes. A existência deste tipo de aplicações coloca questões difíceis ao nível ético por causa dos erros de *software* (podendo levar ao prejuízo ou à morte do paciente) e por isso é importante a certificação do programa, sendo bastante importante que o *software* ser extensamente testado por parte de equipas de profissionais especializado [27].

5.6. Custo da qualidade de *software*

Os custos da qualidade são todos os investimentos com a finalidade de um programa de *software* atinja a qualidade desejada, ao menor custo possível. Estes custos podem ser divididos nas seguintes áreas [3]:

- **prevenção** – são todos os custos relacionados com a intenção de melhorar e garantir a qualidade, como exemplo: planeamento da qualidade, revisões técnicas, material de testes e formação;

- **avaliação** – são todos os custos relacionados com o objectivo de analisar as falhas ocorridas, com exemplo: correcção de falhas e análise da ocorrência de falhas;

- **falhas** – são todos os custos relacionados da actividade de reparar as falhas originadas no desenvolvimento do produto, como exemplo: solução das queixas, substituição do produto, manutenção do produto e trabalho de garantia.

5.7. Normas de qualidade

Nos últimos anos tem sido desenvolvidos padrões com o objectivo de melhorar a qualidade de *software*, que as empresas passaram a adoptam com o intuito de melhorar a qualidade dos seus produtos. As normas definem um conjunto de critérios que guiam o desenvolvimento dos produtos e serviços e caso não sejam seguidos, poderam resultar na falta de qualidade. Em seguida são apresentadas as principais normas e organismos normativos na área da qualidade de *software*.

5.7.1. International Organization for Standardization

A ISO é um organismo internacional que tem como objectivo a actividade de normalização, também com ênfase na área da qualidade de *software*. O comité que acompanha a área da Engenharia de *Software* é o SC7, com os seguintes grupos de trabalho [28], [29]:

- **WG1 (“*Coding of Still Pictures*”)** – é dedicado aos símbolos, gráficos e diagramas;

- **WG2 ("*Software System Documentation*")** – é dedicado à documentação de sistemas de *software*;

- **WG4 ("*Tools and Environment*")** – é dedicado à avaliação, selecção e adopção de ferramentas CASE;

- **WG6 ("*Evaluation and Metrics*")** – é dedicado à avaliação de produtos e métricas para produtos e processos de desenvolvimento de *software*;

- **WG7 ("*Life Cycle Management*")** – é dedicado à gestão do ciclo de vida do desenvolvimento de *software*;

- **WG8 ("*Support of Life Cycle Processes*")** – é dedicado aos processos de gestão do ciclo de vida do desenvolvimento de *software*;

- **WG9 ("*Classification and Mapping*")** – é dedicado à classificação e mapeamento das normas da Engenharia de *Software*;

- **WG10 ("*Process Assessment*")** – é dedicado ao processo de desenvolvimento, entrega, operação, evolução e serviços de apoio;

- **WG11 ("*Data Definition*")** – é dedicado à definição dos dados utilizados e produzidos por processos de engenharia de *software*.

Em relação às principais normas para a área da qualidade de *software* temos:

- **ISO / *International Electrotechnical Commission* 15504** – avaliação e melhoria dos processos de *software*;

- **ISO / *International Electrotechnical Commission* 12207** – implementação dos processos de *software*;

- **ISO 9000** – está focado na garantia do sistema de qualidade;

- **ISO 9126** – está focado nas características de qualidade do produto.

5.7.2. Institute of Electrical and Electronics Engineers

O IEEE é uma organização internacional criada em 1884 que colabora na teoria e aplicabilidade nas áreas da engenharia electrónica, electrotécnica e de computadores. Em relação às principais normas para a área da qualidade de *software* do IEEE temos as seguintes referências [28]:

- **IEEE Std 730 (1998)**: "*Standard for Software Quality Assurance Plans*";

- **IEEE Std 983 (1986)**: "*Guide for Software Quality Assurance Planning*";
- **IEEE Std 1298 (1992)**: "*Software Quality Management Systems*".

