UNIVERSIDADE GAMA FILHO

POSEAD

MARCOS ROGERIO SANTIAGO

“Ensaio do SWEBOK – Software Engineering Body Of Knowledge ”

GOIÂNIA – GO

2011

UNIVERSIDADE GAMA FILHO

POSEAD

MARCOS ROGERIO SANTIAGO

“Ensaio do SWEBOK – Software Engineering Body Of Knowledge ”

Trabalho de Conclusão de Curso apresentado em cumprimento às exigências para obtenção do título de especialista latu sensu em Gestão de Tecnologia da informação da Universidade Gama Filho – POSEAD

Orientador: Professor Rogério Alvarenga

GOIÂNIA – GO

2011

Agradeciemtos

Agradeço a minha esposa Tatiane por tanta paciência nesse momento de minha vida. Quando eu mais precisei, lá estava ela sendo aquele suporte necessário.

Ao professor Rogério Alvarenga, que, com muita tranquilidade soube me ajudar nos momentos em que quase desisti.

Agradeço também ao pessoal da Universidade Federal de Lavras pelo apoio e suporte recebidos.

Sobretudo, ao meu bom Deus, que é o responsável por tudo o que acontece em minha vida. Minhas vitórias são, sem dúvida, por causa Dele.

Resumo

Este trabalho tem por objetivo o estudo do SWEBOK, ferramenta criada pelo IEEE (Institute of Electrical and Electronics Engineers), conhecida entidade de especificação de padrões no mundo para ser a principal referência para engenharia de software. Tem o objetivo também de fornecer uma tradução de sua última versão para a língua portuguesa, e contribuir assim para o seu terceiro objetivo, que é sua divulgação. O SWEBOK comparado a outros modelos de qualidade de software tem sua divulgação e utilização tímida, mas em expansão. Possui seus critérios e conteúdos objetivos, claros e bem definidos, tornando-se uma ferramenta de excelência em aplicabilidade na prática, além de fornecer formalizações direcionadas para certificações a profissionais e acadêmicas.

Sumário

Lista de abreviaturas

ADL	–	Architecture Description Languages
BABOK	–	Guide to the Business Analysis Body of Knowledge
CASE	–	Computer–Aided Software Engineering
CBD	–	Component–Based Design
CCB	–	Configuration Control Board
CM	–	Configuration Management
CMMI	–	Capability Maturity Model Integration
COTS	–	Commercial Off–the–Shelf Software
CRC	–	Class Responsibility Collaborator card
DAG	–	Directed Acyclic Graph
DFD	–	Data Flow Diagram
EF	–	Experience Factory
EG	–	Engenharia de Sfotware
ERD	–	Entity-Relationship Diagram
FCA	–	Functional Configuration Audit
FP	–	Function Point
FSM	–	Functional Size Measurement
GCS	–	Gerência de Configuração de Software
HRM	–	Human Resources Management
ICSM	–	International Conference on Software Maintenance
IDEAL	–	Initiating, Diagnostic, Establishing, Acting, Learning
IDL	–	Interface Description Language
INCOSE	–	International Council on Systems Engineering
KA	–	Knowledge Area
MTBF	–	Mean Time Between Failures
OMG	–	Object Management Group
PCA	–	Physical Configuration Audit
PDCA	–	Plan, Do, Check, Act
PDL	–	Pseudo-Code and Program Design Language
PMBOK	–	Guide to the Project Management Body of Knowledge
QIP	–	Quality Improvement Paradigm – Paradigma de Melhoria de Qualidade
SADT	–	Structured Analysis and Design Tecnique
SCAMPI	–	Standard CMMI Appraisal Method for Process Improvement
SCCB	–	Software Configuration Control Board
SCE	–	Software Engineering Evaluation
SCI	–	Software Configuration Item
SCM	–	Software Configuration Management

SCMP	–	Software Configuration Management Plan
SCR	–	Software Change Request
SCSA	–	Software Configuration Status Accounting
SEI	–	Software Engineering Institute
SEPG	–	Software Engineering Process Group
SQA	–	Software Quality Assurance
SQM	–	Software Quality Management
SRET	–	Software Reliability Engineered Testing
SRS	–	Software Requirement Specification
TQM	–	Total Quality Management
UML	–	Unified Modeling Language
USNRC	–	U.S. Nuclear Regulatory Commission
V&V	–	Verification and Validation
Y2K	–	Year 2000

Lista de figuras

1 Introdução

1.1 Formulação do problema

A Engenharia de Software é uma área do conhecimento da computação voltada para a especificação, desenvolvimento e manutenção de sistemas de software aplicando tecnologias e práticas de gerência de projetos e outras disciplinas, objetivando organização, produtividade e qualidade. Os fundamentos científicos para a Engenharia de Software envolvem o uso de modelos abstratos e precisos que permitem ao engenheiro especificar, projetar, implementar e manter sistemas de software, avaliando e garantindo suas qualidades.

Hoje em dia encontramos software em todas as partes. Estão nos celulares, nos aparelhos de cozinha, equipamentos médicos, brinquedos, etc. Linhas de código estão presentes em todos estes produtos e a cada dia temos mais produtos surgindo. Tudo isso sem entrarmos no campo das aplicações web, dos sistemas empresariais, científicos, educacionais, internet e por aí vai. Com toda essa demanda a Engenharia de Software ainda é uma área nova, com poucas instituições que oferecem cursos de graduação.

Em 1995, o padrão internacional ISO/IEC 12207, serviu de subsídio para a elaboração do corpo de conhecimento da engenharia de software, o SWEBOK. Ocorre que o SWEBOK, um guia de referência tão importante, não tem o devido status do qual ele merece. Não é raro encontrar pessoas da área de desenvolvimento que ainda não sabem de sua existência. Cursos de pós-graduação com ênfase em engenharia de software não fazem menção a este guia.

Este trabalho se propõe a tradução e ao detalhamento deste guia em nível suficiente para que seja evidenciada sua importância bem como sua divulgação.

1.2 Revisão de Literatura

Esta pesquisa influenciada pelos trabalhos de Júlio César Sampaio do Prado Leite e Roger S. Pressman.

Roger S. Pressman é uma autoridade internacionalmente reconhecida em melhoria de processo de software e tecnologias de engenharia de software. Por mais de três décadas, ele trabalhou como engenheiro de software, gerente, professor, autor e consultor, com foco em questões de engenharia de software. Dr. Pressman é atualmente presidente do R. S. Pressman & Associates, Inc., uma empresa de consultoria especializada em métodos de engenharia de software e formação. Dr. Pressman tem escrito muitos artigos técnicos, é um colaborador regular de revistas da indústria, e é autor de seis livros.

Júlio César Sampaio do Prado Leite é Doutor em Ciência da Computação pela Universidade da Califórnia, Irvine, professor do Departamento de Informática da PUC-RIO e vem realizando pesquisas na área de Engenharia de Software. Professor da cadeira de Sistemas de Informação no Departamento de Informática da UERJ. Consultor nas áreas de Engenharia de Software e Sistemas de Informação.
É dele a citação:

> "A década de 60 e princípio da década de 70 mostraram a inviabilidade da ideia do sistema de informação global para toda uma organização. A despeito dos grandes avanços tecnológicos, várias organizações passaram pela desagradável experiência de terem altíssimos gastos em processamento de dados e, contrariamente ao esperado, terem acumulados grandes insucessos na implantação de ambiciosos Sistemas de Informação, os quais englobariam toda a organização. Um Sistema de Informação não é como muitos acreditavam e infelizmente, alguns ainda acreditam, um sistema puramente técnico. "...Cabe a gerência de Engenharia de Software todo o suporte no desenvolvimento e manutenção do software constante do sistema computacional de apoio aos Sistemas de Informação da organização."

O Instituto de Engenheiros Eletricistas e Eletrônicos (IEEE) é uma organização profissional sem fins lucrativos, fundada nos Estados Unidos. É a maior (em número de sócios) organização profissional do mundo. O IEEE foi formado em

1963 pela fusão do Instituto de Engenheiros de Rádio (IRE) com o Instituto Americano de Engenheiros Eletricistas (AIEE). O IEEE tem filiais em muitas partes do mundo, sendo seus sócios engenheiros eletricistas, engenheiros da computação, cientistas da. computação, profissionais de telecomunicações etc. Sua meta é promover conhecimento no campo da engenharia elétrica, eletrônica e computação. Um de seus papéis mais importantes é o estabelecimento de padrões para formatos de computadores e dispositivos.

A leitura destes autores foi de fundamental importância para que se observasse a necessidade de uma ferramenta que agrupasse as melhores práticas, as melhores ideias em nível mundial, na área de Engenharia de Software, em um local, e a partir de onde as direções para o futuro dessa nova ciência fossem apontadas.

1.3 Objetivos gerais

Promover o maior conhecimento do SWEBOK pelos profissionais que atuam na engenharia de software e fornecer um primeiro contato ao corpo de conhecimento de Engenharia de Software.

1.4 Objetivos específicos

- Contextualização da utilização do guia SWEBOK
- Definição dos principais conceitos de Engenharia de Software
- Exposição das áreas de conhecimento
- Exposição das disciplinas relacionadas à Engenharia de Software
- Expansão do uso do guia SWEBOK

1.5 Justificativa

Roger Pressman, no livro Engenharia de Software, 6a edição, 2006, capítulo 1 página 13, diz:

> "O software tornou-se o elemento chave na evolução de sistemas e produtos baseados em computador, e uma das tecnologias mais importantes em todo o mundo. Ao longo dos últimos 50 anos, o software evoluiu de um ferramental especializado em solução de problemas e análise de informações para um produto de indústria. Mas ainda temos problemas na construção de software de alta qualidade no prazo e dentro do orçamento. O software - programas, dados e documentos - dirigido a uma ampla variedade de tecnologias e aplicações, continua a obedecer a uma série de leis que permanecem as mesmas há cerca de 30 anos. O intuito da engenharia de software é fornecer uma estrutura para a construção de software com alta qualidade".

A partir do posicionamento de Pressman, inferimos a necessidade de se manter uma documentação padronizada e dinâmica que incorpore as melhores práticas e metodologias, norteando, em nível global, os rumos – desta relativamente

nova – ciência que desponta.

1.6 Delimitação do tema

Este trabalho tem como objetivo o estudo e divulgação do guia SWEBOK, buscando a ampliação do seu uso entre os profissionais da área de engenharia de software, corroborando ainda para sua sedimentação como principal ferramenta para padronização de conceitos no mundo.

1.7 Metodologia de pesquisa

Este trabalho foi realizado com base em pesquisas bibliográficas que serviram de fundamentação a sua ideia principal, a centralização de padronizações. Para isso, o uso da internet para localização de informações relevantes, foi de crucial valor.

A pesquisa bibliográfica justifica-se por se tratar de tema que não fora abordado no curso de especialização e por se entender que é algo de suma importância para a disciplina de engenharia de software. Tal ferramenta – SWEBOK – deveria ser um dos temas principais a serem abordados pela sua riqueza de informações, que tem como uma de suas premissas manter-se sempre atualizado.

1.8 Estrutura do trabalho

Este trabalho foi elaborado de acordo com os parágrafos a seguir:

O Capítulo 1 trata as razões que motivaram a pesquisa, a formulação do problema, as exposições do objetivo geral e dos objetivos específicos, a justificação do tema do trabalho e sua delimitação.

O Capítulo 2 trata dos fundamentos teóricos para o trabalho, seus aspectos e como é formada sua estrutura. No tópico História, faz uma descrição de toda a origem da engenharia de software desde suas primeiras menções nos meios sociais até a data presente. No tópico Estrutura de Descrição das KAs é informado como o documento foi pensado de forma a resolver algum problema do mundo real. Nos

tópicos subsequentes são detalhadas todas as áreas do conhecimento na forma como elas serão abordadas. Também informa a maneira como deve ser compreendida cada KA e uma breve descrição dos seus objetivos. Os tópicos dos apêndices A, B, C e D referem-se aos suplementos e informações necessárias para a manutenção do próprio guia, tais como critérios, conselhos, padrões e taxonomia.

No Capítulo 3 são abordadas as conclusões deste trabalho.

2 Referencial teórico

2.1 Aspectos do Conhecimento de Eng. de Software

2.1.1 História

Em 1958, John Tukey, o estatístico de renome mundial, cunhou o termo software. O termo "Engenharia de Software" foi utilizado no título de uma conferência da Otan, realizada na Alemanha em 1968. Em 1972, a IEEE Computer Society publica pela primeira vez seu relatório sobre Engenharia de Software. Em 1976 foi fundada uma comissão dentro da IEEE Computer Society com a incumbência de desenvolver padrões de engenharia de software. A primeira visão geral sobre Engenharia de Software a sair da IEEE Computer Society resultou de um esforço liderado por Fletcher Buckley, no padrão IEEE 730 que versava sobre garantia de qualidade de software e que foi concluído em 1979. O propósito do padrão IEEE 730 era fornecer requisitos mínimos de uniformidade para a preparação e conteúdo do planejamento de software. Esta norma influenciou o desenvolvimento e conclusão de outros temas: gerenciamento de configuração, teste de software, requisitos de software, design de software, verificação e validação de software. No período de 1981-1985, a IEEE Computer Society realizou uma série de oficinas sobre a aplicação de padrões de engenharia de software. Nessas oficinas, profissionais envolvidos compartilhavam suas experiências com as atuais normas. Também se realizavam nessas oficinas sessões onde eram discutidos os rumos para as normas, incluindo as medidas e métricas para a engenharia de software, produtos e processos. O planejamento resultou também no padrão IEEE 1002, padrão de taxonomia de engenharia de software (1986), que proporcionou uma nova visão de

engenharia de software. O padrão descreve a forma e conteúdo da taxonomia dos padrões de engenharia de software. Ele explica vários tipos de padrões, seus relacionamentos funcionais e externos bem como o papel das várias funções participantes no ciclo de vida do software. Em 1990, o planejamento para um padrão internacional com uma visão geral foi iniciado. Este planejamento foi focado em conciliar os pontos de vista de processo de software da IEEE 1074 e revisado pelo padrão U.S. DoD 2167A. Esta revisão foi finalmente publicada como DoD Std 498. O padrão internacional foi terminado em 1995 com a designação ISO/IEC 12207 e dado o título de padrão para os processos de ciclo de vida do software. O padrão ISO/IEC 12207 forneceu um importante ponto de partida para o corpo de conhecimentos deste guia. Foi o Conselho de tutores do IEEE Computer Society que aprovou a moção apresentada em maio de 1993 por Fletcher Buckley que resultou na eleboração do guia. O conselho da Association for Computing Machinery (ACM) aprovou a referida moção em agosto de 1993. As duas propostas levaram a criação de uma comissão mista, sob a liderança de Mario Barbacci e Stuart Zweben que atuaram como copresidentes A missão da comissão mista era "Estabelecer os conjuntos de critérios e normas adequadas para a prática profissional da engenharia de software sobre a qual decisões industriais, certificações profissionais e currículos educacionais pudessem se basear". A direção do comitê organizou grupos de trabalho nas seguintes áreas:

1 - Definição do corpo de conhecimentos e práticas recomendadas;

2 - Definição de ética e padrões profissionais;

3 - Definição de currículos educacionais para graduação, pós-graduação e educação continuada.

Este guia fornece o primeiro componente: Corpo de conhecimentos e práticas recomendadas. O código de ética e padrões profissionais de engenharia de software foi concluído em 1998 e aprovado tanto pelo conselho da ACM quanto pelo conselho da IEEE e tem sido adotado por inúmeras empresas e outras organizações. O currículo educacional para alunos de graduação foi concluído com esforço da IEEE e ACM em 2004.

2.1.2 Definições iniciais

O Guia não deve ser confundido com o Corpo de Conhecimento propriamente dito, que já existe publicado na literatura. O propósito do guia é descrever qual porção do Corpo de Conhecimento é geralmente aceito, para organizar esta porção, e para fornecer um acesso atual a esta. Informações adicionais ao significado dado à “geralmente aceito” podem ser encontradas adiante no Apêndice A. O Guia para o Corpo de Conhecimento de Engenharia de Software – SWEBOK - foi estabelecido com os cinco seguintes objetivos:

1. Promover uma visão consciente da engenharia do software no mundo inteiro;
2. Esclarecer o lugar "e estabelecer o limite" da engenharia de software com respeito a outras disciplinas como ciência da computação, gerenciamento de projetos, engenharia da computação, e matemática;
3. Caracterizar os conteúdos da disciplina de engenharia de software;
4. Fornecer um acesso ao Corpo de Conhecimento de Engenharia de Software;
5. Fornecer uma base para desenvolvimento de currículo, certificação individual e licenciamento de material;

O primeiro desses objetivos, uma visão mundial consistente da engenharia de software, foi apoiada por um processo de desenvolvimento que empregou aproximadamente 500 revisores de 42 países na fase de Stoneman (1998–2001) levando à versão de Prova, e mais de 120 revisores de 21 países na fase de Ironman (2003) levando à versão 2004. Mais informação quanto ao processo de desenvolvimento pode ser encontrada no Prefácio e no site www.swebok.org. As sociedades profissionais e eruditas e as agências públicas implicadas na engenharia de software foram oficialmente contatadas, feitas conscientes deste projeto, e convidadas para participar do processo de revisão. Editores associados foram recrutados de América Norte, Costa Pacífica e Europa. Apresentações do projeto foram feitas em vários eventos internacionais e outras foram planejadas para o próximo ano.

