Processos, Qualidade e Gestão de Software

v2 4 nov 2009

Indice

[8.2 O RUP e suas características 102](#_Toc245121480)

[8.3 Visão Geral do RUP 103](#_Toc245121481)

[8.2.1 Concepção 105](#_Toc245121482)

[8.2.2 Elaboração 106](#_Toc245121483)

[8.2.3 Construção 107](#_Toc245121484)

[8.2.4 Transição 108](#_Toc245121485)

[Introdução ao RUP:Rational Unified Process-“ Phillippe Kruchten” 125](#_Toc245121486)

[3.1. Introdução a Processos Ágeis de Desenvolvimento de Software 102](#_Toc245121487)

[3.2. O Manifesto Ágil 103](#_Toc245121488)

[3.3. Principais Processos Ágeis 104](#_Toc245121489)

[3.4 EXTREME PROGRAMMING (XP) 106](#_Toc245121490)

[3.4.1 Valores, princípios e práticas de XP 106](#_Toc245121491)

[3.4.2 Papéis dos integrantes 108](#_Toc245121492)

[3.4.3 Ciclo de Vida 109](#_Toc245121493)

[3.6. Scrum 109](#_Toc245121494)

[3.6.1 Características do Scrum 110](#_Toc245121495)

[3.6.2 Papéis do Scrum 110](#_Toc245121496)

[3.6.3 Práticas do Scrum 111](#_Toc245121497)

[3.6.4 Ciclo de Vida do Scrum 111](#_Toc245121498)

[3.7 FEATURE DRIVEN DEVELOPMENT 111](#_Toc245121499)

[3.7.1 Características do FDD 112](#_Toc245121500)

[3.5.2 Papéis do FDD 112](#_Toc245121501)

[3.5.3 Práticas do FDD 114](#_Toc245121502)

[3.5.4 Ciclo de Vida do FDD 115](#_Toc245121503)

[9.7. Considerações Finais 117](#_Toc245121504)

[9.8. Tópicos de Pesquisa 118](#_Toc245121505)

[9.9. Sugestões de Leitura 118](#_Toc245121506)

[9.11. Exercícios 118](#_Toc245121507)

[Referências 118](#_Toc245121508)

3.1 Introdução 121

3.2 Desenvolvimento Distribuído de Software 121

3.2.1 Motivações para o DDS 122

3.2.1 Níveis de Dispersão 123

3.2.2 Modelos de Negócio 124

3.2.3 Desafios 125

 Arquitetura do *Software*: É um dos fatores mais utilizados para a diminuição do esforço entre as equipes. Conforme KAROLAK [1998], uma arquitetura apropriada para o DDS deve se basear no princípio da modularidade, pois permite alocar tarefas complexas de forma distribuída. Com isso há uma redução na complexidade e é permitido um desenvolvimento em paralelo simplificado. 126

 Engenharia de Requisitos: A engenharia de requisitos contém diversas tarefas que necessitam de alto nível de comunicação e coordenação. Com isso os problemas apresentados são mais complexos em um contexto de DDS. 126

 Gerência de Configuração: O gerenciamento de configuração (CM) é a chave para controlar as múltiplas peças em um projeto distribuído. Controlar modificações nos artefatos em cada uma das localidades com o processo de desenvolvimento de todo produto pode ser complexo. Apesar da utilização de práticas de CM auxiliar no controle da documentação e do software, a gerência de modificações simultâneas a partir de locais diferentes é um grande desafio. Além disso, o uso de ferramentas de CM compartilhadas por duas ou mais equipes de forma inadequada gera diversos riscos e problemas em projetos DDS. 126

 Processo de Desenvolvimento: Em projetos DDS, o uso de uma metodologia que auxilia a sincronização das atividades é essencial. Com isso todos os membros utilizam uma nomenclatura comum em suas atividades. 126

3.3 Processos para Desenvolvimento Distribuído de Software 126

Segundo PRESSMAN [2001], processo é a camada mais importante da Engenharia de Software, fazendo uma ligação entre ferramentas e métodos. Um processo de software é um conjunto de atividades que definem a seqüência em que os métodos serão aplicados e como o produto será entregue, além disso, define os controles que asseguram a qualidade do produto e coordenação das mudanças. 126

Em um ambiente de desenvolvimento distribuído, um processo de desenvolvimento comum à equipe é fundamental, tendo em vista que uma metodologia auxilia diretamente na sincronização, fornecendo aos membros da equipe uma nomenclatura comum de tarefas e atividades, e um conjunto comum de expectativas aos elementos envolvidos no processo [PRIKLADNICKI 2008]. 126

A engenharia de software (ES) sempre está apresentando grandes avanços e transformações relacionadas às técnicas, modelos e metodologias. Esses avanços são destacados quando se trabalha com processo de Desenvolvimento Distribuído de *Software* (DDS), havendo uma necessidade do uso de práticas que dê suporte às dificuldades encontradas nas definições de requisitos que mudam de forma dinâmica no decorrer do tempo. Estudos relacionados a processo para DDS ainda é escasso, sendo assim este capítulo relata o uso de praticas do desenvolvimento tradicional que podem ser implantadas em um ambiente distribuído e as possíveis adaptações. 127

3.4 Processos e adaptação das Práticas em projetos DDS 127

A forma como um produto de *software* é concebido, desenvolvido, testado e entregue ao cliente sofre grande impacto quando o ambiente de desenvolvimento é distribuído [HERBSLEB 2001]. Assim, a estrutura necessária para o suporte desse tipo de desenvolvimento se diferencia da utilizada em ambientes centralizados. Diferentes características e tecnologias se fazem necessárias, crescendo a importância de alguns detalhes antes não percebidos. 127

3.4.1 Modelo de Karolak [1998] 128

Neste trabalho o autor aborda o DDS seguindo o ciclo de vida tradicional de um projeto de desenvolvimento de *software*. O autor propõe um modelo para desenvolver projetos DDS abrangendo as atividades que dever ocorrer ao longo do ciclo de vida. A figura abaixo ilustra o modelo proposto (Figura 3.2): 128

 128

Figura 3.2 – Modelo para projetos DDS [KAROLAK 1998] 128

3.4.2 Uso de Práticas Ágeis 130

3.4.2.1 DXP – Distributed Extreme Programming 131

3.4.2.2 Adoção de *Scrum* em um ambiente DDS 133

3.5 Oportunidades de Pesquisa 136

3.6 Considerações finais 137

3.7 Exercícios 137

3.8 Recomendações de Leitura 137

Referências 138

Herbsleb, J. D., Moitra, D. “Global Software Development”, IEEE Software, March/April, EUA, 2001, p. 16-20. 138

Karolak, D. W. “Global Software Development – Managing Virtual Teams and Environments”. Los Alamitos, IEEE Computer Society, EUA, 1998, 159p. 138

Kiel, L. “Experiences in Distributed Development: A Case Study”, In: Workshop on Global Software Development at ICSE, Oregon, EUA, 2003, 4p. 138

Kircher, M., Jain, P., Levine, A. “Distributed Extreme Programming”, IEEE, Agile 2008. 138

Herbsleb, J.D., Mockus, A., Finholt, T.A. e Grinter, R. E. “An empirical study of global software development: distance and speed”, In: ICSE 2001, Toronto, Canada. 138

Carmel, E. “Global Software Teams – Collaborating Across Borders and Time-Zones” Prentice Hall, EUA, 1999, 269p. 138

Marquardt, M. J., Horvath, L. “Global Teams: how top multinationals span boundariesand cultures with high-speed teamwork”. Davies-Black. Palo Alto, EUA, 2001. 138

Yin, Robert. “Estudo de Caso: planejamento e métodos”. SP: Bookman, 2001, 205p. 138

Young, C., Terashima, H. “How Did Wr Adapt Agile Processes to Our Distributed Development?”, IEEE, Agile 2008. 138

Prikladnicki, R., Audy, J. L. N., Evaristo, R. “Global Software Development in Practice: Lessons Learned”, Journal of Software Process: Practice and Improvement – Special Issue on Global Software Development, 2004. 138

Prikladnicki, R. “MuNDDoS: Um Modelo de Referência para Desenvolvimento Distribuído de Software”. Dissertação de Mestrado, PPGCC – PUCRS, Brasil, 2003. 138

Sommerville, Ian. Software Enginnering. 8.ed. [S.l] ADDISON WESLEY, 2007. 138

J. L. N. Prikladinicki, R.; Audy. Desenvolvimento Distribuído de Software. 2007. 138

Perrelli, Hermano. Visão Geral do RUP. Centro de Informática, Universidade Federal de Pernambuco. Disponível em: http://www.cin.ufpe.br/~if717/slides/3-visao-geral-do-rup.pdf. Acessado em 20 Maio 2009. 138

PRESSMAN, Roger S. Software Engineering: a practitioner’s approach. EUA: McGraw Hill, 2001. 860 p. 139

Sutherland, J., “Distributed Scrum: Agile Project Management with Outsourced Development Teams”, HICSS, 2007. 139

Teles, Vinícius Manhães. Extreme Programming: Aprenda como encantar seus usuários desenvolvendo software com agilidade e alta qualidade. 1. ed. São paulo: Novatec, 2004. 320 p. 139

Travassos, G. H.,Abrantes, J. F., “Caracterização de Métodos Ágeis de Desenvolvimento de Software” 139

[10.1 Introdução 142](#_Toc245121509)

[10.2 Arquitetura Dirigida a Modelos 144](#_Toc245121510)

[*10.2.1. Conceitos Básicos* 144](#_Toc245121511)

[*10.2.2. Padrões OMG e a Arquitetura MDA* 152](#_Toc245121512)

[10.3 Abordagens MDD Modelos 153](#_Toc245121513)

[*10.3.1 OO-Method* 154](#_Toc245121514)

[10.3.1.1. O processo básico de transformação 154](#_Toc245121515)

