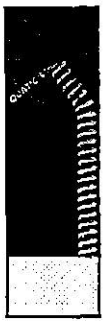


Métricas e Documentação na Qualidade do Software

Joaquim da Conceição Ferreira
Portugal Telecom S.A. – DID/CET

Manuel Marques Crisóstomo
Instituto de Sistemas e Robótica

Resumo

O presente trabalho trata da implementação da qualidade do *software* nas organizações. São apresentadas as métricas de caixa branca e métricas de caixa preta aplicadas ao *software*.

As métricas de caixa preta baseiam-se no princípio de que não é necessário conhecer o programa nem a linguagem em que está implementado, mas sim as suas funcionalidades. É a partir destas que se especificam os procedimentos de testes.

Já nas métricas de caixa branca, é necessário conhecer-se os nós de decisão, como estão implementados os espaços de memória, etc. Em resumo, as métricas de caixa branca tratam da construção interna do programa.

Analisa-se a fundamentação dos princípios da qualidade baseados na experimentação pragmática e racional de Lewis [1], como base da pratica actual da Gestão da Qualidade.

É também tratada a principal fonte de erros ou falhas nos programas de *software*, bem como metodologias para os detectar e corrigir.

Também se abordam aspectos relacionados com a documentação de *software*, nomeadamente a elaboração do plano de testes, procedimentos e casos de testes e

respectivos relatórios. Os relatórios de testes subdividem-se em três tipos distintos de documentos: relatório de gestão, relatório de incidentes e relatório final.

Qualidade e Medidas

Os conceitos de qualidade estão baseados em princípios de experimentação e possibilidade de repetição da experiência ou teste. Daqui se induz que todo conhecimento que está baseado em princípios não mensuráveis, não pode ser entendido como científico [1].

De facto, foi o filosofo da ciência Charles S. Peirce que apresentou as bases da objectividade do conhecimento ou pragmatismo em oposição a uma visão do conhecimento holistica e espiritualista de William James.

Embora concordando com o pragmatismo, C. I. Lewis acrescenta que "os dados da experiência não têm significado sem a respectiva interpretação dos conceitos". A teoria do conhecimento de Lewis, pode ser entendida como pragmática e racional.

Tendo como base que a tomada de decisões requer um conhecimento fundamentado em dados tratados, bem como reconhecendo que a um sistema de medida está associado um certo grau de incerteza, tanto Shewhat como Deming, introduzem a utilização comum das ferramentas estatísticas, método sigma-três e outros, como base de interpretação e tratamento de dados, dando origem à moderna interpretação da qualidade.

Sabendo que o conhecimento se aprofunda com a prática, Shewhat cria o ciclo PDSA (plano, fazer, estudo e acção) que pode ser considerado um modelo que utiliza o método científico para realizar a experimentação, a aprendizagem e o

melhoramento. Os elementos chave para esta aproximação, são: hipótese, experimentação, teste da hipótese e acção.

Temos pois, que os conceitos da Gestão pela Qualidade Total assentam tanto no tratamento estatístico dos dados obtidos no processo de produção, como requer uma avaliação contínua dos métodos em uso, no sentido de os ir melhorando.

Pensa-se que a avaliação da qualidade do *software*, se deve basear nestes princípios. Ou seja as métricas de *software* devem estar associadas a medidas que possam ser repetidas bem como, à existência de um registo e tratamento dos dados obtidos, de forma a determinar as características de funcionamento desse mesmo *software*.

Distribuição dos Erros de *Softwre*

Na área da qualidade do *software*, entende-se que devam existir vários passos no seu processo de desenvolvimento que devem ser seguidos para se obter um produto conforme os requisitos e projecto funcional que tiverem sido especificados e acordados entre as partes contratantes, cliente e fornecedor [2].

Entende-se que os testes de software, não devem começar com o produto acabado mas sim durante a fase de especificação, projecto e desenvolvimento deste. É sabido que testar o software acabado, constitui um método bastante ineficiente do ponto de vista da qualidade.

De acordo com *o Quality Assurance Institute* [3], mais de 55% dos erros são introduzidos nas fases de projecto e análise. Acrescenta ainda que quanto mais tarde no ciclo de vida do software o erro for detectado, mais cara será a sua correcção. Um erro detectado durante a fase final do código custará mais de 100 vezes do que se ele fosse detectado durante a fase de projecto.