O segundo dos objetivos, o desejo de estabelecer um limite da engenharia de software, motiva a organização fundamental do guia. O material que é reconhecido como sendo parte desta disciplina é organizado nas dez primeiras Áreas de Conhecimento (Knowledge Areas - KA) enumeradas na Figura 1. Cada uma dessas

KAs é tratada como um capítulo neste guia.

Requisitos de Software
Design de Software
Construção de Software
Teste de Software
Manutenção de Software
Gerência de Configuração de Software
Gerência da Engenharia de Software
Processo de Engenharia de Software
Ferramentas e Métodos da Engenharia de Software
Qualidade de Software

Figura 1: Áreas de Conhecimento (KA) do SWEBOK

No estabelecimento de um limite, é também importante identificar que disciplinas compartilham este limite, e muitas vezes uma intersecção comum, com a engenharia de software. Para este fim, o Guia também reconhece oito disciplinas relacionadas, enumeradas na figura 2. Engenheiros de software, naturalmente, devem ter conhecimento do material desses campos (e as descrições das KAs podem fazer referência a eles). Contudo, não é um objetivo do Guia caracterizar o conhecimento das disciplinas relacionadas, mas ao invés, que conhecimento é visto como específico à engenharia de software.

* Administração	* Ergonomia de Software
* Ciências da Computação	* Gerenciamento da Qualidade
* Engenharia de Computação	* Gerenciamento de Projetos
* Engenharia de Sistemas	* Matemática

Figura 2: Disciplinas relacionadas

2.1.2.1 Organização Hierárquica

A organização das descrições das KAs, apoia o terceiro objetivo do projeto – uma caracterização dos conteúdos da engenharia de software. Especificações detalhadas fornecidas pela equipe editorial do projeto aos editores associados quanto aos conteúdos das descrições das KAs podem ser encontradas no Apêndice A.

O Guia usa uma organização hierárquica para decompor cada KA em um conjunto de tópicos com títulos reconhecíveis. Dois ou três níveis de separação fornecem um modo razoável de encontrar tópicos de interesse. O Guia trata os tópicos selecionados em uma maneira compatível com as principais escolas do pensamento e com separações geralmente encontradas na indústria e na literatura de engenharia de software e padrões. As separações dos tópicos não presumem determinados domínios de aplicação, usos para negócios, filosofia de gerência, métodos de desenvolvimento, e assim por diante. A extensão da descrição de cada tópico é somente a necessária para entender a natureza geralmente aceita dos tópicos e para o leitor encontrar com sucesso o material de referência.

2.1.2.2 Material de referência e matriz

Para fornecer um acesso atual ao conhecimento — o quarto objetivo do projeto — o Guia identifica o material de referência de cada KA, inclusive capítulos de livros, artigos referenciados, ou outras fontes reconhecidas de informação com autoridade. Cada descrição da KA também inclui uma matriz que relaciona o material de referência aos tópicos enumerados. O volume total da literatura pretende-se que seja dominado pela realização de uma educação universitária com mais quatro anos da experiência.

Pode-se argumentar que algumas KAs, como projeto de software, por exemplo, merecem mais páginas de material de referência do que outros. Tal modulação pode ser aplicada em futuras edições do Guia.

Deve-se observar que o Guia não tenta ser abrangente nas suas citações. Muito material que é tanto conveniente quanto excelente não é referenciado. O material foi selecionado em parte porque — tomado como uma coleção — fornece a cobertura dos tópicos descritos.

2.1.2.3 Profundidade de tratamento

De início, uma pergunta surgiu quanto à profundidade do tratamento que o Guia deve fornecer. A equipe de projeto adotou uma abordagem que apoia o quinto objetivo do projeto — fornecimento de uma base de desenvolvimento de currículo, certificação, e licença. A equipe editorial aplicou o critério de conhecimento geralmente aceito, a distinção de conhecimento avançado e de pesquisa (com base

em maturidade) e do conhecimento especializado (com base em generalidade da aplicação). A definição vem do Instituto de Gerenciamento de Projetos (Project Management Institute - PMI): “O conhecimento geralmente aceito aplica-se à maior parte de projetos, na maior parte do tempo, e o consenso comum valida o seu valor e a eficácia. ”

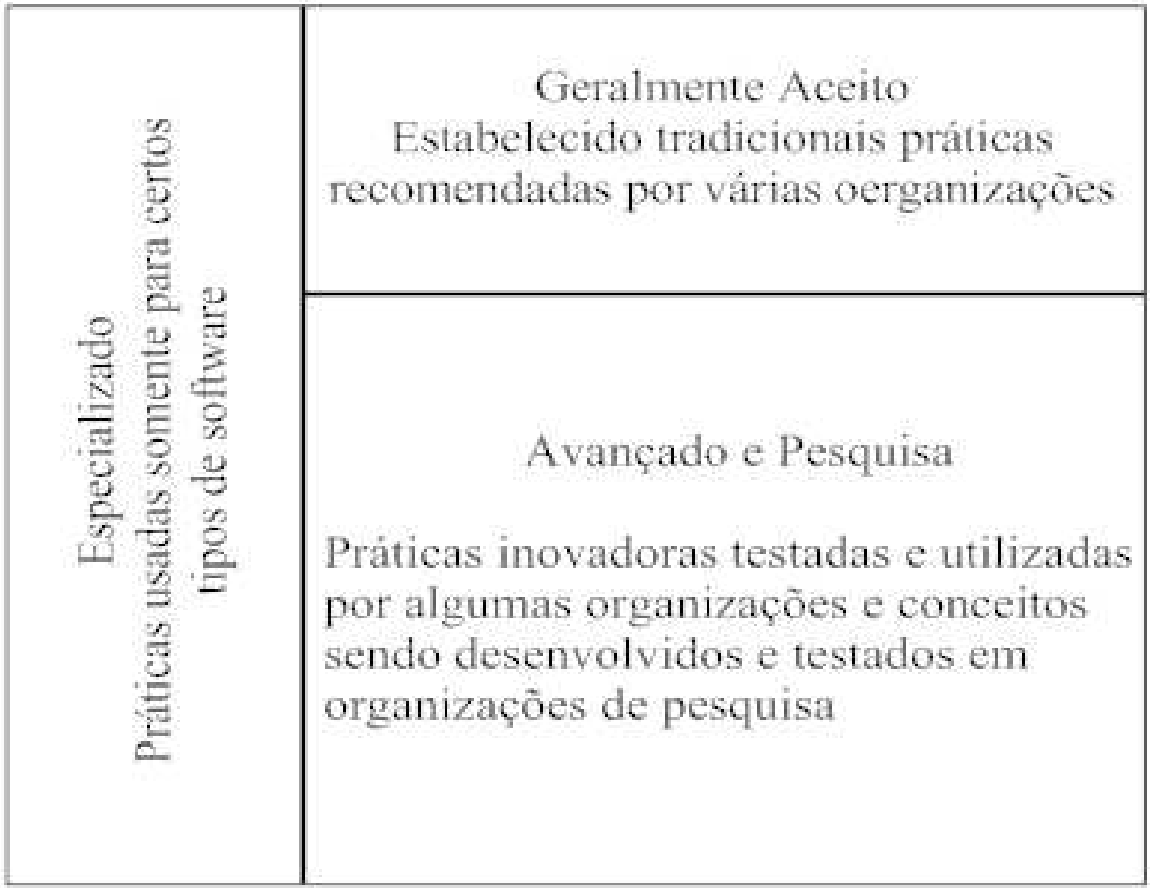


Figura 3: Categorias de Conhecimento

Contudo, o termo “geralmente aceito” não implica que o conhecimento indicado deva ser uniformemente aplicado a todas as tentativas de engenharia de software — as necessidades de cada projeto determinam isto — mas implica que engenheiros de software competentes e capazes devam ser equipados com este conhecimento para potencial aplicação. Mais precisamente, o conhecimento geralmente aceito deve estar incluído no material de estudo do exame de licença de engenharia de software que graduados deveriam prestar depois de quatro anos da experiência de trabalho. Embora este critério seja específico ao estilo de educação dos Estados Unidos e não necessariamente se aplique a outros países, foi considerado útil. No entanto, as duas definições de conhecimento geralmente aceito

devem ser vistas como complementares.

2.1.2.4 Estrutura de descrição das KAs

A descrição das KAs são estruturadas como a seguir.

Na introdução, uma breve definição da KA e um resumo do seu respectivo escopo e do relacionamento entre outras KA são apresentados. O desdobramento em tópicos que compõe a estrutura central da descrição da KA é feita, descrevendo a decomposição da KA em subáreas, tópicos e sub tópicos. Para cada tópico ou sub tópico, uma descrição curta é fornecida, junto com uma ou mais referências.

O material de referência foi escolhido porque se considera que ele constitui a melhor apresentação do conhecimento quanto ao tópico tratado, considerando as limitações impostas à escolha de referências. Uma matriz liga os tópicos ao material de referência.

A última parte da descrição da KA é a lista de referências recomendadas. O Apêndice A de cada KA inclui sugestões leituras complementar para aqueles usuários que desejam aprender mais sobre os tópicos da KA. O Apêndice B apresenta a lista de padrões mais relevantes para a KA.

2.1.2.5 KA de Requisitos de Software

Um requisito é definido como uma propriedade que deve ser exposta para resolver algum problema do mundo real. A primeira subárea de conhecimento é chamada de Fundamentos de Requisitos de Software. Ela inclui definições dos próprios requisitos de software, mas também os principais tipos de requisitos como: diferença entre produto e processo, propriedades funcionais e não funcionais e propriedades emergentes. A subárea também descreve a importância de requisitos quantificáveis e também apresenta a diferença entre requisitos de software e requisitos de sistemas.

A segunda subárea de conhecimento é o *Processo de Requisitos*, que introduz o próprio processo, orientando o resto das cinco subáreas e exposição como a engenharia de requisitos se relaciona com os demais processos de engenharia de software. Ela descreve modelos de processo, atores de processo, processos de suporte e gerenciamento, e o processo de qualidade e melhoria.

A terceira subárea é a Elicitação de Requisitos, que descreve de onde os

requisitos de software vêm e como o engenheiro de software pode obtê-los. Ele inclui as fontes de requisitos e técnicas elicitação.

A quarta subárea, Análise de Requisitos, se preocupa com o processo de analisar os requisitos para:

- Detectar e resolver conflitos entre requisitos
- Descobrir os limites do software e como ele deve interagir com o seu ambiente
- Elaborar requisitos de sistema de forma a se transformarem requisitos de software

A análise de requisitos inclui a classificação dos requisitos, a modelagem conceitual, o desenho da arquitetura e alocação de requisitos, e a negociação de requisitos.

A quinta subárea é a Especificação de Requisitos. A especificação de requisitos tipicamente refere-se à produção de um documento, ou o seu equivalente eletrônico, que pode ser sistematicamente revisto, avaliado e aprovado. Para sistemas complexos, em particular os que implicam o uso de componentes que não necessariamente são software. No mínimo, três tipos diferentes de documentos são produzidos: definição de sistema, especificação de requisitos do sistema, e especificação de requisitos de software. A subárea descreve os três documentos e as atividades subjacentes.

A sexta subárea é a Validação de Requisitos, cujo objetivo é buscar qualquer problema antes que os recursos sejam implantados no envio dos requisitos. A validação de requisitos preocupa-se com o processo de examinar os documentos de requisitos para assegurar que eles estão definindo o sistema correto (isto é, o sistema que o usuário espera). É subdividido em descrições de procedimento de revisão de requisitos, prototipação, validação de modelo e testes de aceitação.

A sétima subárea, que é a última subárea, é chamada de Considerações Práticas. Ela descreve os tópicos que têm de ser compreendidos na prática. O primeiro tópico é a natureza iterativa do processo de requisitos. Os próximos três tópicos são fundamentalmente sobre a gerência de mudanças e a manutenção de requisitos em um estado que exatamente reflete o software a ser construído, ou que já tenha sido construído. Ele inclui gerência de mudanças, atributos de requisitos, e investigação de requisitos. O tópico final é a medição de requisitos.

2.1.2.6 KA de Design de Software

De acordo com a definição do IEEE [IEEE 610.12-90], design é "o processo de definir a arquitetura, componentes, interfaces, e outras características de um sistema ou componente" e também "o resultado de processo". A KA Design de Software é dividida em seis subáreas.

A primeira subárea apresenta os *Fundamentos de Design de Software*, que formam uma base subjacente à compreensão do papel e do escopo do design de software. Existem conceitos de software genéricos, o contexto do design de software, o processo de design de software, e as técnicas de autorização de design de software.

A segunda subárea agrupa o conjunto de *Questões-chave em Design de Software*. Essas questões chaves incluem colaboração, controle e tratamento de eventos, a distribuição de componentes, tratamento de exceções e erros e tolerância à falha, interação e apresentação, e persistência de dados.

A terceira subárea é *Estrutura e Arquitetura de Software*, os tópicos da qual são estruturas de arquiteturas e pontos de vista, estilos de arquitetura, padrões de design, e, finalmente, as famílias dos programas e frameworks.

A quarta subárea descreve *Análise de Qualidade do Design e Avaliação* do software. Enquanto existe uma KA inteira dedicada à qualidade de software, esta subárea apresenta os tópicos especificamente relacionados ao design de software. Esses aspectos são atributos de qualidade, análise de qualidade, e técnicas de avaliação e medidas.

A quinta subárea é *Notações de Design de Software*, que é dividida em descrições estruturais e comportamentais.

A última subárea descreve *Estratégias de Design de Software e Métodos*. Primeiro, as estratégias gerais são descritas, seguidas por métodos de design orientados por função, métodos de design orientados a objeto, design centrando na estrutura de dados, design baseado em componentes e outros.

2.1.2.7 KA de Construção de Software

Construção de software se refere à criação detalhada de trabalho, significando software através da combinação de codificação, verificação, testes de

unidades, testes de integração, e depuração. A KA possui três subáreas.

A primeira subárea é chamada de Fundamentos de Construção de Software. Os três primeiros tópicos são princípios básicos da construção: minimização da complexidade, antecipação de mudanças e construção com foco na verificação. O tópico último discute padrões da construção.

A segunda subárea é descreve o Gerenciamento da Construção. Os tópicos tratados são modelos de construção, planejamento da construção e mensuração da construção.

A terceira subárea cobre as Considerações Práticas. Os tópicos são design de construção, linguagens de programação, codificação, teste de construção, reutilização, qualidade de construção, e integração.

2.1.2.8 KA de Testes de Software

Teste de Software compõe-se da verificação dinâmica de uma seleção de domínios de execuções normalmente infinito, contra o comportamento esperado. Ela inclui cinco subáreas. A KA inicia com a descrição de Fundamentos de Testes de Software. Primeiro a terminologia de testes de é apresentada, então são descritos assuntos relacionados com testes de software, e finalmente, os relacionamentos entre testes e as outras atividades são descritos.

A segunda subárea é chamada de Níveis de Teste. É dividida entre os alvos dos testes e os objetivos dos testes.

A terceira subárea é chamada de Técnicas de Teste. A primeira categoria inclui os testes baseados na intuição e experiência do testador. Um segundo grupo compreende técnicas baseadas na especificação, seguidas por baseadas no código, em falhas, no uso e baseadas na natureza da aplicação. Uma discussão de como selecionar e combinar as técnicas apropriadas também é apresentada.

A quarta subárea cobre Mensurações Relacionadas aos Testes. As medidas são agrupadas conforme o relacionamento de forma a avaliar o programa que esta sofrendo testes e avaliar o os testes executados.

A última subárea descreve o Processo de Testes e inclui considerações práticas e as atividades de teste.

2.1.2.9 KA de Manutenção de Software

Uma vez na operação, as anomalias são descobertas, modificação no ambientes operacional, e novos requisitos dos usuários são expostos. A fase de manutenção do ciclo de vida começa a partir da entrega, porem, as atividades de manutenção ocorrem muito antes. A KA de Manutenção de Software é dividida em quatro subáreas.

Primeiramente é apresentado os Fundamentos de Manutenção de Software: definições e terminologia, a natureza de manutenção, a necessidade de manutenção, os principais custos de manutenção, evolução de software e as categorias de manutenção.

A segunda subárea agrupa as Questões-chave em Manutenção de Software. São as questões técnicas, de gerência, estimativas de custos de manutenção, e a mensuração da manutenção de software.

A terceira subárea descreve o Processo de Manutenção. Os tópicos aqui são os processos de manutenção e atividades de manutenção.

Técnicas para Manutenção constituem a quarta subárea. Essa subárea inclui a compreensão de programa, a reengenharia, e a engenharia reversa.

2.1.2.10 KA de Gerenciamento de Configuração do Software

A Gerência de Configuração de Software (GCS) é a disciplina de identificar a configuração do software em pontos distintos no tempo com o propósito de sistematicamente controlar modificações à configuração e de manter a integridade e a auditoria da configuração em todas as partes do ciclo de vida de sistema. Este KA inclui seis subáreas.