[10.3.1.2. Comparação com MDA 155](#_Toc245121516)

[10.3.1.3. O Modelo Conceitual 156](#_Toc245121517)

[10.3.1.4. O Compilador de Modelos 159](#_Toc245121518)

[10.3.1.5. OLIVANOVA 159](#_Toc245121519)

[*10.3.2 . AndroMDA* 160](#_Toc245121520)

[10.4 Problemas e Desafios dos Processos MDD 161](#_Toc245121521)

[*10.4.1. Visão Geral* 161](#_Toc245121522)

[*10.4.2.* Lições Aprendidas na adoção de soluções MDA 162](#_Toc245121523)

[*10.4.3. O programa FastStart da OMG* 162](#_Toc245121524)

[10.5. Tópicos de Pesquisa 163](#_Toc245121525)

[10.6 . Sugestões de Leitura 163](#_Toc245121526)

[10.7 . Exercícios 164](#_Toc245121527)

[Referências 167](#_Toc245121528)

[1. Introdução 169](#_Toc245121529)

[1.1. O que é Modelagem de Processos 169](#_Toc245121530)

[2.1. Objetivo da Modelagem de Processos Software 171](#_Toc245121531)

[3.1 Vantagens e Desvantagens da Utilização de Modelagem de Processos 171](#_Toc245121532)

[3.1.1 Vantagens 171](#_Toc245121533)

[ Bons modelos de processos (claros), são a chave para a boa comunicação. 171](#_Toc245121534)

[ Se o processo, é alguma coisa nova que a empresa está planejando executar, o modelo pode ajudar a assegurar sua eficiência desde o início. 171](#_Toc245121535)

[ Revelar anomalias, inconsistências, ineficiências e oportunidades de melhoria, permitindo à organização que se compreenda melhor e auxiliando na reengenharia desses processos. 171](#_Toc245121536)

[ Fornecer visão clara e uniformizada das atividades, suas razões e formas de execução. 171](#_Toc245121537)

[ Utilizar o modelo como um meio para distribuição de conhecimento dentro da organização e treinar as pessoas, ajudando-as a conhecer melhor seus papéis e as tarefas que executam. 171](#_Toc245121538)

[3.1.1 Desvantagens 171](#_Toc245121539)

[4.1. Linguagem de Modelos de Processos 172](#_Toc245121540)

[4.1.1.BPM 172](#_Toc245121541)

[4.1.2.SPEM 176](#_Toc245121542)

[4.2.1 OMG(Object Management Group) 182](#_Toc245121543)

[***Common Warehouse Metamodel ™*** 182](#_Toc245121544)

[***Common Object Request Broker Architecture®*** 183](#_Toc245121545)

[6.1. Ferramentas de Modelagem \*(colocar exemplo didatico) 183](#_Toc245121546)

[6.1.1. Comparação entra as Ferramentas 186](#_Toc245121547)

[7.1. Sugestões de Leitura 186](#_Toc245121548)

[8.1. Tópicos de Pesquisa 186](#_Toc245121549)

[Exercício 186](#_Toc245121550)

[Referencias 187](#_Toc245121551)

[6.1. Introdução 103](#_Toc245121552)

[6.2. O que é Qualidade? 103](#_Toc245121553)

[6.3. Competitividade x Produtividade 104](#_Toc245121554)

[6.3.1. Conceito de Produtividade 105](#_Toc245121555)

[6.3.2. Conceito de Competitividade 106](#_Toc245121556)

[6.4. Qualidade Total 108](#_Toc245121557)

[6.4.1. Deming 108](#_Toc245121558)

[6.4.2. Juran 109](#_Toc245121559)

[6.4.3. Crosby 109](#_Toc245121560)

[6.4.4. Feigenbaun 110](#_Toc245121561)

[6.4.5. Ishikawa 110](#_Toc245121562)

[6.5. Controle da Qualidade Total 112](#_Toc245121563)

[6.5.1. Apresentação do Controle da Qualidade Total 113](#_Toc245121564)

[6.5.2. Significado do Controle da Qualidade Total 114](#_Toc245121565)

[6.5.3. Princípios da Qualidade Total 115](#_Toc245121566)

[6.6. Ferramentas da Qualidade 118](#_Toc245121567)

[6.7. Gestão da Qualidade Total / Administração da Qualidade 122](#_Toc245121568)

[1.7.1. Gerenciamento por Processos 122](#_Toc245121569)

[1.7.2. Gerenciamento por Diretrizes 122](#_Toc245121570)

[1.7.3. Gerenciamento da Rotina 122](#_Toc245121571)

[1.8. Garantia da Qualidade 122](#_Toc245121572)

[1.9. Qualidade na Interface Compras/Vendas 124](#_Toc245121573)

[1.9.1. Qualidade nas Vendas 124](#_Toc245121574)

[1.9.2. Qualidade nas Compras 126](#_Toc245121575)

[1.10. Implantação do TQC 128](#_Toc245121576)

[6.9.1 Fundamentos 128](#_Toc245121577)

[6.9.2 Organização para implantação 128](#_Toc245121578)

[6.9.3 Sistema de gerenciamento da implantação do TQC 130](#_Toc245121579)

[6.10 Tópicos de pesquisa 131](#_Toc245121580)

[6.11. Sugestões de Leitura 131](#_Toc245121581)

[6.12. Exercícios 131](#_Toc245121582)

[6.13. Referências 131](#_Toc245121583)

[9.2. Organismos normativos 102](#_Toc245121584)

[9.2.1 ISO 103](#_Toc245121585)

[A primeira versão ISO 9000:1987 subdividia-se em modelos para qualidade, classificados da seguinte forma [MATOS, 2009]: 108](#_Toc245121586)

[ ISO 9001 : Modelo de garantia para qualidade de projeto, desenvolvimento, produção, montagem e fornecedores aplicando-se à organizações cujas atividades eram voltadas para criação de novos produtos. 108](#_Toc245121587)

[9.3.4. Norma ISO 9001 111](#_Toc245121588)

[Com o lançamento da ISO 9000, várias organizações despertaram a temática de que precisavam impor, e principalmente manter, padrões de qualidade em seu funcionamento, seja nos processos, ou mesmo nas pessoas que colaboram para o funcionamento das mesmas. O pensamento com uma melhor visão e ambição para o mercado dispõe da realização de investimentos que prestem alternativas viáveis para o crescimento e melhoramento das atividades. 111](#_Toc245121589)

[MELLO, Carlos Henrique Pereira et. al.(2009) descreve que as normas para sistemas de gestão, principalmente a ISO 9001, fornecem modelos básicos para que as organizações preparem e operem seus fluxos de funcionamento seguramente fortificados. O autor ainda cita que: 111](#_Toc245121590)

[“*As grandes organizações, ou aquelas com processos complexos, poderiam não funcionar bem sem um sistema de gestão, apesar de ele poder ter sido chamado por algum outro nome.”* 111](#_Toc245121591)

[A norma ISO 9001 foi instituída com esse propósito. Descrever os requisitos para possibilitar a implantação e administração de um modelo para garantia de qualidade para produtos e serviços através de um Sistema de Gestão de Qualidade. Como estratégia de negócios para apresentar uma base sólida de segurança e qualidade nas empresas, esta norma condiz um fator de certificação através de auditorias, inspeções, dentre outras atividades que classifiquem e garantam boa procedência para verificação e validação de processos e serviços conforme as terminologias e vocabulários apresentados pela ISO na versão 9000. 111](#_Toc245121592)

[9.4.1 Estrutura da norma: Processos de ciclo de vida 120](#_Toc245121593)

[9.4.2 Processos primários 120](#_Toc245121594)

[9.4.3 Processos de apoio 120](#_Toc245121595)

[9.4.4 Processos organizacionais 120](#_Toc245121596)

[9.5 ISO/IEC 15504 120](#_Toc245121597)

[9.5.1 Avaliação de processos 120](#_Toc245121598)

[9.5.2 Projeto SPICE 120](#_Toc245121599)

[9.5.3 Estrutura da norma: Referência de processos 120](#_Toc245121600)

[9.5.4 Dimensão de processos 120](#_Toc245121601)

[9.5.5 Dimensão de capacidade 120](#_Toc245121602)

[9.5.6 Níveis de capacidade 120](#_Toc245121603)

[9.6 Conclusões 120](#_Toc245121604)

[9.7 Tópicos de pesquisa 120](#_Toc245121605)

[9.8 Sugestões de leiura 120](#_Toc245121606)

[9.7 Exercícios 120](#_Toc245121607)

Introdução 125

Histórico 126

CMMI 127

Representações do Modelo CMMI 129

Representação por Estágios 130

Representação Contínua 131

Representação por Estágios x Contínua 133

Método de Avaliação do CMMI (SCAMPI) 134

3.3.2.1. Conceito Central 134

3.3.2.2. Parâmetros observados no SCAMPI 134

3.3.2.3. Prazo e Exigência de Pessoal 135

3.3.2.4. Características essenciais do Método de SCAMPI 135

3.3.2.5. Modos de Uso 135

3.3.2.6. Descrição do Método 136

MPS.BR 140

3.4.1. Representação do Modelo MPS 141

3.4.1.1. Nível G – Parcialmente Gerenciado 142

3.4.1.2. Nível F – Gerenciado 142

3.4.1.3. Nível E – Parcialmente Definido 142

3.4.1.4. Nível D – Largamente Definido 143

3.4.1.5. Nível C – Definido 143

3.4.1.6. Nível B – Gerenciado Quantitativamente 144

3.4.1.7. Nível A – Em Otimização 144

3.4.2. Método de Avaliação do MPS.BR (MA-MPS) 144

3.4.2.1. Prazo e Exigência de Pessoal 145

3.4.2.2. Descrição do Método 146

CMMI x MPS.BR 149

Exercícios 150

Sugestões de Leitura 151

Tópicos de Pesquisa 151

Referências 152

[Introdução a modelos para melhoria de processos de software 154](#_Toc245121608)