Na prática, os erros ou falhas do *software* podem ser considerados erros humanos, dado que a probabilidade da máquina falhar, em condições controladas de utilização, é extremamente baixa.

Como pode ser visto na figura 1 a maior parte dos erros de *software*, 56%, advêm da especificação dos requisitos. Isto deve-se ao facto de que tanto o analista de sistemas como o cliente, quando encomenda o *software*, têm dificuldades e por vezes não conseguirem elaborar o caderno de encargos com clareza e de forma inequívoca transcrevendo de forma correcta as necessidades reais do cliente.

Para diminuir esta quantidade de erros, o cliente deve realizar várias reuniões com o fornecedor e apresentar os planos da empresa a curto e médio prazo, para que o *software* a desenvolver não esteja ultrapassado quando estiver pronto. Por outro lado, deve ser compatível e funcional com o *software* que o cliente já possua ou venha a adquirir.

A fase de especificação do projecto, 27% das falhas, corresponde à passagem dos requisitos para modelos lógicos e matemáticos de modo a serem mais tarde codificados em qualquer linguagem. Esta fase é muito dependente da experiência profissional do analista de sistemas.

Dado que na maior parte dos programas é impossível apresentarem-
-se modelos matemáticos demostráveis para que o programa seja implementado, podem ocorrer falhas na especificação do projecto que se transmitem em cadeia até que o produto final esteja concluído. Este tipo de erro é bastante difícil de detectar, mesmo na fase de testes finais ao *software*.

Os erros humanos, cometidos durante a fase de codificação, são geralmente mais fáceis de detectar, dado que existem compiladores bastante completos que auxiliam no diagnóstico da anomalia. Por outro lado também existem ferramentas de teste para se verificar a complexidade do programa.

Comparando os dados das falhas obtidos em [3] com os dados de [6], verifica-se que estão coerentes e que a fonte principal dos erros de *software* está na especificação dos requisitos.

Se utilizarmos um modelo de Pareto, em que se deve começar o processo de melhoria da qualidade pelos pontos em que ocorrem mais erros devemos concluir, que em termos de qualidade do *software* o processo de melhoria deve ser iniciado e focado na melhoria da especificação dos requisitos dos programas de *software*.

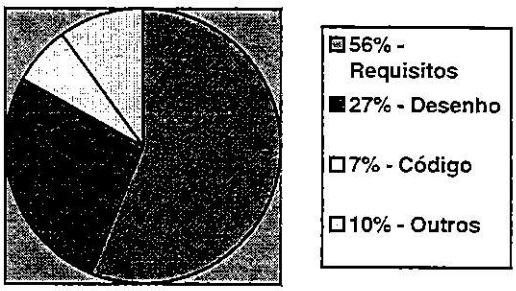


Figura 1 – Distribuição de falhas do *Software* [6]

Métricas de Software

A quantidade do *software* utilizado nas empresas operadoras de telecomunicações é hoje muito elevada e está em constante crescimento. Por outro lado é importante que a qualidade deste *software* seja funcional, fiável e robusta de modo a satisfazer as necessidades dos clientes [7].

Para que a qualidade possa ser quantificada, ela tem de ser medida. Na análise do *software,* uma métrica corresponde a um conjunto de procedimentos observáveis e que se podem repetir, sobre um programa computacional e que é obtido um resultado.

As métricas podem ser utilizadas para a gestão de desenvolvimento e processo de aquisição e devem ser relevantes para o produto particular de *software* que vai ser utilizado.

Para as organizações que não possuem sistemas de qualidade implementados, tais como modelo SEI-CMM, ISO/IEC12207 e ISO9001-3 ou outro, como mínimo, algumas métricas devem ser utilizadas.

E como forma de motivar e despertar a utilidade das métricas de *software,* na organização, pode ser iniciado o processo de avaliação do *software*, pelas métricas de caixa branca e métricas de caixa preta apresentadas a seguir.

Sempre que se utilizem métricas, estas devem ser descritas e documentadas para que os resultados obtidos possam ser comparáveis.

Existem, basicamente, dois tipos de testes de *software*, testes de caixa branca e testes do tipo caixa preta.

Nos teste do tipo caixa branca, o programa e respectivo código, são abertos para se verificar a sua complexidade, as ligações lógicas e valores que as diferentes variáveis vão tomando dentro do programa.

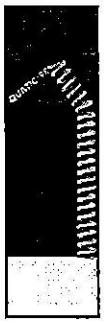
Nos testes do tipo caixa preta, só nos interessa controlar as entradas e saídas (respostas) programa.