A primeira subárea é a Gerência do Processo de GCS. Ela cobre os tópicos do contexto organizacional de GCS, limites e orientação para o GCS, planejamento para o GCS, o próprio plano de GCS, e a monitoração da GCS.

A segunda subárea é a Identificação de Configuração de Software, que identifica itens a serem controlados, estabelece esquemas de identificação dos itens e as suas versões, também estabelece os instrumentos e técnicas a serem utilizadas na aquisição e no gerenciamento dos itens controlados. Os primeiros tópicos nesta subárea são a identificação dos itens a serem controlados e a biblioteca de software.

A terceira subárea é o Controle de Configuração de Software, que é a

gerência de mudanças durante o ciclo de vida de software. Os tópicos são: primeiro, solicitando, avaliando, e aprovando alterações no software; em segundo lugar, implementação de modificações no software; e terceiro, desvios e não aprovações (renúncia de alterações).

A quarta subárea é a Registros de Estado de Configuração de Software. Os seus tópicos são a informação sobre posição de configuração doe software e a reportagem do estado de configuração do software.

A quinta subárea é a Auditoria de Configuração de Software. Ele compõe-se da auditoria de configuração funcional do software, auditoria de configuração física do software, e auditorias no processo a partir de uma base de referência do software.

A última subárea é a Gerência de Liberação e Entrega de Software, cobrindo elaboração de versões de software e gerenciamento de liberação de software.

2.1.2.11 KA de Gerenciamento da Engenharia de Software

O KA Gerenciamento da Engenharia de Software, aponta o gerenciamento e mensuração da engenharia de software. Enquanto a mensuração é um aspecto importante em todas as KAs, está aqui que o tópico de programas de mensuração é apresentado. Há seis subáreas na KA de gerenciamento de engenharia de software. As cinco primeiras cobrem assuntos relacionados com gerenciamento de projetos de software e a sexta o sexto descrevem programas de mensuração de software.

A primeira subárea é a Iniciação e Definição de Escopo, que compreende a determinação e a negociação de requisitos, análise de praticabilidade, e processo da reavaliação e a revisão de requisitos.

A segunda subárea é o Planejamento de Projeto de Software e inclui o planejamento de processo, determinando pacotes de trabalhos (deliverables), o esforço, agendamento e a estimativa de custos, a alocação de recursos, a gerência dos riscos, a gerência de qualidade, e o plano de gerenciamento.

A terceira subárea é a Aprovação do Projeto de Software. Os tópicos tratados nesta KA são a implementação de planos, gerência de contrato de fornecedores, a implementação do processo de mensuração, monitoração de processo, controle de processo, e a divulgação de informações do projeto.

A quarta subárea é Revisão e Avaliação, que inclui os tópicos de determinar a

satisfação dos requisitos e revisão e avaliação da performance.

A quinta subárea descreve o Fechamento: determinação de fechamento e atividades de fechamento.

Finalmente, a sexta subárea descreve a Mensuração da Engenharia de Software, mais especificamente, programas de mensuração. As mensurações de produto e processos são descritos na KA Processo de Engenharia de Software. Muitas outras KA também descrevem medidas específicas para a sua respectiva KA. Os tópicos desta subárea incluem o estabelecimento e comprometimento da mensuração, planejamento do processo de mensuração, execução do processo de mensuração, e avaliação da mensuração.

2.1.2.12 KA Processo de Engenharia de Software

A KA Processo de Engenharia de Software trata da definição, implementação, avaliação, mensuração, gerenciamento, alterações e melhora do próprio processo de engenharia de software. É dividida em quatro subáreas.

A primeira subárea apresenta o Processo de Implementação e Mudanças. Os tópicos nesta KA são a infraestrutura de processo, o ciclo do processo de gerenciamento do software, modelos de implementação do processo e mudanças e considerações práticas.

A segunda subárea trata da Definição do Processo. Ela inclui os tópicos de modelos de ciclo de vida do software, processos de ciclo de vida de software, notações das definições do processo, adaptação do processo e automação.

A terceira subárea é a Avaliação de Processo. Os tópicos aqui incluem modelos de avaliação de processo e métodos de avaliação do processo.

A quarta subárea descreve Mensuração de Produto e Processo. O processo de engenharia de software cobre a medição do produto de processo em geral. As mensurações específicas de cada KA são abordadas nas respectivas KA. Os tópicos são mensuração do processo, mensuração do produto de software, a qualidade da mensuração dos resultados, modelos de informação de software e técnicas de mensuração do processo.

2.1.2.13 KA Ferramentas e Métodos de Engenharia de Software

A KA Ferramentas e Métodos para Engenharia de Software inclui, como a

própria descrição apresenta, ferramentas para serem aplicadas à engenharia de software e também métodos para serem aplicados na engenharia de software.
A subárea de Ferramentas para Engenharia de Software usa a mesma estrutura que Guia, com um tópico de cada uma das nove KAs de engenharia de software. Em um tópico adicional é fornecido: assuntos sobre ferramentas diversas, como ferramentas para técnicas de integração, que são potencialmente aplicáveis a todas as classes de ferramentas.

A subárea Métodos para Engenharia de Software é dividida em quatro subseções: métodos heurísticos que tratam com aproximações informais, métodos formais que se baseiam em aproximações matematicamente, e métodos de prototipação se baseiam em aproximações de desenvolvimento de software em torno de múltiplas formas de prototipação.

2.1.2.14 KA Qualidade de Software

A KA Qualidade de Software aborda considerações relativas à qualidade de software que vão além dos processos de ciclo de vida de software. Um vez que a qualidade de software é um assunto presente em todas as partes na engenharia de software, também é considerado em muitas outras KAs, e o leitor notará pontos desta KA nas outras KAs do Guia. Esta KA possui três subáreas.

A primeira subárea descreve Fundamentos da Qualidade de Software como uma cultura e ética na engenharia de software, os valores e custos da qualidade, características de modelos e qualidade, e melhoria da qualidade.

A segunda subárea trata dos Processos de Gerenciamento da Qualidade do Software. Os tópicos tratados aqui são garantia da qualidade de software, verificação e validação, e por fim, revisões e auditorias.

A terceira e última subárea descreve Considerações Práticas relacionadas à qualidade de software. Os tópicos são requisitos de qualidade de software, caracterização de defeito, técnicas de gerenciamento da qualidade do software, e mensuração da qualidade do software.

2.1.2.15 KA Disciplinas Relacionadas com Engenharia de Software

O último capítulo trata das disciplinas relacionadas com a engenharia de software. Para circunscrever a engenharia de software, é necessário identificar as

disciplinas com que a engenharia de software compartilha limites. Este capítulo identifica, em ordem alfabética, essas disciplinas relacionadas. Para cada disciplina relacionada, e utilização de uma fonte reconhecida à base de consenso, são identificados:

- uma definição informativa (quando factível);
- uma lista de KAs.

As seguintes disciplinas são relacionadas com a engenharia de software:

* Administração	* Ergonomia de Software
* Ciências da Computação	* Gerenciamento da Qualidade
* Engenharia de Computação	* Gerenciamento de Projetos
* Engenharia de Sistemas	* Matemática

Figura 4: Disciplinas relacionadas com engenharia de software

2.1.2.16 Apêndice A

O apêndice descreve as especificações fornecidas pela equipe editorial aos editores associados para o conteúdo, referências recomendadas, formato, e estilo das descrições das KAs.

2.1.2.17 Apêndice B

O segundo apêndice descreve a proposta do projeto da evolução da Guia. O Guia 2004 é simplesmente a edição atual de um guia que continuará evoluindo para conhecer as necessidades da comunidade de engenharia de software. O planejamento quanto à evolução ainda não esta completo, mas uma tentativa é feita com este apêndice. Como está sendo escrito, este processo foi endossado pelo Industrial Advisory Board e resumido para o Board of Governors da Computer Society do IEEE, porém, ainda não é consolidado ou implementado.

2.1.2.18 Apêndice C

O terceiro apêndice é um conjuntos dos padrões mais relevantes, sendo a maior parte formada por padrões do IEEE e da ISO, alocada para as KAs do Guia SWEBOK.

2.1.2.19 Apêndice D

Como uma ajuda, notavelmente dos currículos dos desenvolvedores (e outros usuários), em suporte ao quinto objetivo do projeto, o quarto apêndice aponta um

conjunto de categorias pedagógicas comumente atribuídas a Benjamin Bloom. O conceito é que os objetivos educativos podem ser classificados em seis categorias representando crescimento de profundidade: conhecimento, compreensão, aplicação, análise, síntese, e avaliação. Os resultados deste exercício para todas as KAs podem ser encontrados no apêndice D. Este apêndice não deve, contudo, ser examinado como uma classificação definitiva, mas muito mais como um ponto de partida.

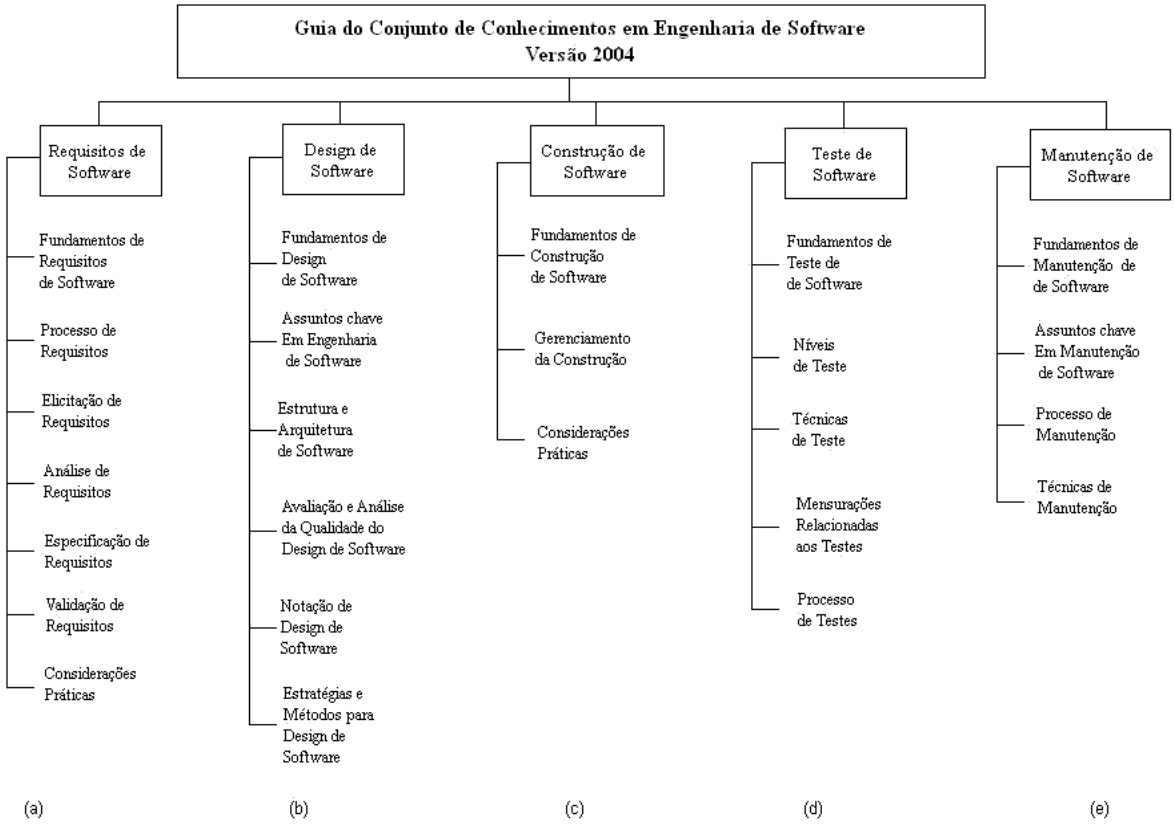


Figura 5: Primeiras 5 áreas do conhecimento

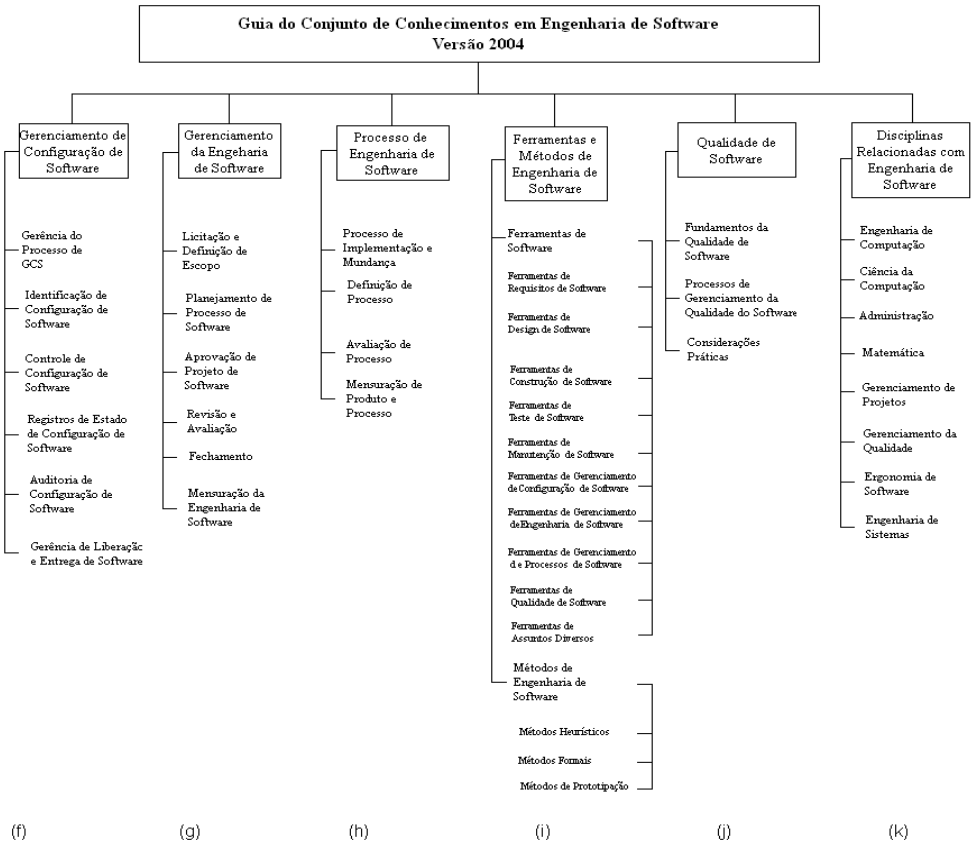


Figura 6: Últimas 6 áreas do conhecimento

2.1.3 Áreas de conhecimento

2.1.3.1 Requisitos de Software

A área de conhecimento Requisitos de Software trata da elicitação, análise, especificação e validação dos requisitos de software. A má condução dessas atividades tornam projetos de engenharia de software criticamente vulneráveis.

Requisitos de Software expressam as necessidades e restrições colocadas sobre um produto de software que contribui para a solução de um problema do mundo real. O termo “Engenharia de Requisitos” é muito utilizado para denotar um tratamento sistemático dos requisitos. Por consistência, será evitado o uso do termo neste guia.

A área Requisitos de Software é fortemente relacionada a:

- Design do Software;
- Teste de Software;
- Manutenção do Software;
- Gerência de Configuração do Software;
- Gerência da Engenharia de Software;
- Processo de Engenharia de Software;
- Qualidade de Software.

A divisão aqui adotada é inteiramente compatível com as seções da IEEE 12207 que referem-se às atividades de requisitos. Um risco inerente à divisão proposta é que um modelo do tipo cascata pode ser inferido. Para se precaver disso, a sub área 2, Processos de Requisitos, é projetada para prover uma visão geral em alto nível do processo de requisitos estabelecendo os recursos e restrições sobre os quais o processo opera e que servem para configurá-lo.

Uma decomposição alternativa poderia ser uma estrutura baseada em produto (requisitos do sistema, requisitos de software, protótipos, casos de uso,..)