[IDEAL 155](#_Toc245121609)

[ Fases do IDEAL 157](#_Toc245121610)

[ Fase inicial (Initiating) 158](#_Toc245121611)

[ Fase de diagnóstico (Diagnosing) 160](#_Toc245121612)

[ Fase de estabilização (Diagnosing) 162](#_Toc245121613)

[ Fase de ação (Acting) 164](#_Toc245121614)

[ Fase de aproveitamento (Leveraging) 166](#_Toc245121615)

[ Fase de gerenciamento do programa de melhoria do processo de software (Manage) 168](#_Toc245121616)

[PRO2PI 170](#_Toc245121617)

[ Engenharia de processo dirigida por perfis de capacidade e seus fundamentos 171](#_Toc245121618)

[ O PRO2PI 172](#_Toc245121619)

[ PRO2PI-PROP: Propriedades de PRO2PI 177](#_Toc245121620)

[ PRO2PI-MODEL: Modelo de PRO2PI 178](#_Toc245121621)

[ PRO2PI-MEAS: Medições para PRO2PI 180](#_Toc245121622)

[ PRO2PI-CYCLE: Processo para ciclo de melhoria 181](#_Toc245121623)

[Seis Sigma 182](#_Toc245121624)

[ PDCA 185](#_Toc245121625)

[ DMAIC 185](#_Toc245121626)

[ Definir 186](#_Toc245121627)

[ Medição 186](#_Toc245121628)

[ Análise 187](#_Toc245121629)

[ Melhoria 187](#_Toc245121630)

[ Controle 187](#_Toc245121631)

[Considerações Finais 188](#_Toc245121632)

[Exercícios 189](#_Toc245121633)

[Sugestões de leitura 190](#_Toc245121634)

[Para entender melhor o que é Melhoria de processo de software lei a norma ISO/IEC 15504-4/2004. 190](#_Toc245121635)

[Para um estudo detalhado sobre o PRO2PI leia tese de doutorado de Clênio Salviano, “Uma Proposta Orientada a Perfis de Capacidade de Processo para Evolução da Melhoria de Processo de Software”. Disponível em: http://libdigi.unicamp.br/document/?code=vtls000380495 190](#_Toc245121636)

[Para um estudo detalhado sobre IDEAL leia o guia oficial de implantação produzido pelo SEI, “IDEAL - A User's Guide for Software process Improvement”. Disponível em: http://www.sei.cmu.edu/library/abstracts/reports/96hb001.cfm 190](#_Toc245121637)

[Tópicos de pesquisa 190](#_Toc245121638)

[Referências 191](#_Toc245121639)

1.1. Modelos de qualidade de produto 198

Os modelos de qualidade objetivam avaliar o produto de software, segundo diferentes aspectos baseados na visão do usuário. Para padronizar internacionalmente as características de implementação do software, foram criadas algumas normas que serão vistas a seguir. 198

1.1.1. ISO 9126 198

1.1.1.1. Diretrizes para uso da norma NBR ISO/IEC 9126-1 198

1.1.1.2. Características e sub-características de qualidade de software 199

1.1.2. ISO 12119 201

1.1.3. ISO 14598 203

É um guia para avaliação de produtos de software, baseado na utilização prática da norma ISO 9126, já que esta define as métricas, características e subcaracterísticas de qualidade de software [Koscianski & Soares, 2007]. 203

1.1.4. Projeto SQuaRE 205

1.1.5. Norma SQuaRE 205

Na reorganização das antigas normas 9126 e 14598, o projeto SQuaRE adotou uma divisão de assuntos em cinco tópicos que aparecem na Figura 1.3: 205

1.2. Teste de Software 207

1.2.1. Abordagens de Testes 208

Existem duas abordagens principais de testes: abordagem funcional (“black box” ou “caixa preta”) e abordagem estrutural (“White box” ou “caixa branca”). 208

 Caixa preta: como o próprio nome já sugere, nesta abordagem o testador visualiza o software como uma caixa preta, ou seja, não considera a estrutura interna do programa, de que forma o código foi implementado ou que tecnologia foi utilizada, por exemplo. Considerando os dados de entrada, o objetivo principal é observar as saídas geradas pelo sistema e verificar se estas estão de acordo com o esperado. A Figura 1.4 ilustra este tipo de abordagem. 208

 Caixa branca: diferentemente da abordagem anterior, neste tipo de abordagem o testador está interessado no que está acontecendo “dentro da caixa”. É caracterizada por avaliar as funcionalidades internas dos componentes do software, baseando-se no código fonte e procurando exercitar estruturas de controle e de dados do programa. Sendo assim, faz-se necessário que o analista de testes tenha boa habilidade em programação de modo a entender todos os caminhos lógicos possíveis. A Figura 1.5 ilustra a abordagem estrutural. 208

1.2.2. Estágios de Testes 208

Os testes de software normalmente são executados em diferentes estágios durante o ciclo de vida do desenvolvimento do software. Dependendo do objetivo principal do teste, quatro estágios foram definidos: 208

 Teste de unidade: realiza testes em componentes individuais (módulos, programas, objetos, classes, etc) de forma a determinar se cada um deles, separadamente, está sendo executado de maneira correta. Normalmente estes testes são testes de caixa branca realizados pelos próprios desenvolvedores do componente. Geralmente utilizam ferramentas que provêem um suporte adicional para checar a corretude do programa, como ferramenta de *debbuging* ou *framework* para teste unitário, por exemplo. Os defeitos encontrados neste estágio são normalmente corrigidos de imediato, sem a necessidade de documentá-los formalmente, e assim, reduzindo o custo, pois antecipa a correção de defeitos. Geralmente é necessária a utilização de *stubs* (módulos que substituem outros módulos subordinados) e *drivers* (um módulo que substitui outro módulo que seja responsável por controlar a chamada de um sistema)*,* para serem utilizados no lugar dos softwares que estejam eventualmente faltando e para simular a interface entre os componentes de software. 208

 Teste de integração: nesta etapa, as unidades que foram testadas individualmente no estágio anterior são testadas de forma integrada, bem como as interfaces entre os componentes. A integração deve ser realizada adicionando-se os componentes um por um, e após cada passo um teste é necessário (teste incremental). Esta técnica tem a vantagem de achar defeitos o mais cedo possível no processo de testes e corrigi-los mais rapidamente, enquanto é mais fácil determinar as causas dos erros. Por outro lado, tem a desvantagem de ser uma prática bastante custosa. Sendo assim, a integração pode ser feita basicamente de duas formas: *Top-down* ou *Bottom-up*. Na primeira, os testes são realizados de cima para baixo (começando da GUI ou do menu principal); componentes ou sistemas são substituídos por *stubs*. Na segunda, os testes começam na parte mais básica do sistema até o nível mais alto; componentes ou sistemas são substituídos por *drivers*. 209

 Teste de sistema: neste nível o propósito do teste está em verificar o funcionamento de todo o sistema, já integrado, e analisar se ele está de acordo com os requisitos que foram especificados. Neste momento não só é testada a integração dos componentes do software entre si, mas também destes com um ambiente de teste correspondente à produção final (hardware, software, outros sistemas), de modo a minimizar o risco de que falhas relacionadas com o ambiente operacional do produto não sejam encontradas. Geralmente a estratégia de caixa preta é utilizada neste estágio, mas testes de caixa branca também podem ser realizados. 209

 Teste de aceitação: o teste de aceitação corresponde ao teste realizado pelo usuário de fato do sistema, no momento em que todos ou quase todos os defeitos encontrados nas etapas anteriores já tenham sido corrigidos. O propósito deste teste é estabelecer a confiança do sistema; ele está mais relacionado com a validação do sistema, em que está se tentando determinar se o sistema faz aquilo que é suposto fazer. Normalmente os testes de aceitação podem ser de duas categorias: testes *alfa* e testes *beta*. Os primeiros são realizados nas instalações do desenvolvedor, que fica observando os usuários utilizarem o sistema, e anotam os problemas identificados. Já os testes *beta* são realizados no ambiente real de trabalho do usuário, que instala o sistema e testa, sem a presença do desenvolvedor. Em seguida, um documento contendo os registros dos problemas encontrados é enviado à organização desenvolvedora. 209

1.2.3. Tipos de Testes 209

Cada tipo de teste é focado em um grupo de atividades com um determinado objetivo de teste. É necessário pensar em diferentes tipos de testes uma vez que testar a funcionalidade de um componente ou sistema pode não ser suficiente em cada um dos estágios envolvidos para se chegar aos objetivos dos testes. Um tipo de teste é focado num objetivo particular de teste, que poderia ser um teste de uma função a ser executada pelo componente ou sistema; alguma característica não funcional; a estrutura ou arquitetura do componente ou sistema, etc. Existem vários tipos de testes, dependendo do objetivo de cada projeto e de cada organização. Abaixo serão apresentados alguns dos mais comuns. 209

 Teste funcional – este tipo de teste está focado nas regras de negócio do sistema, ou seja, o fluxo de trabalho do programa é avaliado. 210

 Teste de recuperação de falha – o sistema é forçado a falhar de diversas maneiras de modo a verificar seu comportamento diante destas falhas, e reparar de que formas ele se recupera. 210

 Teste de interoperabilidade – testa um produto de software de modo a determinar sua capacidade de interagir com um ou mais componentes ou sistemas. 210

 Teste de segurança – verifica se o sistema possui atributos para prevenir acessos não autorizados, acidentais ou propositais, a programas e dados. 210

 Teste de carga – um tipo de teste para medir o comportamento do sistema quando este é submetido a níveis altos de carga, diferente das condições normais. É importante determinar o quanto de carga o sistema consegue suportar sem falhar. 210