5.7.3. CMMI – Capability Maturity Model Integration

O CMMI (*Capability Maturity Model Integration*) foi desenvolvido pelo SEI (*Sofware Engineering Institute*) e é um modelo de referência que contém práticas para a maturidade para a indústria de *software*, sendo uma evolução do CMM (*Capability Maturity Model*) [30]. O CMMI é constituído por duas representações, com a finalidade de melhoria de acordo com o seu interesse [31]:

- **Representação contínua**

Na representação contínua o principal enfoque são as áreas do processo, onde existem metas práticas de dois tipos: específicas a determinada área e generalistas aplicáveis a todo o processo. Através da avaliação pode-se classificar nos seguintes níveis de capacidade do processo:

- **nível 0: incompleto** – processo parcialmente realizado ou não realizado, onde os objectivos dos processos não foram satisfeitos;
- **nível 1: executado** – processo que satisfaz todos os objectivos específicos e produz trabalho;
- **nível 2: gerido** – processo realizado, planeado e executado de acordo com políticas predefinidas, onde a gestão do processo está relacionada com a realização de objectivos para os processos;
- **nível 3: definido** – processo gerido e ajustado para o conjunto de processos da organização e onde descreve os elementos principais dos processos definidos;

- **nível 4: gerido quantitativamente** – definido e controlado com a ajuda de técnicas quantitativas e estatísticas, onde os objectivos quantitativos da qualidade dos processos são bastante importantes na gestão do processo;

- **nível 5: em optimização** – foca a melhoria contínua de desempenho dos processos, através da inovação incremental e onde se espera a sua contribuição para a melhoria do processo.

- **Representação por estágios**

A representação por estágios tem uma abordagem estruturada e sistemática para a melhoria por estágios, que significa que uma estrutura do processo foi atingida e é base para o estágio seguinte. Está estruturada em cinco níveis que definem o caminho de melhoria, do estado inicial ao estado optimizado.

Nesta representação por níveis, os estados são caracterizadas pelos seguintes atributos: compromisso, habilidade e actividade para a execução. Os estágios da maturidade são:

- **nível 1: inicial** – este é o nível mais baixo de maturidade, onde as organizações têm processos imprevisíveis, não fornecendo um ambiente estável;

- **nível 2: gerido** – neste nível de maturidade, já existe a garantia que os requisitos são geridos, planeados, executados, medidos e controlados. O foco principal é a gestão básica dos projectos (gestão de requisitos, planeamento do projecto, controlo do projecto, avaliação, garantia da qualidade do processo);

- **nível 3: definido –** no nível de maturidade 3, os objectivos genéricos de maturidade dos níveis anteriores foram alcançados e tem como objectivo a padronização do processo (requisitos do desenvolvimento e integração do produto, verificação e validação, foco no processo organizacional, definição

do processo organizacional, aprendizagem organizacional, gestão de projecto integrado, gestão de riscos);

- **nível 4: gerido quantitativamente** – este nível de maturidade os objectivos específicos dos níveis anteriores foram alcançados e o principal foco é a medição, controlo e quantificação do processo (desempenho organizacional do processo, gestão quantitativa do projecto);

- **nível 5: em optimização** – este é o nível mais alto de maturidade, onde os objectivos específicos dos níveis a foram atingidos e tem como objectivo a melhoria contínua dos processos (inovação organizacional e análise das causas e resoluções).

5.8. Métricas de qualidade de *software*

No panorama actual do desenvolvimento de *software*, devido ao aumento da exigência da qualidade do produto, é necessário medir e quantificar a produtividade e assim introduzir melhorias no processo de desenvolvimento do sistema de informação.

O principal objectivo das métricas de qualidade de *software* é direccionar para a melhoria contínua em todos os parâmetros da qualidade por um sistema de medição orientado a metas, através da obtenção de dados quantitativos e assim estabelecer decisões de uma forma racional, que vão de encontro aos objectivos das organizações. Algumas das razões para medir o *software* são indicadas seguidamente [32]:

- a melhoria da compreensão da actividade de gestão;
- controlo mais efectivo da qualidade na fase;
- identificação mais precoce de problemas;
- melhoria da imagem da entidade, aumentando a confiança dos clientes em relação aos fornecedores;

- maior produtividade;
- melhoria das estimativas e planeamento dos projectos;
- melhoria da avaliação da eficácia e eficiência de novas ferramentas utilizadas;
- melhoria da avaliação da novas linguagens de desenvolvimento na produtividade e qualidade dos sistemas desenvolvidos;
- melhoria da motivação dos intervenientes no processo de desenvolvimento do produto;
- maior facilidade dos gestores de projectos provarem a melhoria da eficiência conseguida.