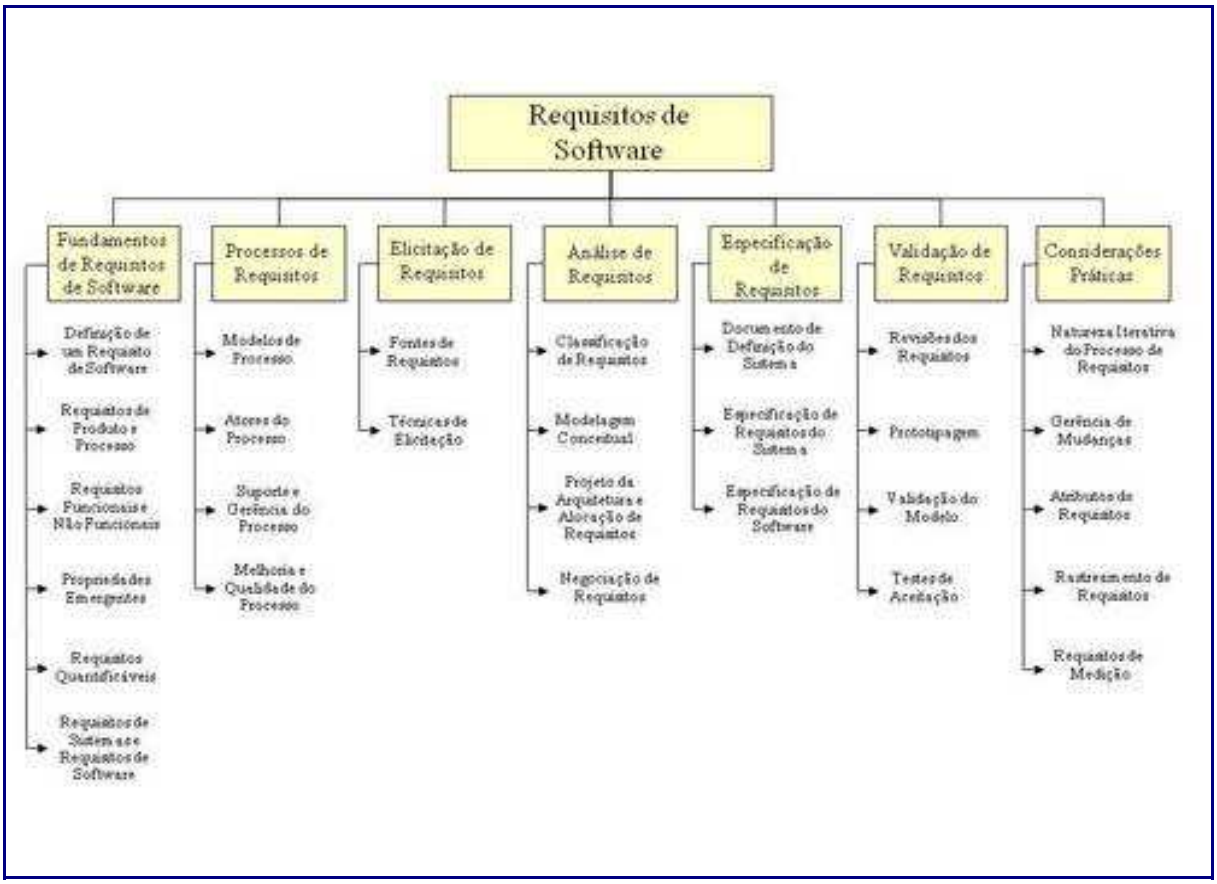


Figura 7: Tópicos dos Requisitos de software

2.1.3.1.1 Fundamentos de Requisitos de Software

a) Definição de Requisito de Software

Propriedade que deve ser apresentada pelo software para resolver um problema do mundo real. Requisito de Software porque ele se ocupa com problemas endereçados pelo Software: Software é desenvolvido ou adaptado para resolver um problema em particular.

Problema pode, por exemplo, ser automatizar uma tarefa utilizando o software, para suportar um processo de negócio de uma organização, controlar um dispositivo qualquer. Requisitos, de software em particular, são tipicamente uma combinação de requisitos de diversas pessoas em diferentes níveis de organização e de ambientes no qual o software opera.

Atributos de requisitos (além das propriedades comportamentais que expressam). Classificação de prioridade para habilitar trade-offs em face de recursos finitos. Status que permita acompanhar e monitorar o progresso do projeto; Identificação única para que possa ser controlado pela Gerência de Configuração e gerenciado durante todo o ciclo de vida.

b) Requisitos de Produto e Processo

Requisitos de Produto: Requisitos sobre o software a ser desenvolvido. Por exemplo: “O software deve verificar se um aluno cumpriu os pré-requisitos antes de aceitar a sua matrícula numa disciplina”; Requisitos de Processo: são essencialmente restrições impostas ao desenvolvimento do software tais como linguagem de programação, processos e técnicas de desenvolvimento. Podem ser implícitos ou explícitos. Podem ser impostos pela organização desenvolvedora, pelo cliente ou por terceiros.

c) Requisitos Funcionais e Não Funcionais

Requisitos Funcionais (capabilities): Descreve funções que o software deve executar como por exemplo “controlar fluxo de caixa”;

Requisitos Não Funcionais (restrições ou requisitos de qualidade): São aqueles que agem para restringir uma solução. São conhecidos como restrições ou requisitos de qualidade que podem ser classificados em Requisitos de Desempenho, Requisitos de Segurança, Requisitos de Manutenibilidade, Requisitos de Confiabilidade e outros tipos; Exemplos: “O custo não pode ultrapassar R$50.000,00” ou “Uma função deve estar disponível ao usuário 99,99% do tempo”.

d) Propriedades Emergentes

Alguns requisitos representam propriedades emergentes do software, isto é, propriedades que não são endereçadas por um componente mas cuja satisfação depende da inter operação de diversos componentes do sistema. Por exemplo, a eficiência (throughput) de um call center depende do sistema telefônico, sistema de informação e de todos os operadores interagindo sob condições operacionais.

A rapidez de atendimento de um cliente numa loja depende de fatores tais como facilidades de consulta ao cadastro de clientes, facilidades para liberação de mercadoria no estoque, agilidade na liberação pelo caixa, etc.
Propriedades emergentes são crucialmente dependentes da arquitetura do sistema.

e) Requisitos Quantificáveis

Requisitos precisam ser definidos com clareza e sem ambiguidade, o quanto

for possível, e, quando for o caso, quantitativamente;

Evitar definições vagas ou que não possam ser verificadas tais como “O atendimento ao cliente deve ser rápido”. Fica melhor “O cliente deve ser atendido num intervalo de tempo de 2 minutos”.

f) Requisitos de Sistema e Requisitos de software

Sistema: Uma combinação de elementos interagindo para obter um determinado objetivo. Inclui software, hardware, pessoas, informações, técnicas, serviços e outros elementos de suporte;

Requisitos de Sistema: Requisitos para o sistema como um todo;

Requisitos de Software: Num sistema que possui componentes de software, requisitos de software são derivados dos requisitos do sistema e alocados ao software;

Requisitos do Usuário: O guia define requisitos do usuário como sendo requisitos dos usuários do sistema ou usuários finais.

2.1.3.1.2 Processo de Requisitos

a) Modelos de Processo

- Não é uma atividade discreta mas cobre desde o início até o final do projeto sendo refinado continuamente durante todo o ciclo de vida;
- Identifica os Requisitos de software como itens de configuração e os gerencia usando as práticas de GC.
- Customizado de acordo com o contexto da Organização e do Projeto

Relaciona-se com atividades de Elicitação, Análise, Especificação e Validação. Inclui atividades de Pesquisa de Mercado e Factibilidade

b) Atores de Processo

- Define os papeis dos participantes do processo;
- Interdisciplinar: Especialistas de requisitos fazem mediação entre os especialistas de domínio e demais interessados (stakeholders);

Exemplos típicos de stakeholders:

- Usuários: os que irão operar o software;

- Clientes: mercado alvo e os que lucram com o produto;
- Analistas de Mercado: especialistas em necessidades do mercado;
- Agentes Reguladores: depende do domínio da aplicação (bancos, transporte, comércio);
- Engenheiros de software: Interessados em facilitar o desenvolvimento do software.

2.1.3.1.3 Elicitação de Requisitos

- Preocupa-se com o “de onde” podem ser obtidos os requisitos de software e como os engenheiros de software podem coletá-los;
- Constitui o primeiro estágio para definir e entender o problema que o software pretende solucionar;
- È uma atividade fundamentalmente humana onde os stakeholders são identificados e são estabelecidas relações entre equipe de desenvolvimento e o cliente;
- Pode ser conhecida por outros nomes: “captura de requisitos”, “descoberta de requisitos” ou “aquisição de requisitos”;
- Um princípio fundamental da boa Engenharia de software: boa comunicação entre usuários do software e os engenheiros de software. Deve-se esforçar para isso. Analistas devem mediar entre o domínio do usuário e o domínio técnico.

a) Fontes de Requisitos

Principais fontes a serem utilizadas:

Metas ou objetivos: Objetivos do Negócio, objetivos de alto nível do software;

Conhecimento do domínio: Conhecimentos sobre o domínio da aplicação;

Stakeholders: Pontos de vista dos diferentes tipos de stakeholders;

Ambiente operacional: Ambiente onde o software será executado;

Ambiente organizacional: Estrutura, cultura e políticas da organização.

b) Técnicas de Elicitação

Identificadas as fontes de requisitos o engenheiro de software deve elicitar os requisitos.

- É uma área difícil, o engenheiro de software poderá ter de enfrentar fatos tais como:
- * usuários podem ter dificuldades em descrever suas tarefas;
- * informações importantes podem ser omitidas;
- * pode haver falta de vontade em cooperar.
- A elicitação não é uma atividade passiva, o engenheiro de software geralmente precisa trabalhar duro para elicitar as informações corretas

Principais técnicas de elicitação utilizadas:
Entrevistas: Forma "tradicional" de elicitação. Possui vantagens e limitações;
Cenários: Criação de contextos para a elicitação com usuários. E se…. ? Como ficaria ?
Protótipos: Para esclarecer dúvidas e simular como seria o funcionamento;
Reunião com facilitadores: brainstorming consultorias, identificação de conflitos;
Observação: Imersão no ambiente alvo para captar os aspectos culturais da organização e as tarefas nele executadas.

2.1.3.1.4 Análise de Requisitos

Processo de analisar requisitos para:

- Detectar e resolver conflitos entre requisitos;
- Descobrir as fronteiras do software e como ele deve interagir com o seu ambiente;
- Elaborar requisitos do sistema para derivar requisitos do software.

A visão tradicional da análise de requisitos faz com que ela se reduza ao modelamento conceitual usando algum método de análise tal como o SADT.
Deve-se ter o cuidado de descrever os requisitos com precisão suficiente para que possam ser validados, sua implementação verificada e seus custos estimados.

a) Classificação de Requisitos

Classificação é feita segundo diversos critérios:

- Funcional ou Não Funcional (Tópico 1.3);
- Derivado de requisito de alto nível, propriedade emergente ou imposta diretamente sobre o software por um stakeholder ou qualquer outra fonte;

- Produto ou Processo se o requisito é sobre o produto ou sobre o processo. Requisitos sobre o processo podem restringir a escolha de um contratante, um processo a ser adotado, a aderência a um padrão,….
- Prioridade em geral são de alta prioridade os requisitos considerados essenciais para preencher os objetivos gerais do software. frequentemente são classificados numa escala de pontos fixos como: obrigatório, muito desejável, desejável ou opcional. A prioridade tem que ser balanceada com o custo de desenvolvimento e implementação;
- Escopo refere-se à extensão com que o requisito afeta o software e os componentes de software. Alguns requisitos e particularmente certos requisitos não funcionais possuem um escopo global e não pode ser alocado a nenhum componente em particular. Podem afetar muito na arquitetura.
- Volatilidade/Estabilidade Alguns requisitos irão mudar durante o ciclo de vida do software e mesmo durante o processo de desenvolvimento. Pode ser útil a informação de quanto um requisito muda. Exemplo: Numa aplicação bancária, requisitos para calcular e creditar juros para clientes são provavelmente mais estáveis que um requisito para suportar uma espécie particular de conta livre de taxas. O molde reflete uma característica fundamental no domínio bancário (essa conta pode ganhar vantagens) enquanto que a anterior pode tornar-se obsoleta por mudança na legislação governamental. Destacando potenciais requisitos voláteis pode ajudar o engenheiro de software estabelecer um design que seja mais tolerante à mudanças.

b) Modelagem Conceitual

Desenvolvimento de modelos de um problema no mundo real é chave para análise de requisitos de software. Seu propósito é auxiliar no entendimento do problema antes de iniciar o projeto da solução. Consequentemente, modelos conceituais compreendem modelos de entidades do domínio do problema configurado para refletir relacionamentos e dependências do mundo real.

Diversos tipos de modelo podem ser desenvolvidos. Esses incluem dados e fluxos de controle, modelos de estado, rastreamento de eventos, interação com usuários,modelos de objetos, modelos de dados, e outros.

Fatores que influenciam na escolha do modelo:

- **Natureza do problema** (software de tempo real => modelo de fluxo de controle e estado);
- **Perícia** do engenheiro de software ou quem irá utilizar o modelo;
- **Requisitos de Processo do Cliente** podem impor sua notação ou método;
- **Disponibilidade** de **ferramentas** e **métodos** (SADT, UML).

Na maioria dos casos é útil começar pela construção de um modelo do contexto do software que fornece uma conexão entre o software pretendido e o ambiente externo. Isso é fundamental para entender o contexto do software no seu ambiente operacional e identificar interfaces.

Modelamento é fortemente acoplado com métodos. Para propósitos práticos, **um método é uma notação (ou um conjunto de notações) suportado por um processo que guia a aplicação das notações.**

Geralmente um método não é muito superior a outros. A larga aceitação de um método ou notação podem resultar em uma extensa e benéfica comunhão de habilidades e conhecimentos.

Modelos formais usando notações baseada na matemática discreta baseados no raciocínio lógico podem ter impacto em alguns domínios especializados. Podem ser impostos por clientes ou padrões e oferecer vantagens na análise de funções ou componentes críticos.

Dois padrões provêm notações que podem ser úteis:

- IEEE Std 1320.1, IDEFO para modelamento funcional
- IEEE Std 1320.2 IDEFIX97 (IDEFObject) para modelar informações.

c) Projeto de Arquitetura e Alocação de Requisitos

Em algum ponto a arquitetura da solução precisa ser derivada. Projeto da arquitetura é o ponto onde o processo de requisitos sobrepõe-se com o projeto do software ou do sistema, e ilustra como é impossível desacoplar com clareza essas duas tarefas. Este tópico é fortemente relacionado com a subárea Estrutura do software e Arquitetura da área de conhecimento Design do Software. Em muitos caos o engenheiro de software age como arquiteto do software pois o processo de analisar e elaborar os requisitos demanda que os componentes que serão responsáveis por satisfazer os requisitos sejam identificados. Isso é alocação

de requisitos a atribuição a componentes, da responsabilidade por satisfazer os requisitos.

- Alocação é importante pois permite a análise detalhada dos requisitos;
- Projeto da arquitetura é fortemente identificado com modelamento conceitual. O mapeamento de entidades do mundo real para componentes de software nem sempre é obvio. Assim, projeto da arquitetura é identificado como um tópico separado. Os requisitos de notações e métodos são os mesmos no modelamento conceitual e no projeto da arquitetura;
- IEEE 1471-2000, sugere uma abordagem de múltiplos pontos de vista para descrever a arquitetura do sistema e seus itens de software.

d) Negociação de Requisitos

- Negociar requisitos X Resolução de conflitos
- Resolver problemas decorrentes de conflitos entre requisitos;
- Dois stakeholders podem requerer features mutuamente incompatíveis;
- Conduzida por especialistas de requisitos;
- Evita-se decisão unilateral: stakeholders envolvidos são consultados em busca de consenso;
- Por razões contratuais, às vezes clientes são envolvidos.

Foi classificado como análise de requisitos de software porque problemas emergem como resultado da análise. Entretanto um caso grande pode também ser considerado como um tópico de validação.

2.1.3.1.5 Especificação de Requisitos

Na maioria das engenharias o termo “especificação” refere-se a atribuição de valores numéricos ou limites para o produto de metas do projeto. Sistemas físicos típicos possuem um número relativamente pequeno desses valores. Software típico, possui um grande número de requisitos e a ênfase é repartida entre executar a quantificação numérica e gerenciar a complexidade da interação entre um grande número de requisitos. Assim, no jargão de engenharia de software, uma ”Especificação de Requisitos de Software” tipicamente refere-se a produção de um

documento ou seu equivalente eletrônico, que possa ser sistematicamente revisado, avaliado e aprovado. Para sistemas complexos envolvendo substanciais componentes não-software pelo menos três documentos são produzidos: Definição do Sistema, Requisitos do Sistema e Requisitos do Software. Nos sistemas simples, apenas o terceiro é requerido.

a) Documento de Definição do Sistema

- Também denominado “Documento de Requisitos do Usuário” ou “Conceitos de Operações”. Registra, num alto nível, os requisitos do sistema;
- Destina-se aos clientes e usuários do sistema (ou seus representantes);
- Define requisitos funcionais e não funcionais do sistema em alto nível, seus objetivos gerais, o ambiente alvo, suas restrições e pressupostos;
- Ponto de vista: Domínio da aplicação;
- Pode conter modelos conceituais para ilustrar o contexto do sistema, cenários, principais entidades do domínio, dados ou informações e fluxogramas.
- IEEE Std 1362, Documento de conceito de operação (modelo).

b) Especificação de Requisitos do Sistema

- Aplica-se a sistemas que possuem número considerável de componentes (software e não software);
- frequentemente separa-se a descrição dos requisitos do sistema da descrição dos requisitos de software. Assim os requisitos de sistema são especificados e os requisitos de software são derivados deles e então os requisitos de componentes do software são especificados;
- Especificação de Requisitos do Sistema é uma atividade de Engenharia de Sistemas (fora do nosso objetivo);
- IEEE Std 1233 é um guia para desenvolver requisitos de sistema.

c) Especificação de Requisitos de Software

- Estabelece base para acordo entre clientes, contratadores ou fornecedores;
- Define o que o produto de software deve ser e o que ele não deve ser;
- Para leitores não técnicos, o documento de especificação de requisitos de

software é frequentemente acompanhado por um documento de definição dos requisitos de software;

- ERS Permite avaliação rigorosa dos requisitos antes do início do seu projeto e reduz “re-projeto”;
- Provê de base realística para estimar custos, prazos e riscos;
- Base para confecção de planos de verificação e validação;
- Normalmente é escrito em linguagem natural. Pode conter suplementos em descrição formal ou semi-formal.
- Normalmente é escrito em linguagem natural. Pode conter suplementos em descrição formal ou semi-formal. A regra geral é escolher notações apropriadas que permitam uma descrição mais precisa e concisa;
- Indicadores de qualidade vem sendo desenvolvidos e podem ser usados para relatar a qualidade da especificação dos requisitos de software para outras variáveis de projeto: custo, aceitação, desempenho, cronograma, reprodutibilidade, …;
- Indicadores de qualidade da especificação de um requisito: imperativos, diretivas, frases fracas, opções, …;
- Indicadores de qualidade da ERS como um todo: tamanho, legibilidade, estrutura do texto, profundidade, …;
- IEEE Std 830 padrão para ERS .