 Teste de performance – verifica o rendimento de um sistema, como o tempo de resposta e processamento, taxa de transferência de dados, para diferentes condições (configurações, numero de usuários, etc) as quais o programa é submetido. 210

 Teste de estresse – teste conduzido para avaliar o comportamento do sistema diante de condições que ultrapassem o limite especificado nos requisitos. 210

 Teste de configuração – testa o funcionamento do sistema em diferentes configurações de hardware/software, testando compatibilidade, configuração do servidor, tipos de conexões com a Internet, etc. 210

 Teste de usabilidade – testes para determinar se um produto é facilmente entendível, fácil de aprender, fácil de operar e atrativo aos usuários, ou seja, se o produto tem uma interface amigável para os que utilizarão o sistema. 210

 Teste de regressão – teste de regressão é a atividade de testar uma nova versão de um sistema para validar esta versão, detectando se erros foram introduzidos devido às mudanças realizadas no software, e então, garantir a corretude das modificações. Uma vez que a re-execução de todos os testes é uma atividade bastante custosa, uma seleção de testes de regressão geralmente é realizada. 210

1.2.4. Processo de testes 210

Segundo [Graham et. al 2007], o processo de testes pode ser dividido basicamente em cinco etapas: *planejamento e controle, análise e projeto, implementação e execução, avaliação de critério de saída e reportagem* e *atividades de encerramento de testes*. Estas atividades são logicamente seqüenciais, porém, em um projeto específico, podem se sobrepor, serem executadas em paralelo ou até mesmo serem repetidas. 210

1.2.4.1. Planejamento e Controle 211

Durante o planejamento de testes deve-se ter certeza de que os objetivos dos clientes e *stakeholders* foram entendidos de maneira correta. Baseados neste entendimento, os propósitos da atividade de testes propriamente dita são estabelecidos, e assim, uma abordagem e plano para os testes é obtida incluindo especificação das atividades de teste. O planejamento de testes apresenta as seguintes atividades principais: 211

 Determinar o escopo e riscos e identificar os objetivos de teste: são determinados os softwares, componentes, sistemas ou outros produtos que devem ser testados; os riscos que devem ser levados em consideração; e qual o propósito do teste (encontrar defeitos, verificar se está de acordo com os requisitos ou dentro dos padrões de qualidade, etc). 211

 Determinar a estratégia de teste: aqui serão estabelecidas as técnicas que serão utilizadas, o que precisa de fato ser testado (selecionar e priorizar os requisitos) e que nível de cobertura é necessário. Serão também analisadas quais pessoas precisarão se envolver e em que momento (desenvolvedores, usuários, etc), incluindo a definição da equipe de teste. 211

 Definir recursos: são definidos todos os recursos necessários durante o ciclo de vida de testes, tanto recursos materiais (PCs, software, ferramentas, etc) como recursos humanos (principais e de apoio). 211

 Fazer um cronograma para análise e projeto, implementação, execução e avaliação de teste: deverá ser elaborado um cronograma de todas as tarefas e atividades, para que seja possível terminar a fase de testes a tempo. 211

 Estabelecer os critérios de saída: critérios de saída, como critério de cobertura, por exemplo, deverão ser estabelecidos de modo a determinar quando a etapa de testes chegou ao fim. 211

Após planejar é necessária uma medida de controle para verificar se tudo está indo de acordo com o planejado. É preciso comparar o andamento real com o que foi estabelecido no plano de testes, e tomar medidas corretivas quando necessário. 211

1.2.4.2. Análise e Projeto 211

Esta é a atividade em que os objetivos gerais de testes são transformados em condições e projetos de teste tangíveis. O propósito principal é identificar e descrever os casos de teste para cada versão de teste, e identificar e estruturar os procedimentos de teste, especificando como executar os casos de teste. As principais tarefas desta etapa podem ser destacadas em: 211

 Revisar a base de testes (como a análise de risco do produto, requisitos, arquitetura, especificação de projeto, e interfaces): a base de testes é utilizada para criar os testes. É possível começar a projetar os testes de caixa preta antes da implementaço, uma vez que a base de testes pode ser usada para compreender o que o sistema precisa fazer. 211

 Identificar e descrever casos de teste: um caso de teste é um cenário associado a um requisito; é um texto contendo: identificador, objetivo, pré-condições de execução, entradas, passos específicos do teste a ser executado e resultados esperados e/ou pós-condições de execução. Um caso de teste bem projetado tem muita chance de encontrar um erro ainda não conhecido. 212

 Estruturar procedimentos de teste: o passo a passo que descreve como os casos de teste devem ser executados. Inclui o estado inicial da aplicação; condições de funcionamento; como e quando fornecer os dados de entrada e obter os resultados; a forma de avaliar estes resultados, dentre outros. 212

 Avaliar a capacidade de testar os requisitos: a especificação de requisitos deve ser completamente clara, informando as condições necessárias para se definir os testes. Por exemplo, se a performance do software é algo crítico, deve ser claramente especificado o tempo de resposta mínimo em que o sistema deve responder. 212

1.2.4.3. Implementação e Execução 212

Uma vez que os casos e procedimentos de teste foram especificados em alto nível na etapa anterior, este é o momento em que o ambiente será preparado para que eles sejam executados e comparados com os resultados desejados. Além disso, é a etapa em que os componentes necessários são implementados para que os testes sejam executados. As principais tarefas destas duas fases serão destacadas a seguir. 212

 Implementação: 212

o Implementar componentes: efetuar a implementação de novos componentes de apoio necessários à aplicação dos testes, ou modificação de componentes já existentes. Ferramentas de automação podem ser utilizadas ou os componentes podem ser desenvolvidos explicitamente. 212

o Criar suítes de teste: baseado nos casos de teste, um conjunto de testes que naturalmente trabalham juntos, forma uma suíte de teste e são utilizados para uma execução de teste eficiente. 212

o Implementar e verificar o ambiente: preparar e verificar se o ambiente de teste está funcionando corretamente. 212

 Execução: 212

o Executar as suítes de teste e casos de teste individuais, de acordo com os procedimentos de teste. Pode ser feito manualmente ou com o auxílio de ferramentas de execução de testes. 212

o Seguir as estratégias de teste definidas na etapa de planejamento. 212

o Criar um *log* com as saídas da execução dos testes e registrar os identificadores e versões do software que está sendo testado. 212

o Fazer a comparação dos resultados esperados e dos resultados obtidos. 212

o Quando houver diferenças entre os resultados esperados e os resultados obtidos, registrar os defeitos em um repositório centralizado. Não se deve registrá-los de forma aleatória. 212

o Realização de testes de regressão para confirmar que uma falha anteriormente registrada foi de fato consertada. 213

1.2.4.4. Avaliação do critério de saída e relatório 213

Esta é a fase em que se deseja observar se já foram executados testes suficientes para garantir a qualidade desejada do produto, sendo assim, critérios de saída são definidos com esta finalidade. Estes critérios informam se uma dada atividade de testes pode ser considerada completa. As principais atividades são: 213

 Checar se os *logs* de testes batem com os critérios de saída especificados no plano de testes: procura-se pelos testes que tenham sido executados e avaliados, e se defeitos foram encontrados, consertados ou re-testados. 213

 Verificar se será necessária a inclusão de mais testes ou se os critérios de saída especificados devem ser mudados: mais casos de testes podem precisar ser executados, se por acaso estes não tiverem sido todos executados conforme esperado, ou se for detectado que a cobertura de requisitos necessária ainda não foi atingida, ou até mesmo se aumentaram os riscos do projeto. 213

 Escrever um relatório de resumo de testes para os *stakeholders*: todos os *stakeholders* devem saber quais testes foram executados e quais os resultados destes testes, de modo a perceber que decisões precisam ainda ser tomadas visando a melhoria da qualidade do software. 213

1.2.4.5. Atividades de encerramento de teste 213

A atividade de encerramento de teste pode ser dada através de diversos fatores, como por exemplo, as informações necessárias do processo de testes já foram atingidas; o projeto é cancelado; quando um marco particular é alcançado; ou quando uma versão de manutenção ou atualização está concluída. As atividades principais são: 213

 Checar se as entregas que foram programadas foram de fato entregues e garantir que todos os problemas reportados foram realmente resolvidos. Para os que permaneceram em aberto devem-se requisitar mudanças em uma futura versão. 213

 Finalizar e arquivar os artefatos produzidos durante o processo necessário para planejar, projetar e executar testes, como por exemplo, documentação, scripts, entradas, resultados esperados, etc. É importante reutilizar tudo que for possível destes artefatos, pois assim se consegue economizar tempo e esforço do projeto. 213

 Repassar os artefatos anteriormente citados para a equipe de manutenção, que irá prover suporte aos usuários do sistema e resolver qualquer problema encontrado depois de sua entrega. 213

 Avaliar como se deu o processo de testes e analisar as lições aprendidas, que serão de grande utilidade para futuras versões dos projetos. Este passo pode permitir não só melhorias no processo de testes, como também melhorias no processo de desenvolvimento do software como um todo. 213

1.2.5. Testes ao longo do ciclo de vida de Software 213

As atividades de teste não são atividades que são realizadas sozinhas, mas sim em paralelo com o ciclo de vida de desenvolvimento do software. Dessa forma, a escolha do ciclo de vida do projeto irá afetar diretamente as atividades de teste. O processo de desenvolvimento adotado depende muito dos objetivos e propósitos do projeto. Portanto, o modo como as atividades de teste são estruturadas deve se ajustar ao modelo de desenvolvimento de software, ou do contrario, não conseguirá obter o sucesso desejado. 213

Um modelo de desenvolvimento de software bastante conhecido é o modelo em cascata, que como o próprio nome já sugere, tem sua base voltada a um desenvolvimento seqüencial das atividades. As primeiras atividades começam no topo da cascata, e então vão seguindo seqüencialmente através das várias atividades de concepção do projeto, e finalmente terminando com a etapa de implementação. Após isso, é que as atividades de teste são introduzidas, e dessa forma os defeitos só podem ser detectados bem perto da fase de implementação. A Figura 1.6 ilustra o modelo em cascata. 214