As métricas são indicadores que permitem quantificar o produto do processo de desenvolvimento do *software*, devendo auxiliar no desenvolvimento de modelos de estimar os parâmetros e tem as seguintes características [32]:

- qualidade do que pode ser medido (mensurabilidade);
- repetibilidade;
- baixo custo;
- maior objectividade daquilo para que foi proposta;
- maior robustez.

A análise das métricas ajuda na tomada de decisões, com a intenção de avaliar o desempenho do processo de desenvolvimento do produto.

5.9. Planeamento da qualidade

O plano de qualidade visa que o *software* tenha um planeamento, contendo os pontos mais importantes no desenvolvimento do produto. O principal objectivo é garantir que o produto possa ser avaliado, ter a qualidade que é assegurada pela equipa de programadores.

A elaboração do documento é da responsabilidade do gestor do projecto e pode ter acompanhamento de grupos de suportes, devendo ser submetido à gestão executiva e / ou cliente. O plano deverá ser revisto sempre que houver desvios, de modo a manter–se consistente.

O documento deve incluir os seguintes tópicos [33]:

- objectivos de qualidade;
- gestão e organização da equipa;
- documentação do projecto;
- padrões e guias a serem adoptados ao projecto;
- métricas da qualidade;
- plano de revisão e de autoria.

5.10. Relação entre testes e qualidade de *software*

Os testes de *software* são um factor crítico para garantir a qualidade do produto. É através dos testes que se pode fornecer informação sobre a qualidade do *software*, embora não se possa garantir que o *software* funcione sem a presença de erros.

O teste funciona como uma parcela do processo da qualidade de *software*, que quando utilizada numa metodologia de desenvolvimento, espera-se que o produto final agrade a todos os interessados (como exemplo os fornecedores e clientes).

Assim os testes são uma actividade contínua e permanente, que não deve ser esquecida, no objectivo de garantir a credibilidade do produto.

5.11. Conclusão

A garantia da qualidade é uma actividade importante na engenharia de *software*, que abrange métodos e procedimentos, com o objectivo da confiabilidade do produto e caso a sua aplicação seja conseguida com sucesso, permite o amadurecimento da própria engenharia de *software*.

Neste capítulo foram descritos as principais características da qualidade de *software*, começando por uma introdução pela qualidade, apresentando uma visão histórica, os principais conceitos de qualidade de *software*, a garantia da qualidade, a confiabilidade de *software*, as principais normas de qualidade, as métricas de qualidade de *software*, o planeamento da qualidade e relação entre testes e qualidade de *software*.

Capítulo 6.

Casos de estudo

6.1. Introdução

Tendo em vista a importância da demonstração da gestão dos testes e da qualidade no desenvolvimento dos produtos de *software*, o principal objectivo deste capítulo é apresentar o modo como algumas empresas lidam no dia-a-dia com esta temática e que foi desenvolvido ao longo desta dissertação.

6.2. Gestão da qualidade em companhia de seguros

Esta secção tem como objectivo apresentar a gestão da qualidade de uma companhia de seguros a actuar em Portugal e o modelo dos processos de entrada em testes e produção

Domínio

Os processos apresentados neste caso de estudo estão limitados a entradas aos ambientes de testes e de produção de uma determinada aplicação, não englobando outros processos de desenvolvimento de outros sistemas.

Procedimentos internos

As secções seguintes descrevem os procedimentos internos seguidos. O processo de disponibilização para testes precede a disponibilização para produção, e ambas compreendem vários passos, descritos a seguir.

Disponibilização para testes

- **Reuniões das equipas de sistemas de informação**

Semanalmente são realizadas reuniões, onde é efectuada uma análise do estado de desenvolvimento dos diversos projectos e constrangimentos à colocação em testes. Através desta análise, é criado uma versão do pacote a colocar em testes no dia seguinte.

- **Definição dos pacotes para testes**

As equipas de desenvolvimento decidem quais os objectos que irão ser disponibilizados para testes no dia seguinte. Normalmente as áreas utilizam um portal, com o intuito de disponibilizar os objectos relacionados com o projecto.

O gestor de *releases* deverá em seguida preencher o documento de *releases*, com a identificação dos projectos e áreas intervenientes dos mesmos. O preenchimento da restante informação deverá ser feito pelas respectivas áreas de cada projecto.