2.1.3.1.6 Validação de Requisitos

- Documentos devem ser revisados por stakeholders diferentes, inclusive por representantes do cliente e do desenvolvedor;
- Documentos de requisitos estão sujeitos à mesma Gerência de Configuração que os demais documentos;
- É normal explicitar um ou mais pontos no processo de requisitos onde a validação é realizada. Isso ajuda prevenir alguns problemas antes que recursos sejam alocados;
- Validação de requisitos diz respeito ao processo de examinar os documentos de requisitos para ter certeza que eles definem o software correto, ou seja, o software que faz o que o usuário espera que ele faça.

a) Revisões de Requisitos

- Provavelmente a inspeção e revisão dos documentos de requisitos constitui a maneira mais comum de validação;
- O grupo de revisão busca por erros, contradições, falta de clareza e desvios de práticas padrões;
- A composição do grupo é muito importante e pelo menos um representante do cliente precisa estar incluído;
- Pode ser útil o fornecimento de checklists para auxiliar;
- Revisões devem ser constituídas quando for completado o Documento de Definição do Sistema, Documento de Especificação do Sistema e Documento de Especificação de Requisitos de Software, ou em qualquer outro ponto do processo;
- IEEE Std 1028 fornece guias para revisão.

b) Prototipagem

- Meio de validar a interpretação que o engenheiro de software faz do requisito;
- Meio para elicitar novos requisitos;

*** Vantagens:**

- Facilita a interpretação de como o engenheiro vê o requisito;
- Feedback útil no caso de interpretações estarem erradas;
- Diminui o esforço em desenvolvimento de requisitos errados.

*** Desvantagem:**

- Desviar a atenção do usuário para a parte “cosmética” ao invés de concentrar no que é essencial;
- Custo de desenvolvimento pode ser alto.

c) Validação do Modelo

- É necessário validar a qualidade dos modelos desenvolvidos durante a análise;
- Por exemplo: Na modelagem de objetos, é útil executar uma análise estática para verificar os caminhos de comunicação existentes entre objetos que, no domínio dos stakeholders, trocam dados;

- Se for utilizada especificação formal, é possível utilizar raciocínio lógico para provar propriedades das especificações.

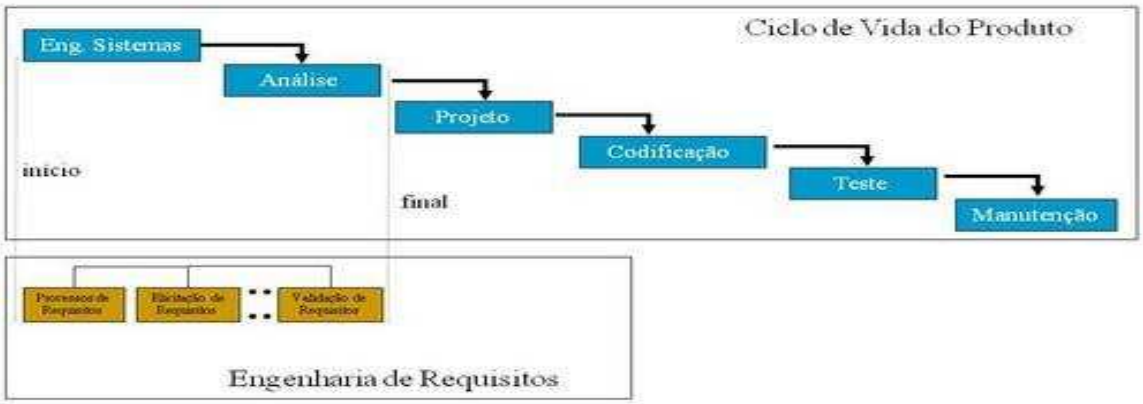
d) Testes de Aceitação

Propriedade importante dos requisitos. **O produto deve ser validado nos seus requisitos:**

- Requisitos que não possam ser validados são meros“desejos”;
- É preciso planejar como verificar cada requisito (Teste de Aceitação);
- Para serem validados, eles precisam primeiro ser analisados até o ponto de serem expressos quantitativamente;
- Pode ser difícil projetar Testes de Aceitação para os Requisitos Não Funcionais;
- Informações adicionais no tópico 2.2.4 Testes de Conformidade - KA Teste de Software.

2.1.3.1.7 Considerações Práticas

Visão Simplista: A decomposição de subáreas apresentada é uma simplificação do processo, na verdade ele se expande por todo o ciclo de vida do software.



Vigura 8: Visão simplista

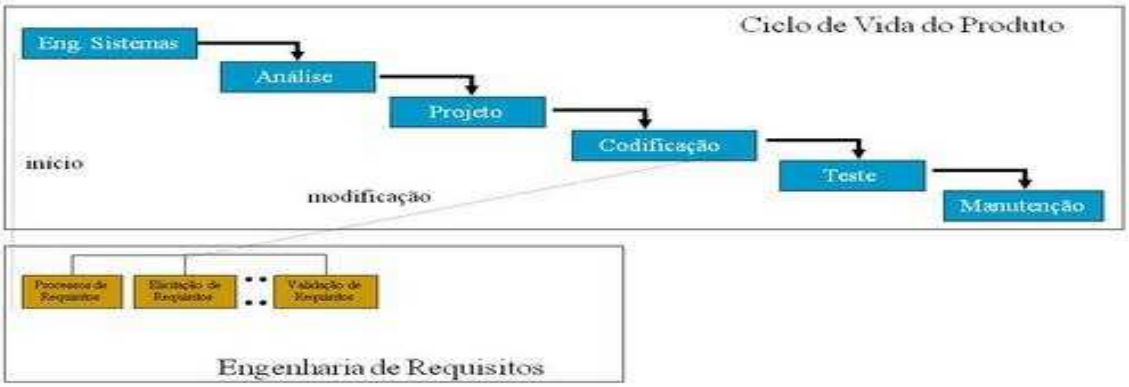


Figura 9: Visão realista

a) Natureza Iterativa do Processo de Requisito

- Mercado atual pressiona indústria para ciclos curtos de desenvolvimento;
- Alguns projetos são desenvolvidos adaptando projetos anteriores ou aproveitando partes deles;
- É quase sempre impraticável implementar requisitos como um processo linear e determinístico;
- É um mito pensar que, nos grandes projetos de software, os requisitos são perfeitamente entendidos e especificados.
- Na maioria das vezes, o requisito sofre aperfeiçoamento continuado só ficando pronto quando o produto estiver terminado. Requisitos podem ir para uma baseline sem que suas propriedades estejam completamente entendidas. Isso arrisca retrabalho de problemas que podem surgir depois no processo de Engenharia de software;
- Engenheiros de software, por necessidades decorrentes do gerenciamento do projeto, precisam tomar algumas atitudes que assegurem a “qualidade” dos requisitos tão alta quanto possível com os recursos disponíveis;
- Devem ser explicitados quaisquer pressupostos, bem como os problemas conhecidos.

- Entendimento dos requisitos envolve tanto os procedimentos do projeto como os de desenvolvimento;
- Ponto crucial no entendimento de requisitos: uma proporção significativa dos requisitos irá mudar devido a erros de análise, mudanças no ambiente de operação ou de negócio, mudanças no mercado onde será inserido o produto;
- Mudanças são inevitáveis e medidas devem ser tomadas para mitigar o seu impacto;
- Mudanças devem ser gerenciadas por um processo definido (rastreamento, análise de impacto e Gerencia de Configuração);
- Processo de requisitos não é tarefa front-end mas expande-se por todo o ciclo de vida do software.

b) Gerência de Mudanças

- A gerência de mudanças é central no gerenciamento de requisitos;
- Gerencia de mudanças descreve o seu papel, os procedimentos que precisam ser efetuados e a análise que deve ser aplicada às propostas de mudanças;
- Gerência de mudanças de requisitos está intimamente ligada à Gerência de Configuração de software.

c) Atributos de Requisitos

Requisitos não são constituídos apenas pela especificação do que é requerido, mas também por um conjunto de informações adicionais que auxiliam a interpretar e gerenciar os requisitos. Isso pode incluir várias classificações, métodos de verificação e aceitação e planos de teste. Pode incluir:

- Resumo do requisito;
- Fonte do requisito;
- Histórico das mudanças;
- **Identificador**: a mais importante pois permite que requisitos tenham identificação única e não ambígua.

d) Rastreamento de Requisitos

- Rastreamento tem a ver com as fontes dos requisitos e com os efeitos sobre

os seus descendentes;

- Fundamental para fazer análise de impacto quando o requisito é modificado;
- Requisitos precisam ser rastreados:
    - Backward: Nas suas origens (requisito e stakeholders que o motivaram);
    - Forward: Nas entidades de projeto geradas a a partir deles (requisitos, módulos de código, testes,…).
- Rastreamento de requisitos num projeto típico irá formar um grafo acíclico dirigido (DAG).

e) Medição de Requisitos

Por questões práticas, é útil possuir algum conceito de “volume” dos requisitos para um particular produto de software.

É útil para avaliar o “tamanho” de uma mudança de requisitos, na estimativa de custo de desenvolvimento, tarefa de manutenção ou simplesmente para ter um denominador comum de comparação com outras medições.

FSM (Medida de Tamanho Funcional): técnica para avaliar o tamanho do corpo de requisitos funcionais (IEEE Std 14143.1 define o conceito de FSM). Informações adicionais sobre medições de tamanho e padrões são encontradas na KA Processo de Engenharia de Software.

**Matriz de Tópicos x Material de Referência **

2.1.3.2 Design de Software

Design é definido no [IEEE610.12-90] de duas formas “o processo de definição da arquitetura, componentes, interfaces, e outras características de um sistema ou componente” e “o resultado do processo.” Visto como um processo, o design de software é o ciclo de vida da engenharia de software no qual os requisitos de software são analisados para produzir uma descrição da estrutura interna dos software’s que servirão como base de sua construção. Mais precisamente, o design de software (resultante) pode descrever a arquitetura de software – é isto, como o software é decomposto e organizado dentro dos componentes – e a interface entre estes componentes. Isso também descreve os componentes no nível de detalhe que permitem sua construção.

O Design de Software emprega uma importante regra no desenvolvimento de software: isso permite engenheiros de software produzirem vários de modelos que formam um tipo de plano de solução para ser implementado. Nós podemos analisar e avaliar estes modelos para determinar se eles nos permitirão atender as diversas necessidades. Nós podemos também examinar e avaliar várias soluções alternativas e confrontá-las. Finalmente nós podemos usar os modelos resultantes para planejar as atividades do desenvolvimento subsequente, em adição ao uso deles como entrada e ponto de começo da construção e teste.

Em uma listagem padrão do processo de ciclo de vida de software tal como IEEE/EIA 12207, Processos de Ciclo de Vida do Software [IEEE12207.0-96], design de software consiste em duas atividades que situa-se entre análises e requerimentos de software e construção de software:

- Design de arquitetura de software (algumas vezes chamada design de alto nível - “Top-Level design): Descreve altos níveis de estrutura e organização de software’s e identificação de componentes.
- Design de software detalhado: Descreve cada componente suficientemente para permitir esta construção

A respeito do escopo da Área de Conhecimento do Design de Software (KA), a atual descrição KA não discute cada tópico do nome que contém a palavra “design.” Na terminologia de Tom DeMarco’s (DeM99), a KA discutida neste capítulo trata principalmente do D-Design (decomposição do design, mapeando o software dentro dos pedaços de componente). Entretanto, por conta dessa importância no campo de crescimento da arquitetura de software nós abordaremos o PF-Design (família de padrão de projeto, cujo o objetivo é estabelecer explorações comuns na família de software). Em contraste , a KA Design de Software não aborda I-Design (design da invenção, usualmente executada durante o processo de requisitos de software como o objetivo de conceitualizar e especificar o software para satisfazer descobertas necessárias e requisitos), desde este tópico deve ser considerado parte da especificação e análise de requisitos.

A descrição do Design de Software KA é relatada especificamente para os Requisitos do software, Construção do Software, Gerenciamento e Engenharia de Software, Qualidade do Software, e Disciplinas Relatadas com a Engenharia de Software.

2.1.3.2.1 Fundamentos do Design de Software

Os conceitos, noções e terminologia introduzidas aqui formam uma linha base para entender o papel e o escopo do design de software.

a) Conceitos Gerais de Design

O Software não é o único campo onde o design está envolvido. No senso geral, nós podemos ver design como uma forma de resolver problemas. Por exemplo, o conceito de um mau problema - um problema sem solução definitiva - é interessante em termos de entendimento dos limites do design. Uma série de outras noções e conceitos são também de interesse para a compreensão de design no seu sentido geral: objetivos, restrições, alternativas, representações e soluções.

b) Contexto do Design de Software

O Para entender o papel do design de software, é importante entender o contexto em que se enquadra o ciclo de vida da engenharia de software. Assim, é importante entender as maiores características das análises de requerimento de software vs. Design de software vs. Construção de software vs. Teste de software. [IEEE12207.0-96]; Lis01:c11; Mar02; Pfl01:c2; Pre04:c2]

c) Processo de Design de Software

O Design de software é geralmente considerado em dois passos:

- Design Arquitetural: descreve como o software é decomposto e organizado dentro de componentes (a arquitetura de software) [IEEEP1471-00]
- Design Detalhado: descreve o comportamento específico desses componentes. A saída desse processo é um conjunto de modelos e artefatos que guardam as maiores decisões que tenham sido tomadas. IEE1016-98; Lis01:c13; Pre04:c9

d) Técnicas Ativas

De acordo com o Dicionário Inglês Oxford, um princípio é “uma verdade básica ou lei geral… que é usada como a base racional ou guia de ação.” Os princípios do design de software, também são chamados de Técnicas Ativas]Bus96], são noções chave consideradas fundamentais para muitas diferentes abordagens e

conceitos de design de software. As técnicas ativas são as seguintes: Bus96:c6; IEEE1016-98; Jal97:c5,c6; Lis01:c1,c3; Pfl01:c5; Pre04:c9.