Com o objetivo de tentar contornar os problemas do modelo em cascata, foi desenvolvido o modelo V, que foca nos testes do produto durante todo o ciclo de desenvolvimento para conseguir uma detecção adiantada de defeitos. A idéia é que as atividades de testes não são simplesmente uma fase única, mas pelo contrário, como já foi visto na sessão anterior, se faz necessária toda uma preparação, passando por etapas de planejamento, análise, projeto, etc, que devem ser executadas em paralelo com as atividades de desenvolvimento. 215

Todos os artefatos gerados pelos desenvolvedores e analistas de negócio durante o desenvolvimento, provêem a base de testes em um ou mais níveis. Promovendo as atividades de teste mais cedo, defeitos podem ser geralmente encontrados nos documentos da base de testes. O modelo V demonstra como as atividades de verificação e validação podem ser executadas em conjunto com cada fase do ciclo de vida de desenvolvimento do software. 215

1.3. Inspeção de Software 216

Como explicado na sessão anterior, a inspeção de software é uma técnica estática do processo de V & V, em que são efetuadas revisões no sistema com o objetivo de encontrar defeitos e então, corrigi-los. O objetivo principal das inspeções é garantir que defeitos sejam reparados o mais cedo possível no processo de desenvolvimento de software, uma vez que quanto mais tarde, mais difícil de se encontrar os erros e mais custoso ainda consertá-los. Qualquer artefato produzido no desenvolvimento do software pode ser utilizado no processo de inspeção, como requisitos, modelo de projeto ou código. 216

O modelo CMMI exige a realização de revisões como uma prática específica do processo de verificação, demonstrando assim sua importância na garantia da qualidade do produto. Segundo Fagan, [Fagan 1986] a utilização de inspeções informais de software capturam em torno de 60% dos erros em um programa. Mills et al. [Mills et al. 1987] sugere que uma aplicação mais formal de inspeção de software pode detectar até mais de 90% dos erros de um programa. Selby e Basili [Selby et al. 1987] comparam empiricamente a efetividade de inspeções e testes. Eles perceberam que a revisão de código estática se mostrava mais efetiva e menos cara do que a procura por erros utilizando testes. 216

1.3.1. A Equipe de Inspeção (Participantes) 217

A equipe de inspeção é composta por um pequeno grupo de pessoas que possuam interesse e conhecimento do produto. Geralmente o tamanho da equipe varia de quatro a sete participantes, e o número mínimo é de três pessoas. Equipes maiores são normalmente utilizadas para analisar documentos de mais alto nível do produto, enquanto que times menores são preferíveis ao se inspecionar detalhes mais técnicos. Uma boa variedade de inspetores, representando diferentes áreas de conhecimento, é interessante para o processo de inspeção. O papel de cada participante será explicado abaixo. 217

 Autor – é o criador (desenvolvedor) do artefato que será inspecionado. Suas principais responsabilidades são: corrigir os problemas detectados durante o processo de inspeção, prover uma visão geral do produto aos demais participantes e tirar quaisquer dúvidas que surgirem com relação ao artefato desenvolvido. 217

 Inspetor – examina o produto antes da reunião de inspeção (fase de preparação) e durante de modo a tentar encontrar defeitos. Pode também identificar problemas amplos que estão fora do escopo da equipe de inspeção, como também sugerir melhorias. 217

 Leitor – pessoa responsável por apresentar o artefato aos demais participantes do processo de inspeção durante a reunião. Uma pessoa que usará o produto numa próxima etapa do seu ciclo de vida é um candidato forte pare esta tarefa, uma vez que a atividade de ler sobre o produto irá permitir a este potencial usuário se tornar bastante familiar com o produto. 217

 Escritor – tem o papel de registrar as informações sobre cada defeito encontrado durante a reunião, que incluem: a localização do defeito, um resumo do problema, sua classificação e uma identificação do inspetor que o encontrou. Todas as decisões e recomendações feitas também são registradas. 217

 Moderador – o moderador tem o papel mais crítico no processo de inspeção e por este motivo faz-se necessário um treinamento mais aprofundado do que os outros membros da equipe. Ele é a pessoa que lidera toda a equipe e participa ativamente de todas as etapas. Dentre suas principais responsabilidades podemos destacar: selecionar e liderar a equipe de inspeção, distribuir o material a ser inspecionado, agendar as reuniões, atuar como moderador nos encontros, supervisionar a correção dos defeitos, e emitir relatório de inspeção. Uma outra responsabilidade muito importante do moderador é garantir que o foco da reunião se mantenha em encontrar falhas no produto, e não em acusar o autor dos problemas encontrados. 217

1.3.2. O Processo de Inspeção de Software (Etapas) 217

O processo tradicional de inspeção de software [Fagan 1976] é definido por seis estágios, cada um representado por seu principal responsável. A Figura 1.8 ilustra esta seqüência de etapas e em seguida cada uma das etapas será explicada detalhadamente. 217

 Planejamento: o moderador é a pessoa responsável por esta etapa. O planejamento envolve selecionar a equipe, checar se o produto está pronto para inspeção, organizar a reunião, delegar as atividades de cada membro e garantir a completude dos materiais a serem inspecionados. Nesta etapa o moderador também deve verificar se o material a ser inspecionado possui um tamanho adequado para uma única reunião. Caso contrario, o material deverá ser dividido em tamanhos menores, com inspeções a serem realizadas para cada uma destas partes. 219

 Visão geral: nesta etapa o autor apresenta o produto aos demais membros da equipe, descrevendo o que o programa é suposto fazer. O moderador é responsável por decidir se esta etapa se faz realmente necessária, pois se a equipe já for bem familiarizada com o material a ser inspecionado ou novas técnicas não estejam sendo aplicadas, este estágio é dispensável. 219

 Preparação: este é o momento em que cada membro do time de inspeção estuda individualmente a especificação e o programa a ser inspecionado, e procura por defeitos no material. Todos os possíveis defeitos devem ser registrados num *log* de preparação, assim como o tempo que foi gasto na preparação. O moderador é encarregado de analisar os *logs* antes da reunião de inspeção para determinar se a equipe está preparada para suas tarefas, e caso contrario, ele pode remarcar a reunião. 219

 Reunião: nesta etapa, o passo a passo principal consiste na leitura e interpretação do produto, pelo leitor; em seguida o autor tira quaisquer dúvidas que eventualmente surgirem com relação ao material, e a equipe de inspetores então identificam os possíveis defeitos. Esta reunião deve ser curta, não podendo passar mais do que duas horas, e deve ser focada na detecção de defeitos, conformidade com o padrão e programação de má qualidade. O time de inspeção não deve discutir como estes defeitos poderiam ser corrigidos e nem sugerir mudanças em outros componentes. 219

 Re-trabalho: o propósito do re-trabalho é corrigir os defeitos identificados durante a reunião de inspeção. O autor é a pessoa responsável por essas correções, devendo corrigir em primeiro lugar os defeitos considerados mais relevantes e graves, e corrigindo os de menor importância apenas se o tempo permitir. 219

 Acompanhamento: aqui o moderador deve decidir se uma nova inspeção é necessária ou não. Ele deve analisar o material corrigido pelos autores e verificar se os defeitos foram corrigidos com sucesso. O moderador pode incluir revisores adicionais nesta etapa se forem necessários conhecimentos técnicos extras. Se todos os problemas mais relevantes forem resolvidos, todos os problemas em aberto solucionados, e o produto satisfizer aos critérios de saída, o moderador aprova o *release* do produto. Se as condições não foram atingidas, ainda será necessário mais um tempo na etapa de re-trabalho. 219

1.3.3. Ferramentas de Apoio ao Processo de Inspeção 219

Baseado na classificação de *groupware* (softwares voltados para o apoio a atividades de trabalho em grupo) e nas constantes mudanças tecnológicas, [Hedberg 2004] identificou quatro gerações de ferramentas de inspeção de software: 220

1. Primeiras Ferramentas (*Early tools*) 220

2. Ferramentas Distribuídas (*Distributed tools*) 220

3. Ferramentas Assíncronas (Asynchrounous tools) 220

4. Ferramentas baseadas em WEB (Web-bsed tools) 220

Nota-se que as primeiras ferramentas a surgirem foram classificadas como Primeiras Ferramentas, no início da década de 90 e logo em seguida vieram as Ferramentas Distribuídas. No fim da década de 90 surgiram as ferramentas para Internet. 220

As ferramentas da primeira geração são aquelas que apenas permitem o trabalho de toda a equipe no mesmo ambiente e ao mesmo tempo (inspeções síncronas). A segunda já permite que a equipe possa trabalhar de forma distribuída, ou seja, em lugares diferentes, porém ainda é preciso que seja ao mesmo tempo (inspeções distribuídas). A total independência de tempo e lugar foi introduzida na terceira geração, com as ferramentas assíncronas. As ferramentas da quarta geração também são assíncronas, diferenciando-se das demais devido a sua base tecnológica. 220

A seguir será apresentada uma ferramenta representante de cada geração introduzida anteriormente. A ferramenta ICICLE representará a geração de Primeiras Ferramentas. Em seguida a ferramenta Scrutiny exemplificará as Ferramentas Distribuídas. Assist ilustrará as Ferramentas Assíncronas, e finalmente, IBIS será a representante das Ferramentas baseadas em WEB. 220