- **Disponibilização do software em ambiente de testes**

Com base no documento da *release*, o gestor de *releases* efectua a validação do pacote. Caso existam divergências, estas devem ser corrigidas. No final do dia são efectuadas as autorizações dos pedidos existentes no portal.

- **Testes à qualidade de *software***

Depois da disponibilização das *releases* em ambiente de testes, as equipas de desenvolvimento devem confirmar o sucesso da disponibilização. A equipa de qualidade do *software* iniciará os testes, tendo em conta os planos de testes previamente elaborados. A formalização da finalização da fase de testes deve ser efectuados pela equipa de qualidade de *software* ao ser enviado o relatório de testes ao gestor de *releases*.

- **Ciclo de correcção de falhas reportadas pela qualidade de *software***

Durante o período de testes, a equipa de qualidade de *software* deverá reportar todas as falhas às equipas de desenvolvimento envolvidas, utilizando o relatório de testes. Assim que as falhas estejam corrigidas e validadas, a equipa da qualidade de *software* deverá formalizar a finalização de testes ao gestor do *releases*.

- **Disponibilização da *release* para aceitação dos testes**

As equipas de desenvolvimento devem confirmar o sucesso da disponibilização da *release* no ambiente de testes. Assim que for confirmado, o gestor de *releases* deverá informar a equipa da qualidade de *software* e os gestores de conta que, por sua vez informa a equipa do negócio que os projectos se encontram disponíveis para testes. Deve ser confirmada a data esperada de conclusão dos testes.

A formalização da finalização dos testes deve ser efectuada pela equipa do negócio aos respectivos gestores de conta, onde deve ser acompanhada do relatório de testes e eventualmente um complemento do documento dos testes da equipa de qualidade de *software*.

- **Ciclo de correcção de falhas reportadas pelos testes aceitação**

Aquando dos testes, os testes de aceitação deverão reportar todas as falhas às equipas de desenvolvimento respectivas, utilizando os relatórios de testes. Assim que todas as falhas estejam corrigidas e validadas, deverá formalizar a finalização da fase de testes ao gestor de conta respectivo.

Disponibilização para produção

- **Conclusão dos testes**

Assim que todas as funcionalidades de uma *release* se encontrem testadas e validadas, o gestor de conta deverá coordenar-se com as equipas de desenvolvimento e o gestor de *releases*, com o intuito de disponbilizar toda a informação necessária e enviada para as reuniões de equipa.

- **Reunião das equipas de sistemas de informação**

Durante esta reunião de equipa, é analisado o estado dos projectos actualmente em testes. A análise é efectuada com base nos relatórios de testes e formalizações de finalização de testes do negócio.

Assim decide-se se o pacote mantém, ou se alguns devem ser retirados, pois mesmo testando, o projecto poderá conter falhas não corrigidas ou validadas, não devendo impedir a entrada em produção dos restantes. Estas situações deverão ser as excepções, podendo decidir-se ainda pelo adiamento da entrada em produção do pacote.

Nesta reunião, deve-se discutir os impactos e eventuais planos de recuperação. Deve-se garantir que os requisitos técnicos definidos pelas equipas de desenvolvimento estão assegurados.

- **Definição do pacote para produção**

As equipas de desenvolvimento definem os pacotes que deverão ser disponibilizados para produção, tendo em conta o pacote já definido para testes. Todos os objectos relacionados com os projectos a incluir no pacote, devem ser catalogados no mesmo pedido e ser enviados pelo portal.

O gestor de *releases* será responsável pelo preenchimento do documento de *releases* com a identificação dos projectos e respectivas áreas intervenientes do mesmo. A restante informação será da responsabilidade das áreas assinaladas, para cada projecto.

As equipas de desenvolvimento deverão especificar um plano de acção a activar, caso a entrada em produção não obtenha o resultado esperado.

- **Disponibilização em ambiente de produção**

Baseado no documento de *releases*, o gestor de *releases* deverá efectuar a validação do pacote. Caso exista divergências, deverão ser corrigidas e ser dadas as respectivas autorizações dos pedidos pelo portal.

- **Confirmação e informação ao negócio**

As equipas de desenvolvimento, juntamente com o gestor de *releases*, devem confirmar o sucesso da disponibilização do projecto em ambiente de produção. Em seguida o gestor de

releases e o gestor de conta informam as equipas de negócio que os respectivos projectos se encontram em produção, ou se a operação foi abortada.