- Abstração: é “o processo de esquecer informações tais como coisas diferentes que podem ser tratadas como se elas fossem a mesma.” [Lis01] No contesto de design de software, dois mecanismos chave de abstração são a parametrização e a especificação. Abstração por especificação guia-nos aos três maiores tipos de abstração: abstração procedural, abstração de dados, e controle (iteração) de abstração. Jal97:c5,c6; Lis01:c1,c2,c5,c6; Pre04:c1
- Acoplamento e Coesão: Acoplamento é definido como uma força de relacionamento entre módulos, enquanto coesão é definido por como os elementos que formam um módulo são relacionados. Jal97:c5; Pfl01:c5; Pre04:c9
- Decomposição e modularização. Decompor e Modularizar um grande software dentro de um pequeno número de partes únicas e independentes, usualmente com o objetivo de colocar diferentes funcionalidades ou responsabilidades em diferentes componentes. [Bas98:c6, Bus96:c6; Jal97 :c5; Pfl01:c5; Pre04:c9]
- Encapsulamento / Ocultamento de Informação: significa agrupar e separar em pacotes, elementos e detalhes internos de uma abstração e fazendo esses detalhes inacessíveis. Bus96:c6; Jal97:c5; Pfl01:c5; Pre04:c9
- Separação da interface e implementação: envolve definir um componente para especificar uma interface, conhecer clientes, separando os detalhes de como os componente é ralizado.Bos00:c10; Lis01:c1,c9
- Suficiência, completude e simplicidade. Alcançar suficiência, completude e simplicidade significa garantir que um componente de software capturar todas as características importantes de uma abstração, e nada mais. Lis01:c5

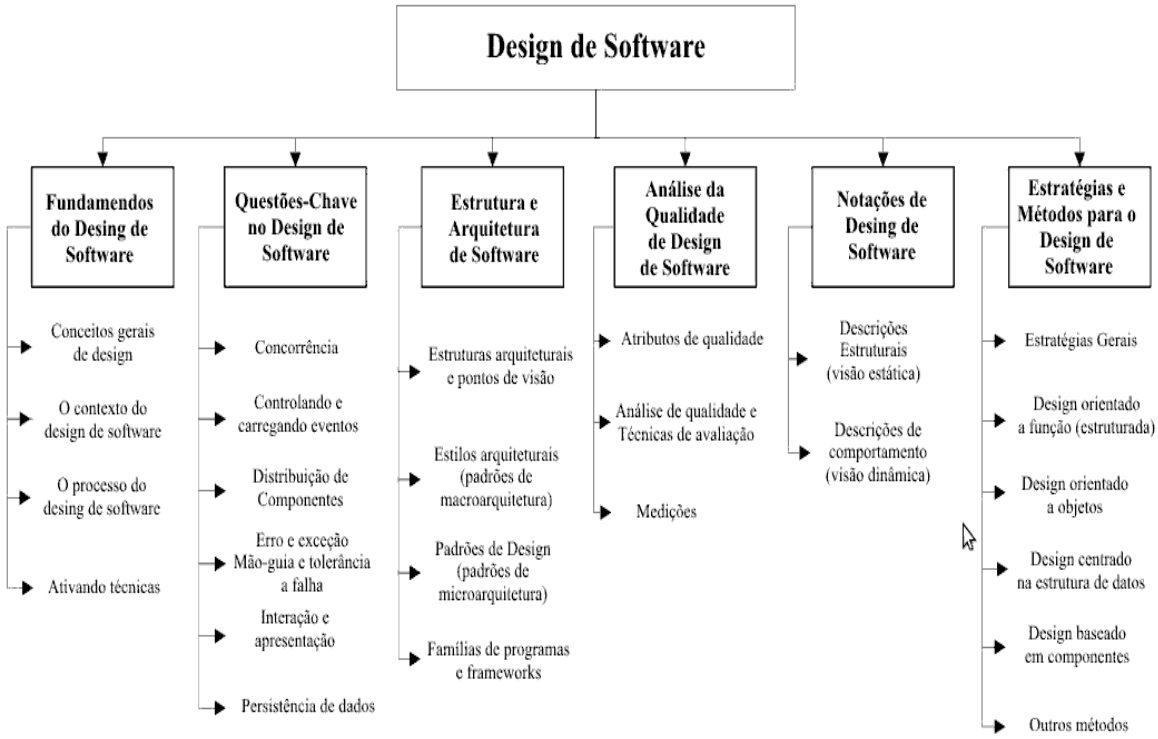


Figura 10: Tópicos para Design de Software

2.1.3.2.2 Os Pontos Chave no Design de Software

Um certo número de pontos chave devem ser tratados na concepção de software. Alguns são preocupações de qualidade. Algumas preocupações são a qualidade que todos os softwares devem ter, por exemplo, performance. Outra questão importante é a forma de decompor, organizar e empacotar componentes de software. Isso é tão fundamental que todas as abordagens endereçam isto em um caminho ou outro. Em contraste a isto, outros pontos “entregam alguns aspectos do comportamento de software que não estão no domínio da aplicação, mas desejam endereçar algo no domínio do suporte.” [Bos00] Tais pontos, desejam frequentemente cortar funcionalidades do sistema, referindo-se a aspectos: “aspectos não tendem a ser unidades de software funcionais decompostos, mas antes propriedades que afetam a performance ou semântica dos componentes em caminhos sistêmicos.

a) Concorrência

Como decompor o software em processos, tarefas e serviço e desejar eficiência, atomicidade, sincronização, e versões agendadas. Mesy97:c30; Pre04:c9

b) Controle e transporte dos Eventos

Como organizar dados e controlar fluxos, como trasportar eventos reativos e temporais através de vários mecanismos tais como invocação implícita e call-back. [Pfl01:c5]

c) Distribuição de Componentes

Como distribuir o software baseado no hardware, como os componentes se comunicam, como um middleware pode ser usado para entregar software heterogêneo. Bos00:c5; Bus96:c2 Mar94:DD; Pre04:c30

d) Transporte de Exceção, Erro e Tolerância a falha

Como prevenir e tolerar falhas e entregar condições excepcionais. [Pfl01:c5]

e) Interação e Apresentação

Como estruturar e organizar interações com usuários e a apresentação da informação (por exemplo, separação da apresentação e lógica de negócios usando a abordagem Model-View-Controller).Bos00:c5;Bus96:c2; Lis01:c13; Mey97:c32. Isto pode ser notado que este tópico não é sobre detalhes de especificação da interface de usuários, deseja-se é que a tarefa do design de interface de usuário (uma parte do Software Ergonômico); veja as Disciplinas Relatadas da Engenharia de Software.

f) Persistência dos dados

Como dados de longa vida podem ser transportados.

2.1.3.2.3 Estrutura e Arquitetura de Software

No seu sentido estrito, uma arquitetura de software é "uma descrição dos subsistemas e de componentes de um sistema de software e as relações entre eles." (Bus96:c6) Arquitetura, assim, as tentativas de definir a estrutura interna. Segundo o Dicionário Inglês Oxford, "a forma como algo é construído ou organizado"- do resultando software. Durante meados da década de 1990, no entanto, arquitetura de software começou a emergi como uma mais ampla disciplina que envolve o estudo de estruturas e de arquiteturas de software de uma forma mais genérica [Sha96].

Isso deu origem a uma série de ideias interessantes sobre a concepção do software em diferentes níveis de abstração. Alguns desses conceitos podem ser úteis durante a concepção arquitetônica (por exemplo, estilo arquitetônico) de um software específico, bem como durante a sua concepção pormenorizada (por exemplo, a concepção de nível mais baixo padrões). Mas eles também podem ser úteis para a concepção de sistemas genéricos, levando à concepção de famílias de programas (também conhecidas como linhas de produtos). Curiosamente, a maior parte destes conceitos pode ser visto como tentativas de descrever, e deste modo o reuso, conhecimentos design genéricos.

a) Estruturas arquitetônicas e Viewpoints

Diferentes facetas de high-level de design de software podem e devem ser descritos e documentados. Estes aspectos são muitas vezes chamados views: "Uma view representa um parcial aspecto de uma arquitetura de software que mostra propriedades específicas de um sistema de software" [Bus96: c6]. Estas distintas views pertencem a distintas questões relacionadas com a design de software – por exemplo, a lógica view (que satisfaça as exigências funcionais) vs os processos view (questões de concurrency) vs físico views (questões de distribuição) vs o desenvolvimento view (o modo como o projeto é dividido em unidades de implementação). Outros autores usam diferentes terminologias, como comportamental vs funcional vs estruturais vs modelagem de dados view. Em resumo, um projeto de software é um multifacetada artefato produzido pelo processo de design e geralmente composto de relativa independência e ortogonais vista. [Bas03: c2; Boo99: c31; Bus96: c6; IEEE1016-98; IEEE1471-00] estilos arquitetônicos (padrões de macroestruturas).

Um estilo arquitetônico é "um conjunto de restrições sobre um arquitetura que define um conjunto ou família de arquiteturas que os satisfazem" [Bas03: c2]. Um estilo arquitetônico pode ser visto como uma metamodelo que possa fornecer software de alto nível (high-level) para as organizações (a macro arquitetura). Diversos autores identificaram uma série de grandes estilos arquitetônicos. Boo99:c28; Bos00:c6; Bus96:c1,c6;Pfl01:c5

- Estrutura geral (por exemplo, camadas, tubos, e filtros, Blackboard);
- Sistemas Distribuídos (por exemplo, cliente-servidor, threetiers, broker);

- Sistemas Interativos (por exemplo, Model-View-Controller, Apresentação-Abstração-Controle);
- Adaptação de sistemas (por exemplo, micro-kernel, reflexão);
- Outros (por exemplo, lote, intérpretes, controle de processos, regras base).

b) Padrões de Design (padrões de micro arquitetura)

Sucintamente descritos, um padrão é "uma solução comum para um problema comum em um determinado contexto." (Jac99) Embora estilos arquitetônicos possam ser vistos como padrões que descrevem o high-level de software das organizações (as suas macro arquiteturas), outros padrões de projeto pode ser usado para descrever em detalhes uma menor, mais nível local (a sua micro arquitetura). Boo99:c28; Bus96:c1;Mar02:DP

- Padrões Creational (por exemplo, construtor, fábrica, protótipo, e Singleton)
- Padrões estruturais (por exemplo, adaptador, ponte, compósito, decorador, fachada, flyweight, e proxy)
- Padrões comportamentais (por exemplo, comando, o intérprete, interação, mediador, memento, observador, estatal, estratégia, modelo, visitante).

c) Famílias de programas e frameworks

Uma possível abordagem para permitir a reutilização de design de softwares e de componentes é a concepção das famílias de software, também conhecidos como linhas de produtos de software. Isto pode ser feito através da identificação dos membros comuns dessas famílias, e usando componentes reutilizáveis e personalizáveis para levar em conta a variabilidade entre os membros da família. [Bos00:c7,c10; Bas98:c15; Pre04:c30] .

Na programação OO, uma chave relaciona noções do que é um framework: um subsistema parcialmente completo de software que pode ser estendido pelas instalações de plug-ins mais específicos (também conhecidos como hot spots).

2.1.3.2.4 Análise e Evolução da Qualidade de Design de Software

Esta secção inclui uma série de tópicos de qualidade e avaliação que estão especificamente relacionadas com a concepção do software. A maior parte é abrangida de uma forma geral, no Quality Software KA.

a) Atributos de Qualidade

Diversos atributos são geralmente considerados importantes para a obtenção de um design de software de boa qualidade e diversas "ilities" (manutenibilidade, portabilidade, testabilidade, rastreabilidade), várias "des" (correção, robustez), incluindo "fitness de propósito." [Bos00: c5; Bus96: c6; ISO9126.1-01; ISO15026-98; Mar94: D; Mey97: c3; Pfl01: c5] Um dado interessante é a distinção entre atributos de qualidade discernível em tempo de execução (desempenho, segurança, disponibilidade, funcionalidade, usabilidade), para aqueles que não são discerníveis em tempo de execução (modificabilidade, portabilidade, reusabilidade, integrabilidade, e testabilidade), e os relacionados com as qualidades intrínsecas da arquitetura (conceitual integridade, a exatidão e a exaustividade, buildability).

b) Análise de qualidade e Técnicas de Avaliação.

Várias ferramentas e técnicas podem ajudar a garantir uma concepção da qualidade de software.

- Revisão do design de Software: semi formal ou informal, muitas vezes baseadas em grupo, técnicas para verificar e assegurar a qualidade da concepção de componente (por exemplo, a revisão da arquitetura [Bas03: C11], opiniões de design, e inspeções [Fre83: VIII; IEEE1028-97; Jal97: C5, C7; Lis01: C14; Pfl01: c5], técnicas de cenário de base [Bas98: c9; Bos00: c5], rastreio de requisitos [Dor02: v1c4s2; Pfl01: C11]);
- Análise estática: formal ou semi formal estático (não executável) análises que possam ser utilizadas para avaliar um projeto (por exemplo, análise fault-tree ou verificação cruzada automatizada) [Jal97: c5; Pfl01: c5]
- Simulação e prototipagem: técnicas dinâmicas para avaliar um projeto (por exemplo, desempenho ou simulação viabilidade de protótipo [Bas98: c10; Bos00: c5; Pfl01: c5])

c) Medidas

As medidas podem ser utilizadas para avaliar quantitativamente estimativos ou vários aspectos do tamanho do design de software, estrutura, ou qualidade. A maior parte das medidas que foram propostas dependem geralmente da abordagem

utilizada para produzir o design. Estas medidas são classificadas em duas grandes categorias:

- Medidas de Design Orientados a Função (estruturados): a concepção da estrutura, obtido principalmente através da decomposição funcional; geralmente representada como uma estrutura gráfica (às vezes chamado de diagrama hierárquico), em que várias medidas podem ser computadas [Jal97: C5, C7, Pre04 : c15]
- Medidas de Design Orientados a Objeto: a concepção global da estrutura é muitas vezes representada como um diagrama de classes, em que várias medidas podem ser computadas. Medidas sobre as propriedades de cada classe interna do conteúdo também podem ser computadas [Jal97: C6, C7; Pre04: c15]

2.1.3.2.5 Notações de Design de software

Muitas notações e linguagens existem para representar o design de software e artefatos. Algumas são usadas principalmente para descrever a organização estrutural de desenho, outras para representar o comportamento de software. Certas notações são usadas pela maior parte durante o desenho arquitetônico e outros principalmente durante detalhamento do desenho, embora algumas notações possam ser usadas em ambos os passos. Além do mais, algumas notações são usadas pela maior parte no contexto de métodos específicos (ver as Estratégias de Desenho de Software e subárea de Métodos). Aqui, eles são categorizados em notações para descrever a visão estrutural (estática) contra a visão comportamental (dinâmica).

a) Descrições Estruturais (visão estática)

As seguintes notações, pela maior parte do gráfico (mas não sempre), descrevem e representam os aspectos estruturais de um design de software isto é, eles descrevem os componentes principais e como eles são interligados (visão estática):

- Linguagens de descrição de arquitetura (ADLs): textual, muitas vezes formal, linguagens usadas para descrever uma arquitetura de software quanto aos componentes e conectores. [Bas03:c12]

- Diagramas de classe e objeto: usado para representar um jogo de classes (e objetos) e as suas relações mútuas. [Boo99:c8, c14; Jal97:c5, c6]
- Diagramas de componentes: usado para representar um jogo de componentes (“parte física e substituível [s] do sistema que [se conformam] com e [fornecem] a realização de um jogo de interfaces” [Boo99]) e suas relações mútuas [Boo99:c12, c31]
- Cartões de colaboração de classes responsáveis (CRCs): usado para denotar os nomes de componentes (classe), suas responsabilidades, e os nomes de seus componentes de colaboração Bus96
- Diagramas de desenvolvimento: usado para representar um jogo de nós (físicos) e as suas relações mútuas, e, assim, modelar os aspectos físicos de um sistema [Boo99:c30]
- Diagramas de entidades e relacionamento (ERDs): usado para representar modelos conceituais de dados fornecidos em sistemas de informação Dor02:v1c5; Mar02:DR
- Linguagens de descrição de interface (IDLs): parecido a com uma linguagem de programação usada para definir as interfaces (nomes e tipos de operações exportadas) de componentes de softwareBoo99:c11
- Estrutura de diagrama Jackson: usado para descrever os dados das estruturas quanto a sequência, seleção, e iteração Mar02:DR
- Diagramas de estrutura: usado para descrever a estrutura que chama programas (que o módulo chama, e é chamado por outro módulo) Jal97:c5; Mar02:DR;Pre04:c10

b) Descrições Comportamentais (visão dinâmica)

As seguintes notações e linguagens, alguns gráficos e alguns textuais, são usados para descrever o comportamento dinâmico de componentes de software. Muitas dessas notações são útil pela maior parte, mas não exclusivamente, durante o detalhamento do design.

- Diagramas de atividade: usado para mostrar o controle de fluxo de atividade (“execução não atômica contínua dentro de uma máquina de estado”) à atividade [Boo99:c19]
- Diagramas de colaboração: usado para mostrar as interações, isto ocorre

entre um grupo de objetos, onde a ênfase está nos objetos, as seus links, e as mensagens trocadas entre eles por esses links[Boo99:c18]

- Diagramas de fluxo de dados (DFDs): usado para mostrar fluxo de dados entre um jogo de processos Mar02:DR; Pre04:c8
- Diagramas e tabelas de decisão: usado para representar combinações complexas de condições e ações [Pre04:c11]
- Fluxogramas e fluxogramas estruturados: usado para representar o controle de fluxo e as ações associadas a ser executadas Mar02:DR; Pre04:c11
- Diagramas de sequência: usado para mostrar as interações entre um grupo de objetos, com ênfase no timeordering de mensagens [Boo99:c18]
- Transição de estado e diagramas de statechart: usado para mostrar o fluxo de controle de estado para estado em uma máquina de estado ar02:DR; Jal97:c7
- Linguagens formais de especificação: linguagens textuais utiliza noções básicas da matemática (por exemplo, a lógica, o jogo, sequência) para definir com rigor e de maneira abstrata as interfaces de componente de software e comportamento, muitas vezes em termos de pré e pós-condições Dor02:v1c6s5; Mey97:c11
- Pseudocódigo e as linguagens de design de programação(PDLs): as linguagens estruturadas-como-linguagens de programação para descrever, geralmente na etapa de detalhamento de design, o comportamento de um procedimento ou método Fre83:VII; Jal97:c7; Pre04:c8, c11

2.1.3.2.6 Estratégias e Métodos de Design de Software

Existem várias estratégias gerais para ajudar a orientar o processo de planejamento. [Mar02:D] Em contraste com as estratégias gerais, os métodos são mais específicos nisto, eles geralmente sugerem e fornecem um conjunto de notações a ser utilizadas com o método, uma descrição do processo a ser utilizado quando o seguinte método e um conjunto de orientações na utilização do método. Tais métodos são úteis como um meio de transferir conhecimento e como um framework comum de equipes de engenheiros de software.Veja também os Instrumentos de Engenharia de Software e Métodos KA.

a) Estratégias Gerais

Algumas vezes, exemplos de estratégias gerais são citados como úteis em design de processo é dividir e conquistar em um refinamento gradual Fre83:V, top-down vs. estratégias bottom-up, abstração de dados e informações ocultas[de Fre83:V], o uso da heurística, uso de modelos e linguagens de modelo Bus96:c5, uso de uma aproximação iterativa e incremental. [Pfl01:c2].

b) Design orientado por função (Estruturado)