 ICICLE – O ICICLE (*Intelligent Code Inspection Environment* *in a C Language* *Environment*) é o primeiro software de revisão publicado e visa apoiar o processo tradicional de inspeção de software. Como o próprio nome já sugere, ele foi desenvolvido para o contexto específico de inspeção de código C e C++, podendo ser usado para o auxílio da inspeção do código, tanto nas fases de preparação individual como nas reuniões em grupo. A reunião de inspeção em grupo deve ser realizada no mesmo local e a inspeção individual permite entrar com comentários em cada linha de código. A ferramenta não se aplica a inspeções mais genéricas, limitando o tipo de artefato a ser inspecionado e a técnica de detecção de defeitos, mas pode, entretanto, ser utilizada para inspecionar linhas de texto numa análise inicial. Um dos principais objetivos desta ferramenta é o de ajudar os inspetores de código a encontrarem defeitos óbvios. 220

 Scrutiny – O Scrutiny é uma ferramenta colaborativa *online*, sendo a primeira a permitir que os membros do time de inspeção se encontrassem dispersos geograficamente, podendo ser usada tanto de forma síncrona como assíncrona. Ela pode ser integrada com outras ferramentas e customizada para apoiar diferentes processos de desenvolvimento. Atualmente apenas suporta inspeções de textos. A ferramenta é baseada num processo de inspeção dividido em quatro etapas. No primeiro estágio, de iniciação, o moderador disponibiliza o documento a ser inspecionado na ferramenta. No próximo estágio, preparação, os inspetores inserem seus comentários a serem discutidos na reunião. Depois, na fase de resolução, o moderador guia os inspetores através dos documentos e dos defeitos coletados. Finalmente, no estágio de finalização, após as discussões e acordos referentes aos defeitos levantados, a ferramenta fornece um resumo dos defeitos que foram discutidos. 220

 Assist – *Asynchronous/ Synchronous Software Inspection Support Tool* foi desenvolvida para prover inspeções individuais e em grupo. Como o nome sugere, permite inspeções síncronas e assíncronas, com reuniões tanto em locais diferentes como no mesmo ambiente. Utiliza uma linguagem de definição de processo de inspeção (IPDL) e um sistema flexível para o tipo de documento inspecionado, permitindo o suporte a qualquer tipo de processo de inspeção de software. Inspeção de código, coletas de dados para métricas e cálculos para apoio as inspeções também estão presentes nesta ferramenta. É baseada numa arquitetura cliente/servidor, em que o servidor é usado como um repositório central de documentos e outros tipos de dado. Um *browser C++* pode automaticamente apresentar itens relevantes de *checklist* para a sessão de código inspecionado. 221

 IBIS – *Internet-Based Inspection System* é uma ferramenta baseada em WEB com notificações por *email* que auxilia no processo de inspeção desenvolvido por Fagan. Permite que as inspeções sejam realizadas entre pessoas geograficamente distribuídas e possui uma interface bastante leve e amigável, tendo toda sua estrutura e dados armazenados em arquivos XML. Ela não limita o tipo de artefato a ser inspecionado e provê suporte a decisões, apoio a anotações e *checklists*. As principais vantagens desta ferramenta são: por ser baseada em WEB, permite que os inspetores acessem a aplicação de seus próprios computadores; permite que a inspeção seja realizada com integrantes da equipe distribuídos em locais diferentes, até mesmo em países diferentes; permite que especialistas diferentes participem da reunião, podendo ser especialistas de outro departamento ou mesmo fora na organização. 221

1.4. Modelos de Maturidade de Testes de Software 221

Para se construir software com qualidade, é necessário que se tenha um processo de testes bem definido e que ele esteja alinhado ao processo de desenvolvimento. Nesta seção serão vistos três modelos de maturidade de teste de software, os quais indicam como criar e/ou melhorar o processo de testes. 221

1.4.1. Processo de melhoria de testes – TPI 221

1.4.1.1. Escopo do TPI 222

1.4.1.2. Áreas Chave 223

1.4.1.3 Passos para implantar a melhoria 224

1.4.2. TMM – Test Maturity Model 225

1.4.2.1. Níveis de Maturidade do TMM 226

1.4.3. TIM – Test Improvement Model 227

1.4.3.1. Modelo de Maturidade 227

1.4.3.2. Áreas Chave 229

Considerações Finais 232

O desenvolvimento de software engloba um mercado de extrema competitividade. Tendo em vista que os sistemas que apresentam melhor qualidade garantem seu espaço no mercado, as empresas que os desenvolvem têm investido bastante esforço para assegurar a qualidade de seus produtos e garantir a satisfação dos clientes. A qualidade de um produto pode ser definida como sua capacidade de cumprir os requisitos inicialmente estipulados pelos clientes, e sendo assim, está diretamente relacionada à qualidade do processo de desenvolvimento. Por este motivo, tem surgido uma grande demanda ao incentivo de pesquisas que levem em consideração à procura por formas de melhoria da qualidade dos produtos. 232

Este capítulo procurou introduzir ao leitor boas práticas no que diz respeito à qualidade dos produtos, apresentando um conjunto de normas que representam a padronização mundial para avaliação da qualidade de produtos de software. As atividades de teste e inspeção também foram destacadas como forma de encontrar defeitos no software e corrigi-los a tempo, antes de entregar o produto a seus clientes, e analisar se o sistema faz o que é suposto fazer. Finalmente, modelos de maturidade de testes foram apresentados como mais uma tentativa de atingir os objetivos desejados, buscando melhorias na qualidade do processo de teste de software, que afeta diretamente a qualidade do produto. 232

Exercícios 233

1. Quais são as diretrizes para uso da norma NBR ISO/IEC 9126-1? 233

2. A que se propõe a norma ISO 12119? 233

3. Que subdivisões da norma ISO 14598 estabelecem itens necessários para o suporte à avaliação? 233

4. Quais são os componentes do projeto SQuaRE? Defina-os. 233

5. Qual a diferença entre testes e inspeções de software? 233

6. Cite 5 tipos de testes e explique cada um deles. 233

7. Quais os estágios de testes possíveis e quais as características de cada um deles? 233

8. O que são testes beta? 233

9. O que são testes de regressão? 233

10. Qual a diferença entra a abordagem de caixa preta e a abordagem de caixa branca? 233

11. Quais são os papéis existentes na equipe de inspeção de software e quais suas responsabilidades? 233

12. Quais são as etapas do processo de inspeção de software? Explique cada uma delas. 233

13. Explique como é feita a implantação da melhoria no TPI. 233

14. Defina os níveis de maturidade do TMM. 233

15. No aspecto Organização, como são caracterizados os níveis de maturidade do TIM? 233

Sugestões de Leitura 234

Para conhecer mais sobre normas de qualidade de produto de software, é recomendada a leitura do livro Tecnologia da Informação: Qualidade de Produto de Software, Guerra & Colombo 2009. 234

Para ampliar o entendimento sobre o assunto de teste de software é recomendada a leitura do livro *Foundations of software testing*, Graham, D., Veenendaal, E. v., Evans, I. and Black, R., 2007. Este livro é utilizado por pessoas que desejam tirar o certificado ISTQB (*International Software Testing Qualifications Board)*, portanto, é muito interessante para adquirir melhores conhecimentos sobre este conteúdo. 234

Para um melhor conhecimento sobre os conceitos e o processo de inspeção de software é sugerida a leitura de *Design and Code Inspection to Reduce Errors in Program Development*, Fagan, M.E.,1976. 234

Para se aprofundar mais sobre as ferramentas de inspeção de software é recomendada a leitura de *Modern Software Review Techniques and Technologies*, Wong, Y. K., 2006. 234

Para melhor conhecimento sobre o TPI (*Test Process Improvement*) é recomendada a leitura do livro *Test Process Improvement A practical step-by-step guide to structured testing*, Koomen & Pol, 1999. 234

Para aprofundar a leitura sobre TMM (*Test Maturity Model*), é sugerida a leitura do livro *A Model to Assess Testing Process Maturity*, Burnstein & Grom, 1998. Tópicos de Pesquisa 234

Tópicos de Pesquisa 235

O TMM (*Testing Maturity Mode*), como visto anteriormente, foi desenvolvido pelo *Illinois Instituty of Tecnology* como um guia para melhoria de processos de testes. Existe uma organização sem fins lucrativos, em Dublin – Irlanda, que foi fundada para tentar transformar o modelo TMM em uma norma e, consequentemente, promover a sua aceitação como um padrão da indústria internacional de avaliação e de organizações de teste de software. 235

A Fundação TMMi tem os seguintes objetivos: 235

 Definição de um núcleo TMMi para a padronização internacional 235

 Criação e gestão de uma organização independente, imparcial com repositório central de dados e prestação de serviços 235

 Métodos de Avaliação com base no modelo padrão 235

 Prestação de um mecanismo independente para facilitar a verificação formal de avaliações TMMi 235

 Definição e manutenção de avaliador independente 235

 Prestação de um fórum público das partes interessadas para facilitar a livre troca de informação, educação, idéias e uso da norma pública. 235

O site oficial da fundação é http://www.tmmifoundation.org. 235

Existem vários estudos atualmente na academia no que diz respeito à seleção de testes de regressão, uma vez que executar todos os casos de teste novamente sempre que uma nova versão do sistema for liberada é uma prática inviável. Dessa forma, várias pesquisas e propostas de soluções e técnicas para realizar uma quantidade suficiente de testes que atinja a cobertura necessária para garantir a corretude do software podem ser encontradas na literatura. 235

Outra área de pesquisa bastante desafiadora na área de teste de software é a geração automática de casos de teste, considerando que a elaboração de casos testes manualmente é um processo que consome muito tempo e esforço. Sendo assim, diversas propostas são elaboradas dia após dia com o objetivo de tornar o processo de teste mais ágil, menos susceptível a erros e dependente da interação humana. 235

Na área de inspeção de software, grandes desafios podem ser observados com o objetivo de encontrar estratégias para diminuir a quantidade de defeitos de um software. Na literatura, podem ser encontradas pesquisas e artigos com estudos focados neste objetivo. 235