Deve ser notado que os testes de caixa preta são executados normalmente pelo cliente, ou, na óptica deste. O cliente normalmente não se interessa em saber como as partes do programa estão implementadas, mas sim como ele funciona, a sua robustez, permutabilidade e manutenção etc.

Métricas de Caixa Branca

Nos testes de caixa branca é verificada a estrutura interna do programa em análise. Pode-se verificar através de testes que cada módulo do programa foi verificado pelo menos uma vez.

Um dos aspectos das métricas de *software*, consiste em verificar a correcta gestão dos recursos da memória a que um programa tem acesso. Não ter isso em consideração, gera por vezes gastos excessivos da memória de um computador ou



base de dados, bem como podem acarretar a não conveniente definição e iniciação de um "array" ou ponteiro. Uma das ferramentas de teste que executa uma boa gestão da memória é o *Purify* [4].

Para além da gestão da memória temos também de analisar o tempo gasto para que os ciclos de um programa decorram. A ferramenta *Qualify* [4] destina-se à analise de funções, mostra-nos o número de vezes que cada função é chamada e dá-nos uma aproximação do número de ciclos por função, com indicação do tempo máximo e mínimo, para que o programa seja executado. Esta aproximação é extremamente importante pois, a partir dela podemos ter uma boa aproximação do tempo gasto por função para correr o programa.

A título de exemplo, da verificação de software nas características de funcionalidade e fiabilidade, cita-se o caso ocorrido com o foguetão Ariane lançado de Cauro (na Guiana Francesa) em Junho de 1996, que transportava um satélite para telecomunicações. Aquele foguetão teve que ser destruído (passados 45 segundos do seu lançamento) dado não se poder prever, corrigir e acompanhar a trajectória deste, por falta de memória no computador de bordo. Os peritos detectaram que não tinha sido reservado espaço de memória suficiente para os dados referentes às variáveis de controlo horizontais das características atmosféricas (velocidade, intensidade e direcção do vento, etc.).

Dentro das medidas de caixa branca, temos também a métrica de complexidade de um programa, que quantifica a sua complexidade lógica do projecto, mede o número dos testes de integração, mostra também as partes do código que foram testadas [5]. Para McCabe a métrica de complexidade (Cc) pode ser calculada por:

$$\mathbf{Cc = E - N + 2P}$$

onde,

Cc – Métrica de Complexidade

E – Número de ramos
de transferência de controlo

N – Número de nós (*grupo sequencial de*

declarações contendo sòmente uma transferência de controlo).

P – Número de caminhos controláveis do programa.

Os resultados da métrica de complexidade, podem ser comparados com o número de linhas de código assim como o número de defeitos do programa testado.

Métricas de Caixa Preta

As medidas de *software* do tipo caixa preta obedecem a um planeamento bastante rigoroso, para que os seus resultados possam ser mais tarde comparáveis e possíveis de repetir.

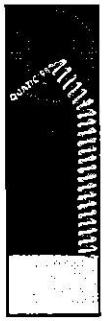
A elaboração dos casos e procedimentos (métricas) de teste do tipo caixa preta é feita a partir da especificação dos requisitos do programa e manual do utilizador tendo em atenção as suas funcionalidades.

Os casos de teste são constituídos pelo somatório das partes em que um programa possa ser subdividido na sua apresentação final.

A um conjunto de casos de teste chamamos de módulo de teste, que deve corresponder a um módulo do programa.

Depois de identificados todos os casos de teste, procede-se à elaboração e documentação do procedimento teste para verificação e execução de cada teste.

Deve ser estipulado, pelo menos um caso de teste para cada função diferente do programa havendo funções mais complexas que devam ser estipulados vários casos de teste para a mesma função, no sentido dela ser completamente descrita e testada.



Além dos testes funcionais, podem também ser executados testes de *stress* ou de carga do programa. Ou seja, criar situações em que o programa tenha que executar um número de operações muito superior ao esperado na realidade.

Deverão também ser prescritos testes de campos, ou seja introdução de valores não esperados tais como:

a) Valores reais ou alfabéticos, quando o programa espera valores inteiros.

b) Valores negativos para datas e idades.

c) Ter em atenção o problema do *"ano 2000"*, como mínimo: verificar se o campo referente ao ano tanto à entrada como à saída, apresenta os quatro dígitos.

d) Verificar o comprimento de cada campo de dados, por exemplo aonde se espera que exista espaço para 20 caracteres, tentar colocar 21 ou mais caracteres.