Este é um dos métodos clássicos do design de software, onde a decomposição centra na identificação das principais funções de software e refinação elaborando-as de cima para baixo. O design estruturado é geralmente usado depois da análise estruturada, assim produção, entre outras coisas, diagrama de fluxos de dados e descrições de processo associadas. Os pesquisadores propuseram várias estratégias (por exemplo, a análise de transformação, análise de transação) e heurística (por exemplo, fan-in/fan-out, o alcance do efeito contra o alcance do controle) para transformar um DFD em uma arquitetura de software geralmente representada como um diagrama de estrutura. [Dor02:v1c6s4; Fre83:V; Jal97:c5; Pre04:c9, c10]

c) Design orientado a objeto

Numerosos métodos de design de software baseados em objetos foram propostos. O campo evoluiu com o primeiro design baseado em objeto em meados dos anos 80 (substantivo = objeto; verbo = método; o adjetivo = atributo) pelo design de OO, onde a herança e o polimorfismo desempenham um papel-chave, ao campo do design à base de componente, onde a informação de Meta pode ser definida e acessada (pela reflexão, por exemplo). Embora as raízes de design OO venham do conceito de abstração de dados, o design levado por responsabilidade também tenha sido proposto como uma aproximação alternativa ao design de OO. [Dor02:v1:c6s2,s3; Fre83:VI; Jal97:c6; Mar02:D; Pre04:c9]

d) Design centrado por estruturas de dados

O desenho centrado por estruturas de dados (por exemplo, Jackson, Warnier-Orr) partidas das estruturas de dados que um programa manipula e não da função que ele executa. O engenheiro de software primeiro descreve os dados de produção

e a entrada das estruturas (usando os diagramas de estrutura de Jackson, por exemplo) e logo desenvolvem a estrutura de controle do programa baseada nesses diagramas de estrutura de dados. Várias heurísticas foram propostas para tratar com casos, por exemplo, especiais, quando há uma má combinação entre as estruturas de produção e entrada. [Fre83:III,VII; Mar02:D]

e) Design baseado em componente

Um componente de software é uma unidade independente, tendo interfaces bem definidas e dependências que podem ser compostas e desdobradas independentemente. O design baseado em componente dirige questões relacionadas a fornecimento, desenvolvimento, e integração de tais componentes ara melhorar reutilização.

f) Outros métodos

Outros interessantes mas com enfoque menor também existem: métodos formais e rigorosos Dor02:c5; Fre83; Mey97:c11; Pre04:c29 e métodos transformacionais. [Pfl98:c2]

	[Bas03] {Bas98}	[Boo99]	[Bos00]	[Bud03]	[Bus96]	[Dor02]	[Fre83]	[IEEE1016-98]	[IEEE1028-97]	[IEEE1471-00]	[IEEE12207.0-96]	[ISO9126-01]	[ISO15026-98]	[Jal97]	[Lis01]	[Mar02]* {Mar94}	[Mey97]	[Pfl01]	[Pre04]	[Smi93]
1. Fundamentos do Design de Software																				
1.1 Conceitos Gerais de Design de Software				C1																*
1.2 O Contexto do Design de Software											*				c11s1	D		c2s2	c2	
1.3 O Processo do Design de Software	c2s1, c2s4			C2		v1c4s2	2-22	*		*	*				c13s1, c13s2	D			c9	
1.4 Ativando Técnicas	{c6s1}		c10s3		c6s3			*						c5s1, c5s2, c6s2	c1s1,c1s2, c3s1-c3s3, 77-85, c5s8, 125-128, c9s1-c9s3			c5s2, c5	c9	
2. Questões-chave no Design de Software																				
2.1 Concorrência			c5s4.1													CSD	c30		c9	
2.2 Controle e transporte de eventos	{c5s2}																c32s4, c32s5	c5s3		
2.3 Distribuição de Componentes	c16s3, c16s4		c5s4.1		c2s3											{DD}	c30		c30	
2.4 Erro e Mão-guia de exceção e Tolerância a falha															c4s3-c4s5		c12	c5s5		
2.5 Interação e apresentação	{c6s2}		c5s4.1		c2s4										c13s3		c32s2			
2.6 Persistência de dados			c5s4.1														c31			

Figura 11: Tópicos x referências (1)

	[Bas03] {Bas98}	[Boo99]	[Bos00]	[Bud03]	[Bus96]	[Dor02]	[Fre83]	[IEEE1016-98]	[IEEE1028-97]	[IEEE1471-00]	[IEEE12207.0-96]	[ISO9126-01]	[ISO15026-98]	[Jal97]	[Lis01]	[Mar02]* {Mar94}	[Mey97]	[Pfl01]	[Pre04]	[Smi93]
3. Estrutura e Arquitetura de Software																				
3.1 Arquiteturas estruturas e pontos de visão	c2s5	c31		c5	c6s1			*		*										
3.2 Estilos arquiteturais	c5s9	c28	c6s3.1		c1s1-c1s3, c6s2													c5s3		
3.3 Padrões de design	{c13s3}	c28			c1s1-c1s3											DP				
3.4 Famílias de programas e frameworks	{c15s1, c15s3}	c28	c7s1, c7s2, c10s2-c10s4, c11s2, c11s4		c6s2														C30	
4. Anásile de qualidade de design de software e avaliação																				
4.1 Atributos de qualidade	c4s2		c5s2.3	c4	c6s4							*	*			{D}	c3	c5s5		
4.2 Análise de qualidade e técnicas de avaliação	c11s3, {c9s1, c9s2, c10s2, c10s3}		c5s2.1, c5s2.2, c5s3, c5s4	c4		v1c4s2	542-576		*					c5s5, c7s3	c14s1			c5s6, c5s7, c11s5		
4.3 Medições														c5s6, c6s5, c7s4					c15	

Figura 12: Tópicos x referências (2)

	[Bas03] [Bas98]	[Boo99]	[Bos00]	[Bud03]	[Bus96]	[Dor02]	[Fre83]	[IEEE1016-98]	[IEEE1028-97]	[IEEE1471-00]	[IEEE12207.0-96]	[ISO9126-01]	[ISO15026-98]	[Jal97]	[Lis01]	[Mar02]* {Mar94}	[Mey97]	[Pfl01]	[Pre04]	[Smi93]
5. Notações de Design de software																				
5.1 Descrições estruturais (visão estática)	{c8s4} c12s1, c12s2	c4, c8, c11, c12, c14, c30, c31		c6	429									c5s3,c6s3		DR			c10	
5.2 Descrições comportamentais (visão dinâmica)		c18, c19, c24		c6, c18		v1c5	485-490, 506-513							c7s2		DR	c11		c8, c11	
6. Estratégias e Métodos de Design de software																				
6.1 Estratégias gerais				c8, c10, c12	c5s1-c5s4		304-320, 533-539							c5s1.4	c13s13			c2s2		
6.2 Design orientado a função (estruturado)				c14		v1c6s4	328-352							c5s4					c9, c10	
6.3 Design Orientado a objetos				c16		v1c6s2, v1c6s3	420-436							c6s4		D			c9	
6.4 Design centrado na estrutura de dados				C15			201-210, 514-532									D				
6.5 Design baseado em componentes (CBD)				c11																
6.6 Outros métodos				c18		181-192	395-407										c11	c2s2	C29	

Figura 13: Tópicos x referências (3)

2.1.3.3 Construção de Software

O termo construção de software refere-se a criação trabalho detalhado, significativo de software através de combinação de códigos, verificação, testes unitários, integração de teste e depuração. A área de conhecimento construção de

software tem conexão com outra KAs (áreas de conhecimentos), mais firmemente o design de software e testes de software. Isto porque processo construção de software se aplica significativo design de software e atividade de teste. Ele utiliza também a produção de design e fornece uma contribuição de teste, ambos design e teste pode ser atividades, não KAs neste caso. Detalhado limite entre design, construção, e testes irá variar dependendo do ciclo de vida do processo de software que é usado no projeto. Embora alguns detalhes de design podem ser realizados anteriores a construção, muito trabalho de design é realizado na atividade construção. Portanto, KA (área de conhecimento) construção de software está intimamente ligado KA design de software.

Durante toda construção, engenheiros de software trabalham os dois testes, de unidade e de integração. Assim, a KA construção de software está intimamente ligada a teste de software como KA. A construção de software tipicamente produz alto volume de itens de configuração, esses necessitam de gerenciamento de projeto de software. (fonte, arquivo, conteúdo, caso de teste e etc.). Portanto, a KA construção de software é também intimamente ligada a KA gerência de configuração de software.

A construção de software invoca desde ferramentas e métodos e são provavelmente ferramentas intensivas das KAs. Elas são ligadas a KA engenharia de software ferramentas e métodos. Enquanto qualidade de software é importante em todas as KAs, o código é a ultima realização do projeto de software, e, portanto qualidade de software é também intimamente ligada a construção de software.

A relação entre disciplina de engenharia de software, a KA construção de software é semelhante a ciência da computação no uso de seu conhecimento de algoritmo e de detalhamento de práticas de codificação, ambos geralmente são considerados pertencentes ao domínio da ciência da computação. Ela também está relacionada a gerencia de projeto, na medida em que a gerencia da construção pode apresentar desafios consideráveis.

A repartição da KA construção de software apresenta abaixo, junto com breve descrição dos principais tópicos associados com ela. Referências adequadas também são fornecidas para cada um dos tópicos. A figura 14 apresenta um gráfico da decomposição de nível superior da repartição para está KA.

2.1.3.3.1 Fundamentos da construção de software

Os fundamentos da construção de software inclui:

- minimizar complexidade
- antecipação da mudança
- construção para verificação
- normas de construção

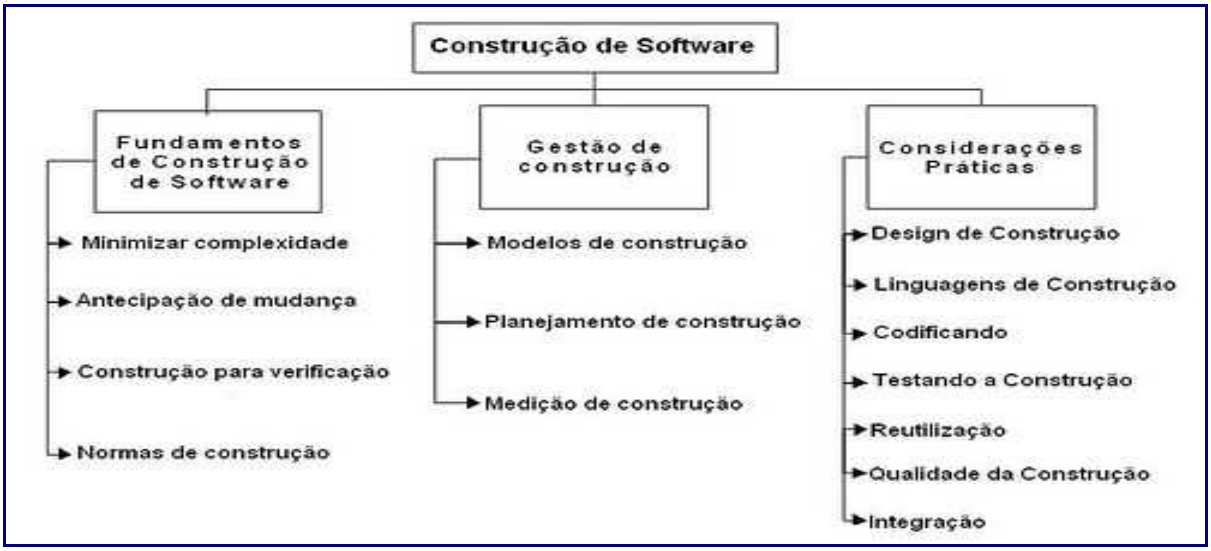


Figura 14: Repartição dos tópicos para o área de conhecimento de construção de software

Os três primeiros conceitos se aplicam ao design, bem como a construção. As seguintes seções definem estes conceitos e descreve como eles se aplicam na construção.

a) Minimizar complexidade

Um fator principal em como as pessoas transmitirem as intenções dos computadores limitações serias das capacidades das pessoas em realizar complexas estruturas e informações em suas memórias de trabalho, especialmente durante longos períodos de tempo. Isto conduz a um forte controle em construção de software. Minimizar complexidade. O necessário de reduzir aplicações complexas basicamente de todos os aspectos da construção de software, e é particularmente critico dos processos de verificação e testes de construção de software. Em construção de software, reduzir complexidade é obter por enfatiza a criação de código que é simples e legível e bastante engenhoso. Minimizar complexidade é realizada por meio da utilização de padrões, que é discutido no tópico 1.4 Padrão em

construção, e por numerosas técnicas especificas resumidas no tópico 3.3 codificação. Ele é também apoiado pela construção com foco na técnica de qualidade resumida no tópico 3.5 qualidade de construção.

b) Antecipação de mudança

A maior parte dos software irá mudar com o tempo, e a antecipação das mudanças controla muitos aspectos da construção de software. A mutação de ambientes externos de software é parte inevitável, e as mudanças nesses ambientes externos afetam o software em diversas maneiras. As antecipação da mudança é apoiada por muitas técnicas específicas resumidas no tópico 3.3 codificação.

c) Construção para verificação

Construção para verificação meio de construir software de forma que as falhas podem ser prontamente encontradas pelo engenheiro de software que escreveu o software, bem como durante os testes independentes e atividades operacionais. Técnicas especificas de suporte a construção para verificação inclui seguinte normas de codificação de suporte a revisão de código, teste de unidades, organização de código de suporte automação de testes, e restrição de uso complexo ou linguagem de difícil compreensão das estruturas, entre outras.

d) Normas de construção

Normas que afetam diretamente questões de construção incluem:

- Métodos de comunicação (por exemplo, padrão de formatos de documentação e contexto);
- Linguagem de programação (por exemplo, padrões de linguagem com linguagens Java e C++);
- Plataforma (por exemplo, programador padrão para a interface do sistema operacional chamado);
- Ferramenta (por exemplo, diagramas padrão para notações ligadas a UML (Linguagem de Modelagem Unificada));

Uso de normas externas. Construção depende do uso de normas externas para construção de linguagens, construção de ferramentas, técnicas de interfaces, e interações entre construção de software e outras KAs. Normas vêm de numerosas

fontes, incluindo hardware e interface de especificação de software tais como o Grupo Gestão Objeto (OMG) e organizações internacionais tais como a IEEE ou ISO.

Uso de normas interna. Normas também podem ser criadas sobre uma base organizacional no nível corporativo ou para utilização em projetos específicos. Estas normas apoiam a coordenação de grupos atividades, minimizar complexidade, antecipação de mudança, e construção para verificação.

2.1.3.3.2 Gestão de Construção

a) Modelos de construção

Inúmeros modelos têm sido criados para desenvolver software, algumas dos quais enfatizam construção mais do que outros.

Alguns modelos são mais lineares e parti do ponto de construção e visualização, tal como o modelo de ciclo de vida cascata. Estes modelos trata construção como uma atividade na qual ocorre apenas depois de significativo trabalho de detalhamento de requisitos foi concluído, extenso trabalho de design, e detalhe de planejamento. O mais linear tende abordar para enfatizar a atividade esse precede construção (requisitos e design), e tende para criar mais clara separação entre as atividades. Nestes modelos, a ênfase principal pode ser a codificação.

Outros modelos são mais interativos, tal como prototipagem evolutiva, Extreme Programming (XP) e Scrum. Estes tende a abortar e tratar construção como uma atividade que ocorre concomitantemente com outras atividades de desenvolvimento de software, incluindo requisitos, design, e planejamento, ou sobrepor-lhes. Estes tende abordar para misturar design, codificação, e atividades de testes, e eles muitas vezes tratam a combinação de atividades como construção. Consequentemente, o que é considerado “construção” depende em certa medida do ciclo de vida utilizado.

b) Planejamento de construção

A escolha de métodos de construção são aspectos chaves da atividade planejamento de construção. A escolha de métodos de construção afeta o grau a qual pré-requisitos de construção são realizados, a ordem em qual eles são realizados, e o grau para qual eles são esperados para ser concluídos antes de

começar os trabalhos de construção.

A abordagem de construção do projeto afeta capacidade para reduzir complexidades, antecipar mudanças, e verificação para construção. Cada um desses objetivos pode também ser abordado no processo, requisitos, e níveis de design mas eles também serão influenciados pela escolha dos métodos de construção.

Planejamento da construção também define a ordem em qual os componentes são criados e entregados, a gerencia de processo de qualidade de software, a distribuição de tarefas específicas atribuídas aos engenheiros de software, e a outras tarefas, segundo o método escolhido.

c) Medição de construção

Inúmeras atividades de construção e artefatos podem ser medidas, incluindo desenvolvimento de código, modificação de código, reutilização de código, destruição de código, complexidade de código, inspeção estatísticas de código, a falhas e corrigir falhas, esforço, e agendamento. Estas medições podem ser úteis para propósito de gerencia da construção, assegurar qualidade durante a construção, melhorar o processo de construção, bem como por outros motivos. Ver a KA processo de engenharia de software para mais sobre medições.