Referências 236

Boehm, B. W. and BASILI, V.R. (2001) “Software Defect Reduction Top 10 List.”, IEEE Computer 34 (1), p. 135-137. 236

Guerra, A., C and Colombo, R., M., T. (2009) “Tecnologia da Informação: Qualidade de Produto de Software”, PBQP Software. 236

Fagan, M.E. (1976) “Design and Code Inspection to Reduce Errors in Program Development”, IBM Systems Journal, vol. 15, no. 3, p. 182-211. 236

Hedberg, H. (2004) “Introducing the Next Generation of Software Inspection Tools”, In: International Conference of product focused software process improvement, 5, Kansai. Lecture notes in computer science, Berlin: Springer, p. 234-247. 236

Graham, D., Veenendaal, E. v., Evans, I. and Black, R. “Foundations of software testing”, ISTQB Certification, Thomson Learning, 2007. 236

Wong, Y. K. “Modern Software Review Techniques and Technologies”, IRM Press, 2006. 236

Selby, R. W. and Basili, V. R., et al. (1987). “Cleanroom software development: an empirical evaluation”, IEEE Trans on Software Engineering , SE-13(9), 102-37. (Chs. 4,22) 236

Mills, H. D. and Dyer, M., et al. (1987). “Cleanroom software engineering”, IEEE Software, 4(5), 19-25. (Chs. 3,4,22) 236

11.2.2. Áreas de Conhecimento 240

11.2.2.1. Requisitos de Software 241

11.2.2.5. Manutenção de software 245

11.2.2.8. Processo de Engenharia de Software 250

11.2.2.9. Métodos e ferramentas de engenharia 252

11.2.2.10. Qualidade de software 254

Referências 259

Visão Geral da Comunicação 296

14.1. Processo da Comunicação 297

14.1.1. A Comunicação 297

Através das hierarquias de autoridade e orientações formais. 297

Integração social dentro de grupos satisfazendo as necessidades sociais. 297

Fornece subsídios para facilitar a tomada de decisão 298

14.1.4. A Comunicação em Organizações 300

Atualmente, o ambiente organizacional é caracterizado por mudanças contínuas, assim, surgindo a necessidade de mudança nos modelos tradicionais das práticas da comunicação organizacional para manter a competitividade empresarial. 300

14.1.5. Comunicação em projetos 300

14.1.6. A Comunicação como desafio para o Gerente de Projetos 302

14.2. Gerenciamento de Comunicação em Projetos 303

14.2.1.1. Entradas para o Planejamento das comunicações: 305

14.2.1.2. Ferramentas e Técnicas para o Planejamento das comunicações: 308

14.2.1.3. Saídas do Planejamento das comunicações: 309

*1. Plano de gerenciamento das comunicações* 309

TEMPLATE DO PLANO DE COMUNICAÇÃO 309

1. Introdução 309

2. Necessidades de Informação 309

3. Tipos de Informação 310

4. Formatos (templates de relatórios) 310

5. Glossário 310

14.2.2. Distribuição das informações 310

14.2.3. Relatório de Desempenho 314

14.2.4. Gerenciar as partes interessadas 319

Em Projetos Distribuídos, a comunicação é a base para definir como serão repassadas as informações para as partes interessadas envolvidas no projeto. Não existe uma regra para gerenciar projetos distribuídos, mas existem boas práticas que são pontos relevantes e que ajudam os projetos a chegarem a seu objetivo fundamental: sua conclusão no prazo, dentro do custo e com qualidade. Na literatura, podem ser encontradas pesquisas e artigos com estudos focados neste assunto. 323

Referências 324

Alves, A. A comunicação na gerência do projeto. Revista: Techoje: uma revista de opinião. Disponível em: http://www.ietec.com.br/site/techoje/categoria/ detalhe\_artigo/101. Acessado em: set. 2009. 324

Arcanjo, C. (2008). Contexto da Comunicação nas Organizações. Disponível em: http://www.webartigos.com/articles/5381/1/contexto-da-comunicacao-na-gestao-das-organizacoes/pagina1.html. Acessado em: out. 2009. 324

Barbosa, L. O desafio da comunicação eficaz no Gerenciamento de Projetos. Revista: Techoje: uma revista de opinião. Disponível em: http://www.ietec.com.br/site/techoje/categoria/detalhe\_artigo/61. Acesso em: set. 2009. 324

Carvalho, M.; Mirandola, D. A comunicação em projetos de TI: uma análise comparativa das equipes de sistemas e de negócios, v.17 n.2, São Paulo maio/ago. 2007. Disponível:http://www.scielo.br/scielo.php?script=sci\_arttext&pid=S0103-6513200700 0200009&lng=pt&nrm=iso&tlng=pt. Acessado em: out. 2009. 324

Castelo, L. Gerência Participativa: A Comunicação e o Gerente. Disponível em: http://www.geranegocio.com.br/html/geral/gp4.html. Acessado em: set. 2009. 325

Jacob, M. Importância da comunicação na Gerência de Projetos.Revista: Techoje: uma revista de opinião. Disponível em: http://www.ietec.com.br/site/ techoje/categoria/detalhe\_artigo/100. Acessado em: set. 2009. 325

Pimenta, J. A Comunicação nas Empresas e em Projetos. Revista: Techoje: uma revista de opinião. Disponível em: http://www.ietec.com.br/site/techoje/categoria/ detalhe\_artigo/691. Acessado em: out. 2009. 326

PMI (Project Management Institute) A Guide to the Project Management Body of Knowledge – Guia PMBOK® 4. ed. Upper Darby, 2008. 326

Rivas, M. Planejamento & comunicação para estabelecer um diferencial competitivo. Revista: Techoje: uma revista de opinião. Disponível em: http://www.ietec.com.br/site/ techoje/categoria/detalhe\_artigo/379. Acessado em: set. 2009. 327

Schneider, G. (2008) O gerente de projetos também cuida da comunicação. Webinsider. Disponível em: http://webinsider.uol.com.br/index.php/2008/11/05/o-gerente-de-projetos-tambem-cuida-da-comunicacao/. Acessado em: set. 2009. 327

[15.1. Importância da Medição 103](#_Toc245121640)

[15.2. O que são Métricas 104](#_Toc245121641)

[Referências 106](#_Toc245121642)

[16 Gestão de Programas 109](#_Toc245121643)

[Programas 109](#_Toc245121644)

[Gerenciamento de Programas 111](#_Toc245121645)

[Relação entre Gerenciamento do Programa e Gerenciamento do Projeto 112](#_Toc245121646)

[Temas do Gerenciamento de Programa 112](#_Toc245121647)

[Gerenciamento de Benefícios 113](#_Toc245121648)

[Gerenciamento de *Stakeholders* 113](#_Toc245121649)

[Governança 114](#_Toc245121650)

[Ciclo de vida do programa 116](#_Toc245121651)

[Fase 1: Set up Pré-Programa 116](#_Toc245121652)

[Fase 2: Set up Programa 117](#_Toc245121653)

[Fase 3: Estabelecer estrutura de gestão do Programa 118](#_Toc245121654)

[Fase 4: Benefícios Incrementais 118](#_Toc245121655)

[Fase 5: Encerramento 119](#_Toc245121656)

[Processos do Gerenciamento de Programa 120](#_Toc245121657)

[Grupo Processos de Iniciação 120](#_Toc245121658)

[Grupo Processos de Planejamento 122](#_Toc245121659)

[Grupo Processos de Execução 124](#_Toc245121660)

[Grupo Processos de Monitoramento e Controle 125](#_Toc245121661)

[Grupo Processos de Encerramento 125](#_Toc245121662)

[Tópicos de Pesquisa 126](#_Toc245121663)

[Sugestões de Leitura 126](#_Toc245121664)

[Exercícios 127](#_Toc245121665)

[Referências 127](#_Toc245121666)

[Gestão de Portfólio de Projetos 101](#_Toc245121667)

[Introdução 101](#_Toc245121668)

[Definição de Portfólio 102](#_Toc245121669)

[Estratégia Corporativa e Gestão de Portfólio 103](#_Toc245121670)

[Gestão de Portfólio versus Gestão de Múltiplos Projetos 104](#_Toc245121671)

[Relação entre a Gestão de Portfólio e a Gestão de Projetos/Programas 105](#_Toc245121672)

[Métricas em Gestão de Portfólio 105](#_Toc245121673)

[Gerente de Portfólio 106](#_Toc245121674)

[Modelos e Padrões de Gestão de Portfólio 106](#_Toc245121675)

[Padrão de Gestão de Portfólio [PMI 2006] 107](#_Toc245121676)

[Processo Stage-Gate [Cooper et al 2001] 111](#_Toc245121677)

[Processo Integrado de Seleção e Priorização de Projetos [Archer and Ghasemzadeh 1999] 114](#_Toc245121678)

[Estudo de Caso: Gestão de Portfólio de Projetos no SERPRO 116](#_Toc245121679)

[Sugestões de Leitura 118](#_Toc245121680)

[Tópicos de Pesquisa(trabalhos futuros e correntes) 118](#_Toc245121681)

[O Impacto da Gestão de Portfólio de Projetos em Projetos de Tecnologia da Informação [Reyck et al 2005] 118](#_Toc245121682)

[Portfolius: Um Modelo de Gestão de Portfólio de Projetos de Software [Correia 2005] 118](#_Toc245121683)

[Seleção de Projetos em um Portfólio para Apoio a Tomada de Decisão [Ghasemzadeh and Archer 2000] 119](#_Toc245121684)

[Um Processo Integrado para Seleção de Projetos em um Portfólio [Archer and Ghasemzadeh 1999] 119](#_Toc245121685)

[Exercícios 119](#_Toc245121686)

[Referências Bibliográficas 119](#_Toc245121687)

[ Introdução 101](#_Toc245121688)

[ Papéis e funções 102](#_Toc245121689)

[ Objetivos de um PMO 103](#_Toc245121690)