6.3. Experiência, metodologias e ferramentas em consultora de sistemas de informação

Esta secção tem como objectivo fornecer uma visão de uma consultora de sistemas de informação a actuar em Portugal na área da qualidade, resultante da experiência adquirida ao longo dos anos. A oferta está assente em três áreas importantes: experiência, metodologias e ferramentas.

Experiência

Ao longo dos anos a consultora efectuou um conjunto de projectos relevantes que permitiu adquirir *know-how* na área da qualidade e testes de *sofware* (como por exemplo em empresas brasileiras). Ainda de referir as certificações que possui nesta área (*ISTQB - International Software Testing Qualifications Board, ISO 9001, NATO e UEO*).

Principais características das metodologias de testes

As principais metodologias usadas vão de encontro aos *best practices*, conceitos dos modelos padronizados e experiência adquirida. Pode-se referir as seguintes metodologias usadas:

- **abordagem interactiva** – baseia-se na refinação, *re-working* dos testes nas interacções e não existindo testes fechados;
- **ferramentas de suporte** – devido às exigências dos sistemas actuais, todo o processo são baseadas em ferramentas de suporte;
- **foco nos testes de regressão** – este tipo de testes têm um aspecto bastante importante em conjunto com as capacidades das ferramentas, sendo fundamentais na garantia da qualidade do *software* produzido;

- **automatização dos testes** – pode-se ter benefícios, através da rapidez de execução dos testes em cada interacção.

Interacção das metodologias de testes

Uma das características mais importantes na metodologia está assente na interacção. Em cada interacção são adicionados novos testes e refinados outros, construindo uma base de trabalho e podendo responder às devidas adaptações do projecto. Assim pode-se medir a completude da interacção, através da verificação dos requisitos implementados.

Ciclo de vida dos testes

Umas importantes características da metodologia utilizada pela consultora têm a ver com o conceito de vida dos testes, que consiste nos seguintes passos: planeamento dos testes, *design* dos testes, implementação dos testes; execução dos testes (de integração e de sistema) e avaliação dos testes.

Coberturas e métricas dos testes

Outra questão refere-se à completude dos testes, ou seja, até que ponto os testes conseguem aferir a qualidades do *software* alvo. Assim é usada as seguintes métricas *Requirements-based* e *Code-based*, com o objectivo de verificar essa mesma completude.

Estas métricas consistem na verificação dos requisitos, como medida quantificável de completude dos testes (*Requirements-based*) e na quantidade de código fonte foi executado por testes (*Code-based*).

Ferramentas

A consultora usa normalmente dois pacotes de referência do mercado, usando pontualmente outras ferramentas de outros fornecedores. Ainda de referir os *testwares* específicos desenvolvidos pela própria consultora, que têm sido adaptados e reutilizados em vários projectos.

Testes de aceitação

A consultora utiliza duas estratégias de aceitação para a implementação dos testes, que são acompanhadas e planeadas com o cliente:

- **a aceitação formal** - onde são utilizadas com o extensão do testes de sistema, sendo planeadas e executadas com o mesmo detalhe destes;
- **a aceitação informal** - onde não definidos de forma tão rigorosa como os da aceitação formal.

Metodologia de testes da arquitectura orientada a serviços

Este tipo de testes é baseado no ciclo de vida em cada uma das camadas de abstracção, onde cada uma desta camada é alvo de testes específicos.

6.4. Gestão da melhoria do processo da qualidade de *software*

Esta secção tem como objectivo fornecer uma visão do diagnóstico, definição e melhoria do processo de *software* de uma pequena empresa que actua na área da *Internet* e telecomunicações.

Descrição da empresa

Este caso de estudo baseia-se numa pequena empresa de prestação de serviços que actua na área da ligação à *Internet*, desenvolvimento de *sites* e serviços de telecomunicações.

A empresa investe continuamente na melhoria da qualidade de processos de desenvolvimento, procurando uma melhor gestão e controlo das actividades internas, com o intuito de satisfazer os clientes, melhorar os seus produtos e a qualidade dos seus funcionários. Estão a preparar a certificação CMM – nível 2, esperando assim superar as expectativas dos seus clientes e satisfazer o crescente mercado da *Internet*.