Após terminar a execução de todos os módulos do programa, iniciam-se os testes de integração de todo o programa ou sistema.

Terminada a fase dos testes de integração e depois do programa estar a funcionar completamente, procede-se aos testes do sistema. Os testes do sistema são executados na óptica do fornecedor e devem ser equivalentes aos testes de aceitação executados pelo cliente.

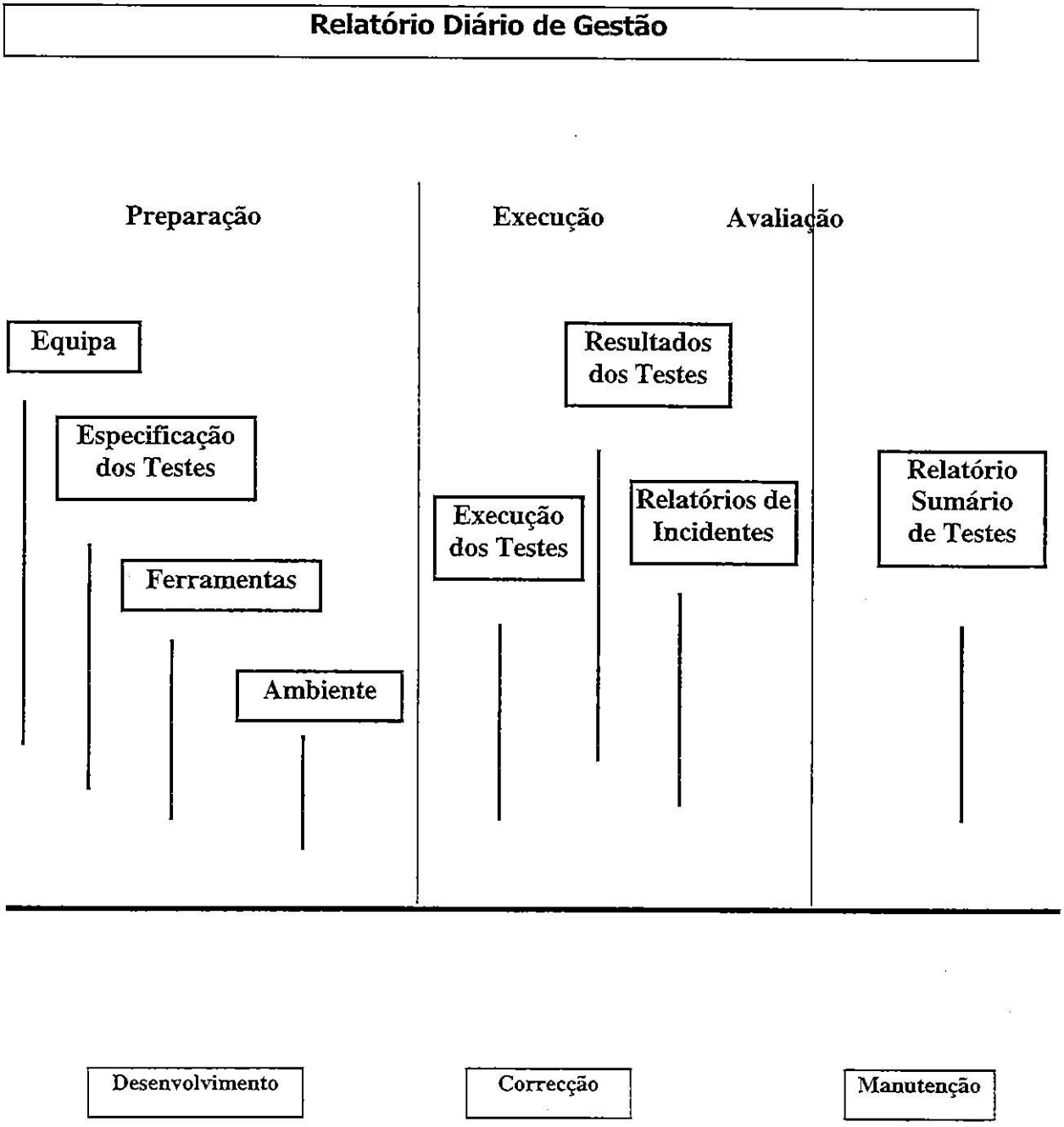


Figura 2 – Planeamento do processo de testes de *software*

Documentação e Qualidade

A documentação, para testes do *software* é um dos aspectos mais importantes para se determinar a qualidade deste tipo de produto e deve ser relevante, dentro das organizações, como uma actividade produtiva e essencial.