[ Classificações dos PMOs 104](#_Toc245121691)

[ Boas práticas na Implantação de PMOs 111](#_Toc245121692)

[ Caso de sucesso na implantação de um PMO 113](#_Toc245121693)

[ O Serviço Federal de Processamento de Dados – SERPRO 113](#_Toc245121694)

[ Motivação 115](#_Toc245121695)

[ Implantação 116](#_Toc245121696)

[ Estratégia da Implantação 116](#_Toc245121697)

[ Fases da Implantação 117](#_Toc245121698)

[ Benefícios Alcançados 118](#_Toc245121699)

[ Melhoria Contínua 118](#_Toc245121700)

[ Tópicos de Pesquisa 119](#_Toc245121701)

[ Sugestões de Leitura 120](#_Toc245121702)

[Durante a construção deste capítulo foram identificados algumas sugestões de leitura interessantes que podem ajudar o leitor a melhor compreender o contexto de escritório de projetos. Estas leituras são listadas a seguir: 120](#_Toc245121703)

[ Exercícios 121](#_Toc245121704)

[ Qual das opções abaixo não faz parte das típicas funções de um PMO? 121](#_Toc245121705)

[ Reportar status dos projetos para gerentes superiores. 121](#_Toc245121706)

[ Desenvolver e padronizar uma metodologia padronizada. 121](#_Toc245121707)

[ Monitorar e controlar o desempenho dos projetos. 121](#_Toc245121708)

[ Desenvolver e manter um painel de controle de projetos. 121](#_Toc245121709)

[ Autorizar investimentos para a organização. 121](#_Toc245121710)

[ Qual dos grupos de funções abaixo não faz parte do grupo definido na pesquisa de Hobbs e Aubry? 121](#_Toc245121711)

[ Monitoramento e Controle do Desempenho do Projeto. 121](#_Toc245121712)

[ Financiamento de recursos. 121](#_Toc245121713)

[ Desenvolvimento das Competências e Metodologias em Gerenciamento de Projetos. 121](#_Toc245121714)

[ Gerenciamento de Múltiplos Projetos. 121](#_Toc245121715)

[ Gerenciamento Estratégico. 121](#_Toc245121716)

[ Quais dos objetivos apresentados abaixo não podem ser citados como objetivos de um PMO? 121](#_Toc245121717)

[ Contratação de pessoal. 121](#_Toc245121718)

[ O apoio na comunicação eficiente entre os gerentes de projeto e a alta administração. 121](#_Toc245121719)

[ Melhora na eficiência do planejamento e condução dos projetos. 121](#_Toc245121720)

[ Orientar e dar suporte aos gerentes de projetos. 121](#_Toc245121721)

[ Uniformizar processos, práticas e operações de Gerenciamento de Projetos. 121](#_Toc245121722)

[ De acordo com Kerzner, os três tipos de PMO’s são: 122](#_Toc245121723)

[ Escritório de Projetos Funcional, Escritório de Projetos de Grupo de Clientes e Escritório de Projetos Corporativo. 122](#_Toc245121724)

[ Escritório de Projetos Nível 1, Escritório de Projetos Nível 2 e de Escritório de Projetos Nível 3. 122](#_Toc245121725)

[ Escritório de Projetos local, Escritório de Projetos regional e corporativo. 122](#_Toc245121726)

[ Equipe de Projeto Autônoma, Escritório de Suporte de Projetos e Centro de Excelência em Gestão de Projetos. 122](#_Toc245121727)

[ Centro de Excelência em Gestão de Projetos, Escritório de Gerência de Programas e *Chief Project Officer*. 122](#_Toc245121728)

[ Segundo Dinsmore cinco são os tipos de PMO, sendo eles: 122](#_Toc245121729)

[ Escritório de Projeto Master, Escritório de Suporte de Projetos, Centro de Excelência em Gestão de Projetos, Escritório de Gerência de Programas e *Chief Project Officer* . 122](#_Toc245121730)

[ Equipe de Projeto Autônoma, Escritório de Suporte de Projetos, Centro de Excelência em Gestão de Projetos, Escritório de Gerência de Programas e *Chief Project Officer*. 122](#_Toc245121731)

[ Equipe de Projeto Autônoma, Escritório de Suporte de Projetos, Centro de Excelência em Gestão de Projetos, Escritório de Gerência de Programas e Centro de Projetos. 122](#_Toc245121732)

[ Equipe de Projeto Autônoma, Escritório de Suporte de Projetos, Centro de Excelência em Gestão de Projetos, Escritório de Controle de Projetos e *Chief Project Officer*. 122](#_Toc245121733)

[ Equipe de Projeto Autônoma, Escritório de Suporte de Projetos, Escritório Estratégico de Projetos, Escritório de Gerência de Programas e *Chief Project Officer*. 122](#_Toc245121734)

[ De acordo com Crawford podemos classificar os PMO’s em 3 (três) níveis denominados: 123](#_Toc245121735)

[ Analisando a Figura abaixo, a qual representa um organograma organizacional de uma empresa fictícia, que tipo Escritório de Projetos melhor se enquadraria segundo a caixa destacada em cinza? 124](#_Toc245121736)

[ Analisando a Figura abaixo, a qual representa um outro organograma organizacional de uma segunda empresa fictícia, que tipo Escritório de Projetos melhor se enquadraria segundo a caixa destacada em cinza? 124](#_Toc245121737)

[ 124](#_Toc245121738)

[Referências 125](#_Toc245121739)

[14.1. Introdução a Maturidade em Gestão de Projetos 102](#_Toc245121740)

[14.2. Modelos de Maturidade em Gestão de Projetos 103](#_Toc245121741)

[14.2.1. Organizational Project Management Maturity Model - PMI 103](#_Toc245121742)

[14.2.2. Project Management Maturity Model – PM Solutions 104](#_Toc245121743)

[14.2.3. Modelo de Maturidade em Gerenciamento de Projetos – Darci Prado 106](#_Toc245121744)

[14.2.4. Portfolio, Programme and Project Management Maturity Model – OGC 106](#_Toc245121745)

[14.2.5. Kerzner Project Management Maturity Model – Harold Kerzner 109](#_Toc245121746)

[14.3. OPM3 109](#_Toc245121747)

[14.3.1. Estrutura do Modelo 109](#_Toc245121748)

[14.3.2. Avaliação da Maturidade 110](#_Toc245121749)

[14.3.3. Implantação do Modelo 112](#_Toc245121750)

[14.4 MMGP 114](#_Toc245121751)

[14.4.1. Estrutura do Modelo 114](#_Toc245121752)

[14.4.2. Avaliação da Maturidade 116](#_Toc245121753)

[14.4.3. Implantação do Modelo 116](#_Toc245121754)

[14.5. KPMMM 117](#_Toc245121755)

[14.5.1. Estrutura do Modelo 117](#_Toc245121756)

[14.5.2. Avaliação da Maturidade 118](#_Toc245121757)

[14.5.3. Implantação do Modelo 121](#_Toc245121758)

[14.6. Um estudo de caso 122](#_Toc245121759)

[14.6.1. Metodologia 122](#_Toc245121760)

[14.6.2. Resultados coletados 123](#_Toc245121761)

[14.6.3. Perfil dos participantes 123](#_Toc245121762)

[14.6.4. Segmentação por nível de maturidade 126](#_Toc245121763)

[14.6.5. Segmentação por percentual de aderência aos níveis de maturidade 128](#_Toc245121764)

[14.6.6. Conclusão 130](#_Toc245121765)

[14.7. Análise Comparativa 131](#_Toc245121766)

[14.8. Sugestões de Leitura 133](#_Toc245121767)

[14.9. Tópicos de Pesquisa 134](#_Toc245121768)

[14.10. Exercícios 134](#_Toc245121769)

[Referências 135](#_Toc245121770)

Governança em TIC 139

o Gestão em TIC 139

Relevância e Evolução do Papel da TIC nas Organizações 141

Da Gestão à Governança em TIC 144

o Modelos de Gestão em TIC 147

COBIT 147

ITIL 148

BSC 148

IT Flex 148

COSO 149

ISO/IEC 20000 150

VAL IT 151

CMMi sob a perspectiva de Governança de TI 152

o ITIL 152

Definição 152

Histórico 152

Regulamentação do ITIL 153

 Certificações / Treinamentos 153

 Direitos Autorais 154

 Publicação de Conteúdos Oficiais 154

 Fórum de Fomento (iTSMF) 155

Estrutura do ITIL 156

 *Service Strategy* (Estratégia de Serviços) 156

 *Service Design* (Planejamento de Serviços) 156

 *Service Transition* (Transição de Serviços) 156

 *Service Operation* (Operação de Serviços) 156

 *Continual Service Improvement* (Aprimoramento Contínuo de Serviços) 156

O que não é ITIL 158

Fronteiras com outros modelos e limitações 159

Ponto de Partida 160

Comentários sobre práticas de sucesso 161

Público Alvo 162

Utilização do ITIL 163

o COBIT 164

Definição 164

Histórico 165

Regulamentação do COBIT 166

 Certificações / Treinamentos 166

 Direitos Autorais 167

 Publicação de Conteúdos Oficiais 167

 Fórum de Fomento (ISACA) 168

Estrutura do COBIT 168

 Primeira Dimensão do Cubo – Processos de TI 169

 Segunda Dimensão do Cubo – Critérios de Informação 171

 Terceira Dimensão do Cubo – Recursos de TI 172

Não é COBIT 173

Fronteiras com outros modelos 174

Ponto de Partida 175

Comentários sobre práticas de sucesso 176

Público Alvo 176

Utilização do COBIT 177

o Iniciativas de Integração dos Principais Modelos 178

o Implantação de Modelos de Gestão 179

o Tópicos de Pesquisa 182

o Sugestões de Leitura 183

o Exercícios 185

o Referências 187

Parte 1

PROCESSOS