Descrição do caso de estudo

O principal objectivo do caso de estudo é dar início à melhoria do processo de desenvolvimento de *software*, com base no nível 2 do CMM. No início foram executadas 2 actividades (diagnóstico da situação actual e definição do processo da empresa) e onde foi definido seguidamente um plano de melhoria para a respectiva empresa.

- **Diagnóstico da situação actual**

Para avaliar e diagnosticar a situação actual, foi efectuado um questionário ao director e gestores de áreas de desenvolvimento. Após a avaliação das respostas do questionário, foi elaborado o relatório de actividades de acordo com as directrizes para a melhoria da qualidade, contendo práticas-chave que a empresa deveria realizar para atingir a respectiva certificação.

- **Definição do processo da empresa**

Esta actividade foi efectuada em reuniões com o director e respectiva equipa, ficando estabelecido que em relação à terminologia, continuaria a ser a que a empresa já utilizada. Através do diagrama de fluxo de dados e formulários que foram sendo preenchidos, alguns pontos de melhoria foram observados.

- **Melhorias decorrentes da identificação da situação actual**

Ao definir o processo da situação actual, apercebe-se que alguns pontos poderiam ser melhorados através da forma da realização das tarefas e onde se decidiu alterar essas mudanças e propor um plano de melhoria mais amplo. No final deste passo, a empresa ficou com um processo de desenvolvimento de aplicações bem definido, mais eficiente em relação à situação inicial.

- **Utilização do processo definido**

Com a definição do processo já efectuado, o mesmo foi utilizado no desenvolvimento de um *site*, com o objectivo de verificar a consistência do processo, das vantagens e desvantagens e possíveis melhorias.

- **Planeamento da melhoria do processo**

Com os resultados do diagnóstico e definição do processo actual efectuado, o passo seguinte é a aplicação das directrizes da melhoria. Assim foi criado o plano de acção, que contêm as actividades do relatório de actividades e do relatório de pré-condições e onde descriminam as actividades a serem executadas pela empresa de acordo com as prioridades.

Conclusão

Através das pequenas alterações realizadas no processo, provocou melhorias sentidas pelos funcionários da empresa e onde contribui para a mudança cultural, podendo trabalhar de uma forma mais organizada e que influenciou o cumprimento do prazo da entrega do produto.

6.5. Conclusão

Neste capítulo foram descritos casos de estudos de empresas e como fazem a gestão dos testes e qualidade do *software* no seu dia-a-dia e que serviu para exemplificar o uso de conceitos estudados em projectos reais.

No caso de estudo da seguradora, consegue-se uma melhoria da qualidade do *software* através dos testes da aplicação em ambiente de teste.

Na consultora de sistema de informação a experiência adquirida nos diversos projectos, as certificações obtidas, a utilização de metodologias e ferreamentas de teste permitiram evoluir na área da qualidade.

Na empresa que actua na área da Internet e telecomunicações, a qualidade no desenvolvimento de *software* obtêm-se através da certificação.

Capítulo 7.

Considerações finais

O processo de testes e qualidade deve ser considerado em todas as etapas do processo de desenvolvimento de *software*. Embora não se tenha consciência da importância real dos testes, é uma actividade que tem vindo a ganhar importância nas organizações e quando aplicada convenientemente diminui o custo de manutenção e a quantidade de erros encontrados. Neste contexto, conceitos como prevenção, detecção, avaliações, auditorias e avaliações de métricas devem ser entendidas como fases do projecto para garantir a qualidade da aplicação.

Foram apresentadas ao longo da tese várias técnicas que podem usadas nos testes e qualidade de *software*, com o objectivo de ser usadas para nos ajudar a ultrapassar os problemas do dia-a-dia, aquando do desenvolvimento do produto. No final apresenta-se casos de estudo de empresas e a forma como estas aplicam o processo de testes e qualidade de *software* ao longo do desenvolvimento das aplicações.

Nem sempre são implementados todos os testes e métodos de qualidade que se espera serem uma ajuda preciosa no desenvolvimento do produto e mais fáceis de manter, embora se possa verificar esforços com o objectivo de aumentar a qualidade do *software* em geral.

Quanto ao caso de estudo da seguradora, verifica-se que se teve o cuidado de enviar a aplicação para um ambiente de testes com intuito de verificar se o produto está de acordo com o esperado e serem corrigidas as falhas, antes de enviar para o ambiente de produção. Falta melhorar alguns aspectos, como por exemplo a análise, a documentação e integração com testes de carga e desempenho, quer do próprio produto, que com outras possíveis aplicações.