Como iremos ver mais à frente, o software possui várias fases ou ciclos e é desenvolvido por grupos de pessoas distintas. Para que todo o processo se possa manter sob controlo é necessário que haja uma concatenação entre as pessoas que é obtida através da documentação de suporte em que cada um sabe exactamente as suas atribuições.

A *Norma Std/IEEE – 829 Standard for Software Test Documentation*, apresenta um modelo de documentação para testes de software que é constituído por oito documentos:

1 – Plano de Testes,

2 – Especificação de Desenho de Testes,

3 – Especificação de Caso de Testes,

4 - Especificação de Procedimento de Testes,

5 – Relatório de transmissão de um item de teste,

6 – Relatório do Diário de Teste,

7 – Relatório do Incidente de
Teste,
8 - Relatório Sumário de
Testes.

A figura 2 representa um diagrama de causa e efeito, tipo "Ishikawa". Este diagrama deve ser lido da esquerda para a direita e representa os diferentes recursos e processos do método de testes do software.

A fase de especificação de testes, corresponde a: especificação de desenho, especificação de procedimento e relatório de transmissão de um item de teste.

O relatório de incidente de teste deve ser elaborado quando ocorrer durante a execução de testes algum acontecimento não previsto com o software que por exemplo, force a interrupção dos testes.

O objectivo do Plano de Testes, consiste em definir os recursos, abordagem e âmbito, bem como o calendário das diversas actividades de teste. Deve também identificar os itens, funções e características a serem testados, as tarefas de teste a serem executadas, as pessoas responsáveis por cada tarefa, e o risco associado com o plano.

A estrutura do Plano de Testes, é composta pelos itens identificados abaixo:

1. Identificador do plano de testes
2. Introdução
3. Itens de testes
4. Funcionalidades e características a testar
5. Funcionalidades e características a não testar
6. Metodologia

7. Critério passa e falha
8. Critérios para suspensão dos testes e requisitos para recomeço.
9. Documentos de testes
10. Tarefas de teste
11. Necessidades de ambiente
12. Responsabilidades
13. Equipa de testes e necessidade de treino
14. Calendário
15. Riscos e contingências
16. Aprovação.

Para cada programa ou sistema de *software,* deverá existir apenas um único plano de testes, que cobrirá todas as necessidades de recursos técnicos e humanos.

Conclusões

Neste trabalho abordou-se a origem da filosofia da qualidade total, apresentaram-se os conceitos pragmáticos e racionais e necessidade da experimentação como modelo de obtenção e tratamento de dados de modo a tornar fiável a tomada de decisões.

Focou-se também a qualidade do software, como modelo experimental com ênfase na elaboração de documentação de teste, como base para futuras verificações e enquadramento dos resultados obtidos.

Foi feito o enquadramento das medidas de teste de caixa branca e preta que apresentam resultados bastante fiáveis e expeditos de se realizar.

Em conclusão, apresentou-se a necessidade da documentação formal e rigorosa como meio de se ter a certeza de que o programa em teste funciona exactamente de acordo com os requisitos estipulados e acordados entre cliente e fornecedor.

Bibliografia

[1] - Michael R. Lovitt *The New Pragmatism: Going Beyond Shewhart and Deming* (Quality Progress, April 1997).

[2] - Qualidade e Telecomunicações, 3ª Conferencia da Engenharia Electrotécnica da Ordem dos Engenheiros, Exponor – Matosinhos de 20 a 24 de Junho de 1997, autores Joaquim da Conceição Ferreira, Manuel Marques Crisóstomo.

[3] –Zohar Gilad, *Automated Software Quality Solutions* (Mercury Interactive Corporation, 1995).

[4] – Free software fundation, Inc. 1993.

[5] – McCabe & Associates UK Ltd YousufQ@McCabe.Co.UK.

[6] – Software Development Tecnologies – US 1994.

[7] –Software Quality Assurance –
EURESCOM, Project P227, 1994.

Temas da Conferência abrangidos:

- Qualidade da comunicação nas telecomunicações,
- Métodos, técnicas e ferramentas de verificação e validação,
- Métricas de complexidade do software.

Palavras chave: Qualidade, Software, Métricas, Metodologias, Falhas e Documentação.