No caso de estudo da consultora de sistemas de informação, verifica-se o cuidado que houve em pôr em prática as *best-practices* na área da qualidade, não havendo referência à documentação para controlo dos testes e qualidade do *software*.

Em relação à empresa que actua na área da *Internet* e telecomunicações, pode-se verificar que já existe o cuidado de se certificar e serem mais competitiva na área da qualidade.

Assim ao longo da dissertação, pretendeu-se dar uma visão da importância da utilização dos requisitos, modelos, testes e qualidade do *software*, com o objectivo de que este trabalho possa constituir um contributo para uma melhoria das práticas relacionadas com esta área.

Bibliografia

[1] Ian Sommerville, Engenharia de *Software*, Pearson – Addison Wesley, 2005, ISBN – 84-7829-074-5

[2] António Miguel, Gestão do Risco e da Qualidade no Desenvolvimento de *Software*, FCA, 2002, ISBN – 972-722-333-8

[3] Roger S. Pressman, Engenharia de *Software*, McGraw Hills, 2002, ISBN: 85-86804-25-8

[4] Simão, Inês e Varela, Patrícia, Universidade Nova de Lisboa, "A Engenharia de Requisitos como processo inovador nas organizações", http://run.unl.pt/bitstream/ 10362/1972/1/WPSeries_08_2009ISimao_PVarelaB.pdf" – acedido em 08/12/2009

[5] Turine, Marcelo Augusto Santos e Masiero, Paulo Cesar, Universidade de São Paulo, "Especificação de requisitos: uma introdução", http://www.san.uri.br/ ~pbetencourt/engsoftII/documentorequisitos.pdf – acedido em 08/12/2009

[6] de Moraes, Marcela Balbino Santos, Universidade Federal de Pernambuco, "Processo de Engenharia de Requisitos e Linhas de Produto de *Software*", http://www.cin.ufpe.br/~in1020/arquivos/monografias/2008-1/processo%20de%20engenharia %20de%20requisitos%20em%20linhas%20de%20poduto%20de%20software.pdf – acedido em 08/12/2009

[7] Alves, Carina Frota, Universidade Federal de Pernambuco, "Uma Experiência de Engenharia de Requisitos em Empresas de *Software*", http://sistemas.dis.ufro.cl/eig2008/files/papers/Alves(pp13-24).pdf – acedido em 08/12/2009

[8] Figueiredo, Carlos et al, Universidade do Porto, " Especificação Formal de *Software*", http://paginas.fe.up.pt/~ei99030/trabalhos/EspecificacaoFormaldeSoftware.pdf – acedido em 08/12/2009 dos Santos, Gilberto, Universidade Estadual de Londrina, "A Necessidade e a Importância da Engenharia dos Requisitos", http://www2.dc.uel.br/nourau/document/?code=739 – acedido em 08/12/2009

[9] dos Santos, Gilberto, Universidade Estadual de Londrina, "Necessidade e a Importância da Engenharia dos Requisitos", http://www2.dc.uel.br/nourau/document/?code=739 – acedido em 08/12/2009

[10] Wiegers, Karl "Automating Requirements Management", http://www.processimpact.com/articles/rm_tools.pdf - acedido em 18/01/2010

[11] "Requirements Managements Tools" http://www.jiludwig.com/ Requirements_Management_Tools.html – acedido em 18/01/2010

[12] "10 golden rules for Requirements Based Testing", http://itknowledgeexchange. techtarget.com/quality-assurance/10-golden-rules-for-requirements-based-testing/ – acedido em 17/01/2010

[13] David A. Gustafman, Engenharia de *Software*, Coleção Schaum, Bookman, 2003, 85-363-0185-6

[14] "Major 30 *Software* Development Mistakes", http://www.streetdirectory.com/ travel_guide/135704/software/major_30_software_development_mistakes.html - acedido em 17/01/2010

[15] Spectrum IEEE, "Why *Software* Fails", http://spectrum.ieee.org/computing/ software/why-software-fails - acedido em 17/01/2010

[16] Topolsky, Joshua, "Live from Steve Ballmer's CES 2010 keynote", 06/01/2010, http://www.engadget.com/2010/01/06/live-from-steve-ballmers-ces-2010-keynote/ – acedido em 20/01/2010

[17] Lousa, Hugo Antônio de Azevedo e Nunes, Carla de Tunes, Universidade de Brasília, "Automação de Testes Utilizando Ferramentas Open Source", http://monografias.cic.unb.br/dspace/bitstream/123456789/36/1/monografiaDG.pdf – acedido em 30/11/2009

[18] Oliveira, Patricia Lemos: http://www.uniritter.edu.br/graduacao/informatica/sistemas/downloads/tcc2k7-2/PatriciaLemosOliveira.pdf pdf – acedido em 12/05/2009

[19] Quintela, Heitor, Dias, Roni e Teixeira, Manoella, "Análise dos impactos na qualidade de *Software* em instituições financeiras segundo a norma ISO/IEC 9126 na adoção das práticas de testes do modelo CMMI", http://www.producao.uff.br/conteudo/rpep/volume62006/RelPesq_V6_2006_03.pdf – acedido em 17/01/2010

[20] Shari Lawrence Pfleeger, Engenharia de *Software* – Teoria e Prática – 2ª Edição; ISBN: 85-87918-31-1; Editora: Prentice Hall

[21] Bernardo, Paulo Cheque e Kon, Fabio, Engenharia de *Software* Magazine "A Importância dos Testes Automatizados – Controle ágil, rápido e confiável de qualidade", http://www.ime.usp.br/~kon/papers/EngSoftMagazine-IntroducaoTestes.pdf – acedido em 03/12/2009

[22] Qualidade_de_Software_Resumo.doc, http://Br.geocities.com/conhecimentovirtual/Qualidade_de_*Software*_Resumo.doc – acedido em 12/05/2009

[23] Qualidade de *software*, Universidade de Passo Fundo, Instituto de Ciências Exatas e Geociências, Ciência da Computação – acedido em 11/06/2009

[24] Abreu, Fernando Brito e, Pereira, Gonçalo (1996), Inesc, "Dos Princípios da Engenharia de Software e do seu Impacto na Qualidade: Conceitos, Técnicas e Recomendações para a Codificação" – acedido em 12/05/2009

[25] MSW Métricas e *Software*, http://www.mswconsult.com.br/quali_4.html – acedido em 09/06/2009

[26] Porto, Daniel P. et al, "Avaliação da Confiabilidade de um *Software* utilizando Aspectos", http://www.ucb.br/prg/professores/anquetil/Publicacoes/sbqs06.pdf - acedido em 16/11/2009

[27] Sabbatini, Renato, Revista Informédica, 1993, "Problemas Éticos no Uso do *Software* de Apoio à Decisão Médica", http://www.informaticamedica.org.br/informed/etica.htm – acedido em 19/01/2010

[28] Abreu, Fernando Brito e, et al, "Organizações e Iniciativas Nacionais e Internacionais em Prol da Qualidade no Software", 1º Encontro Nacional para a Qualidade nas Tecnologias de Informação e Telecomunicações, Lisboa, 27 de Maio de 1994, http://www-ctp.di.fct.unl.pt/QUASAR/Resources/Papers/others/quatit94.pdf – acedido em 17/11/2009

[29] Abreu, Fernando Brito e, "Normalização e Certificação em Engenharia de *Software*", http://ctp.di.fct.unl.pt/QUASAR/Resources/Papers/others/encee94.pdf – acedido em 17/11/2009

[30] da Silva, Danila Santos, et al, Faculdades Jorge Amado, "SiTest: Uma metodologia de teste de *software* com base no modelo CMMI" – acedido em 11/06/2009

[31] Manera, Aline Fátima, et al, "Modelo de Qualidade – CMMI ", Universidade Federal de São Carlos – Departamento de Computação, 06 de Setembro de 2007, http://www.comp.ufscar.br/~bruno_abrahao/Artigos/CMMI.pdf, acedido em 17/10/2009

[32] Abreu, Fernando Brito e, INESC/ISEG, Dezembro 1992, "As métricas na Gestão de Projectos de Desenvolvimentos de Sistemas de Informação", http://ctp.di.fct.unl.pt/QUASAR/Resources/Papers/Metrics/apq_1992.pdf – acedido em 22/11/2009

[33] Câmara, Rafael," Atividade 2: Plano de Qualidade Maduro", http://sites.google.com/site/camararg/Home/lista-de-exercicios/atividade-2-plano-de-qualidade-maduro – acedido em 10/06/2009