2.1.3.3.3 Considerações práticas

Construção é uma atividade na qual o software tem que se condicionar as restrições arbitrarias e caóticas do mundo real e, fazê-la precisamente. Devido à sua proximidade a restrições do mundo real, construção está mais orientada por considerações práticas do que algumas áreas de conhecimento, e a engenharia de software assemelhasse a esta área de construção

a) Design de Construção

Alguns projetos reservam mais atenção para o design de construção; outros em uma fase explícita focam o plano. Independente do momento, alguns detalhes do design ocorrerão em nível de construção, e este design de trabalho será ditado pelas restrições imóveis impostas pelo mundo real, problemas que são abordados pelo software. Tal como trabalhadores da construção que constroem uma estrutura física

devem fazer modificações, contabilizar pequenas falhas no plano do construtor, construtores de software devem fazer modificações em escala menor ou maior para enriquecer detalhes do design de software durante sua construção
Os detalhes da atividade de design a nível de construção são, essencialmente, os mesmos como descrito na área de conhecimento de design de software, mas foram aplicadas numa escala menor.

b) Linguagem de Construção

Linguagens de construção incluem todas as formas de comunicação em que um homem pode especificar um solução de um problema executável para o computador.

O tipo mais simples de linguagem de construção é uma linguagem de configuração, na qual engenheiros de software escolhem um conjunto limitado de opções predefinidas para criar instalações de software novas ou personalizadas.

Os arquivos de configuração baseado em texto usados nos sistemas operacionais Windows e UNIX são exemplos, e as listas de seleção de estilo de menu de alguns geradores de programa constituem outro. Kit de ferramentas de linguagem são usadas para criar aplicativos em kits (conjuntos integrados de partes reutilizáveis específicas do aplicativo) e são linguagens de configuração mais complexas. Kit de ferramentas de linguagem, podem ser explicitamente definidas como linguagens de programação de aplicativo (por exemplo, scripts) ou podem simplesmente ser implícito no conjunto de interfaces de um kit de ferramentas.

Linguagens de programação são o tipo mais flexível das linguagem de construção. Elas também contém o mínimo de informações sobre áreas de aplicação específicas e processos de desenvolvimento e assim o exigem mais formação e habilidade para utilizá-las de forma eficaz. Existem três tipos gerais de notação usada para linguagens de programação, que são:

- Linguística
- Formal
- Visual

Notações linguísticas distinguem-se em especial através da utilização de cadeias de palavras chave de texto para representar construções de software complexo e a combinação de tais strings de palavra chave em padrões que possuem

uma sintaxe de frase semelhante. Corretamente utilizados, cada cadeia de caracteres deve ter uma forte conotação semântica fornecendo uma compreensão intuitiva imediata do que acontecerá quando a construção de software subjacente é executada.

Notações formais contam com menos significados intuitivos, palavras comuns e cadeias de texto, e, mais em definições de suporte preciso, sem ambiguidades e definições formais (ou matemática). Notações de construção formal e os métodos formais, são o centro da maioria das formas de sistema de programação, onde a precisão, comportamento de tempo e testabilidade são mais importantes, do que a facilidade de mapear para linguagem natural. Construções formais também usam precisamente formas definidas de combinar os símbolos, que evitam a ambiguidade de muitas linguagens natural de construção. Notações Visual baseiam-se muito menos sobre as notações de palavras chaves da construção linguística e formal e em vez disso dependem a interpretação visual direta e o posicionamento das entidades visual que representam o software subjacente. Construção Visual tende a ser um pouco limitado pela dificuldade de fazer “complexas“ declarações usando apenas o movimento das entidades visuais numa exibição. No entanto, também pode ser uma ferramenta poderosa nos casos em que a tarefa de programação principal é simplesmente construir e “ajustar“ uma interface visual a um programa, comportamento detalhado do que foi definido anteriormente.

c) Codificando

As considerações seguintes se aplicam à codificação na atividade de construção software: [Ben00; IEEE12207-95; McC04]

- Técnicas para criar código fonte compreensível, incluindo a nomeação e layout de código de origem.
- Utilização de classes, tipos enumerados, variáveis, constantes nomeados e outras entidades similares.
- Uso do controle estruturas.
- Manutenção das condições de erro — tanto planejadas erros e excepções (entrada de dados incorreto, por exemplo).
- Prevenção das violações de segurança de nível de código (saturações de buffer ou matriz índice estouros, por exemplo).

- Uso de recursos através da utilização de mecanismos de exclusão e disciplina no acessar série de recursos reutilizáveis (incluindo threads ou bloqueios de banco de dados).
- Organização de código de origem (em declarações, rotinas, classes, pacotes ou outras estruturas).
- Documentação de código.
- Ajuste de código

d) Testando a Construção

Construção envolve duas formas de testes, que frequentemente são executadas pelo engenheiro de software que escreveu o código: [Bec99; Hun00; Mag93; McC04]

- Testes de unidade
- Testes de integração.

O objetivo dos testes de construção visa reduzir a lacuna entre o tempo em que os defeitos são inseridos no código e o tempo que os mesmos são detectados. Em alguns casos, testes de construção são executados depois de código foi escrito. Em outros casos, os testes podem ser criados antes que código seja escrito.
Testes de construção envolvem tipicamente um subconjunto de tipos de testes, que são descritos na área de conhecimento de teste de software. Por exemplo, testes de construção normalmente não incluem testes de sistema, testes alfa, teste beta, stress testing, teste de configuração, testes de usabilidade ou outros, mais tipos especializados dos testes.

Duas normas foram publicadas sobre o tema: padrão IEEE 829-1998, padrão de IEEE para documentação de teste de software e padrão de IEEE 1008-1987, padrão de IEEE para testes de unidade de software.

e) Reutilização

Tal como consta a introdução de (IEEE1517-99): “Executando reutilização de software implica mais do que criar e usar bibliotecas de recursos. É necessário formalizar a prática de reutilização, integrando os processos de reutilização e atividades no ciclo de vida do software.“ No entanto, a reutilização é suficientemente importante na construção de software, e que é incluída aqui como um tópico.

As tarefas relacionadas com a reutilização na construção de software durante a codificação e testes são: [IEEE1517-99; Som05]

- A seleção das unidades reutilizáveis, bancos de dados, procedimentos de ensaio ou testar dados.
- A avaliação da reutilização de código ou de teste
- A comunicação de informações de reutilização sobre o novo código, testar procedimentos ou testar dados.

f) Qualidade da Construção

Numerosas técnicas existem para garantir a qualidade de como o código é construído. As principais técnicas utilizadas para construção incluem:[Bec99; Hun00; IEEE12207-95; Mag93; McC04]

- Unidade ensaios e testes de integração (tal como mencionado no tópico 3.4 testes de construção)
- Primeiro teste de desenvolvimento (ver igualmente o teste de software na área de conhecimento, tópico 2.2 objetivos de teste)
- Código de reforço
- Utilização de afirmações
- Depuração
- Revisões técnicas(ver igualmente o software qualidade na área de conhecimento, análises técnicas do sub-tópico 2.3.2)
- Análise estática (IEEE1028) (ver igualmente na área de conhecimento de qualidade de software, tema 2.3 revisões e auditorias).

A técnica específica ou técnicas selecionadas dependem da natureza do software a ser construído, bem como dos engenheiros de software no conjunto habilidades executando a construção. Atividades de qualidade de construção são diferenciadas das outras atividades de qualidade pelo seu foco. Atividades de qualidade de construção concentram-se no código e não nos artefatos que estão intimamente relacionados ao código: desenhos pequenos — em vez de outros artefatos que estão menos diretamente conectados com o código, como requisitos, design de alto nível e planos.

g) Integração

Uma atividade fundamental durante a construção é a integração das rotinas construídas separadamente, classes, componentes e subsistemas. Além disso, um sistema de software específico talvez precise ser integrado com outros sistemas de software ou hardware.

Preocupações relacionadas com a integração da construção incluem planejamento a sequência em que os componentes serão integrados, criando suporte para apoiar a versões provisórias do software, determinar o grau de testes e trabalho de qualidade realizado nos componentes antes de serem integrados e em que determinantes pontos no projeto, versões provisórias do software são testadas. [Bec99; IEEE12207-95; McC04]

2.1.3.4 Teste de Software

Teste de software é uma atividade executada para avaliar a qualidade do produto, e para melhorá-lo, pela identificação de defeitos e problemas. Teste de software consiste da verificação dinâmica do comportamento de um programa em um conjunto finito de casos de teste, adequadamente selecionados de um domínio de execução usualmente infinito, contra o comportamento esperado. Na definição anterior, as palavras em itálico correspondem a assuntos chave na identificação de Área do Conhecimento de Teste de Software. Em particular:

- Dinâmico: Esse termo significa que testar sempre implica executar o programa com entradas (valorizadas). Para ser preciso, só os valores de entrada não são sempre suficientes para determinar um teste, desde que um sistema complexo, não determinístico pode reagir à mesma entrada com diferentes comportamentos, dependendo do estado do sistema. Nessa área do conhecimento, embora o termo “entrada” será mantido, com a convenção incluída que seu significado também inclua um estado específico de entrada, nesses casos para os quais isso é necessário. Diferente de testar e complementar a isso, são as técnicas estáticas descritas na Área de Conhecimento Qualidade de Software.
- Finito: Até mesmo em programas simples, tantos casos de testes são teoricamente possíveis que testes exaustivos possam exigir meses ou anos para executar. Isso é por que na prática o conjunto inteiro de testes pode geralmente ser considerado infinito. Testes sempre implicam em uma troca

entre recursos limitados e programados de um lado, e exigências inerentemente ilimitadas de testes de outro.

- Selecionado: As muitas técnicas propostas de teste diferem essencialmente em como elas selecionam o conjunto de teste, e engenheiros de software precisam estar atentos que critérios de seleção diferentes podem render diferentes graus de efetividade. Como identificar o critério de seleção mais satisfatório em determinadas condições é um problema muito complexo; na prática, técnicas de análise de risco e perícias de engenharia de testes são aplicadas.
- Esperado: Deve ser possível, embora nem sempre fácil, decidir se os resultados observados da execução do programa são aceitáveis ou não, caso contrário, o esforço de teste seria inútil. O comportamento observado pode ser conferido contra expectativas de usuário (comumente chamado teste para validação), contra uma especificação (teste para verificação), ou finalmente, contra o comportamento antecipado de exigências implícitas ou expectativas razoáveis.

A visão de testes de software tem evoluído para uma mais construtiva. Teste já não é mais visto como uma atividade que começa somente depois que a fase de codificação está completa, com a finalidade limitada de detecção de falhas. Teste de software agora é visto como uma atividade que deve abranger todo o processo de desenvolvimento e manutenção e ele é parte importante da construção atual do produto. Realmente, o planejamento para teste deveria começar nos estágios iniciais dos processos de requisitos, e planos de teste e procedimentos precisam ser sistematicamente e continuamente desenvolvidos, e possivelmente refinados, como desenvolvimento contínuo. Estes planos de teste e atividades de planejamento constituem contribuição útil para desenvolvedores realçando potenciais debilidades (como descuidos ou contradições, e omissões ou ambiguidades na documentação).

É considerado atualmente que a atitude certa em direção à qualidade é uma de prevenção: é obviamente muito melhor evitar problemas que corrigi-los. Testes devem ser vistos, então, principalmente como meios para conferir não só se a prevenção foi efetiva, mas também para identificar faltas nesses casos onde, por alguma razão, não foi efetivo. É talvez óbvio mas com reconhecido valor que, até mesmo depois da conclusão de uma extensiva campanha de testes, que o software

ainda possa conter falhas. O remédio para falhas experimentadas de software depois da entrega é feito por ações de manutenção corretiva. Tópicos de Manutenção de Software estão cobertos na Área de Conhecimento Manutenção de Software.

Na Área de Conhecimento Qualidade de Software, técnicas de administração de qualidade de software são categorizadas notavelmente em técnicas estáticas (nenhuma execução de código) e técnicas dinâmicas (execução de código). Ambas as categorias são úteis. Esta Área de Conhecimento focaliza em técnicas dinâmicas.

Teste de software também é relacionado a construção de software (veja tópico 3.3). Testes de unidade e integração são relacionados intimamente com a construção de software, e não parte disso.

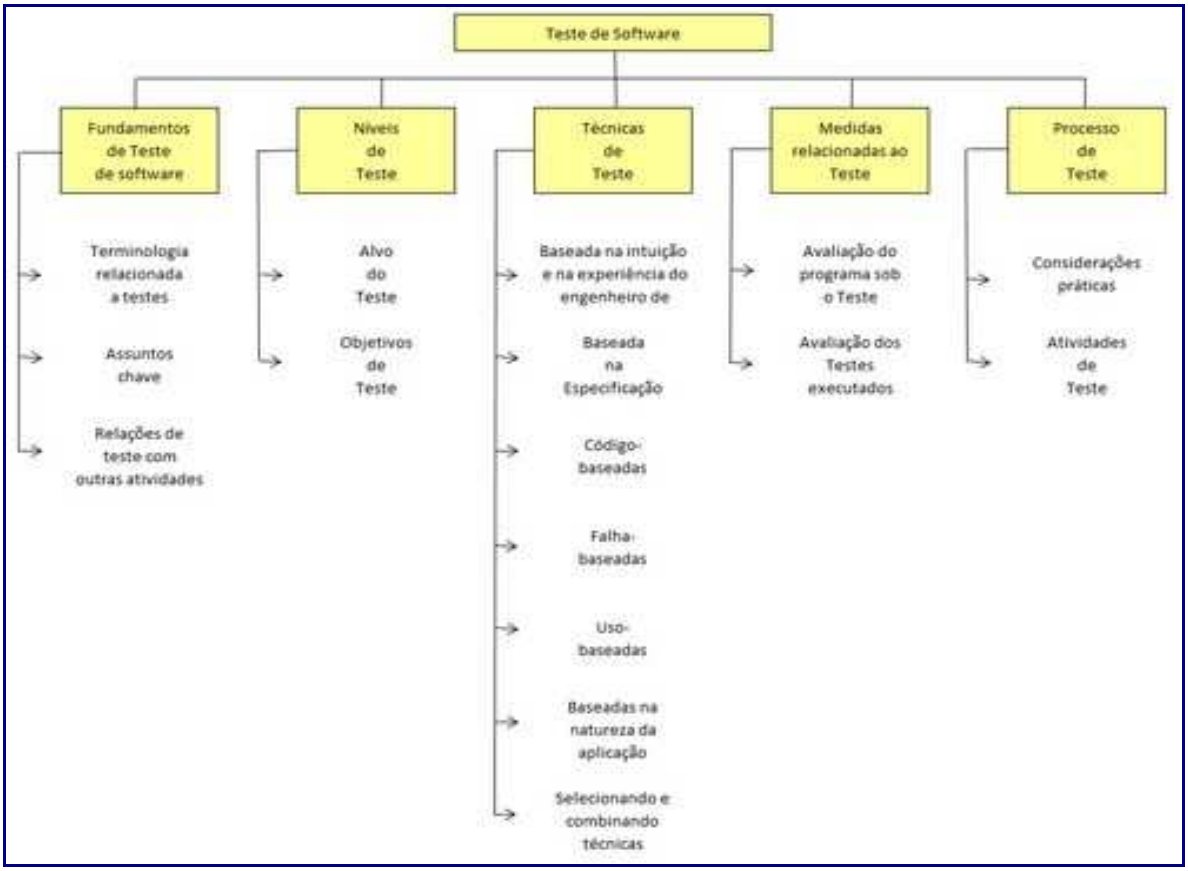


Figura 15: Tópicos para Área de Conhecimento Teste de Software

2.1.3.4.1 Fundamentos de Teste de Software

Muitos termos são usados na literatura da engenharia de software para descrever um mau funcionamento, falha notável, fracasso, erro, e outros vários.

Essa terminologia é precisamente definida em IEEE Standard 610.12-1990, Standard Glossary of Software Engineering Terminology (IEEE610-90), e isso é

discutido também na área de conhecimento Qualidade de Software. Isso é essencial para distinguir claramente entre a causa de um mau funcionamento, para o qual o termo falha ou defeito será usado aqui, e um indesejável efeito observado no serviço entregue do sistema, o qual será chamado um fracasso. Testes podem revelar fracassos, mas são as falhas que podem e devem ser removidas. Porém, deveria ser reconhecido que a causa de um fracasso não pode ser sempre inequivocamente identificado. Não existe um critério teórico para determinar definitivamente que falta causou o fracasso observado. Poderia ser dito que era a falta que tinha de ser modificada para remover o problema, mas outras modificações poderiam ter trabalhado bem da mesma maneira. Para evitar ambiguidades, alguns autores preferem falar de entradas causando fracassos (Fra98) em vez de falhas, quer dizer, esses conjuntos de entradas que causam um fracasso.

a) Critérios de seleção de testes / critérios de suficiência de Teste (ou regras de parada)

Um critério de seleção de testes é um meio de decidir qual conjunto satisfatório de casos de testes deve ser. Um critério de seleção de testes pode ser usado para selecionar os casos de testes ou checar qual suite de testes é adequada, quer dizer, decidir se o teste pode ser parado. Veja também as terminações do sub tópico, no tópico 5.1 Considerações práticas. Zhu97:s1.1 (Wey83; Wey91; Zhu97)

b) Efetividade do teste / Objetivos de teste

Testar é a observação de uma amostra de execuções do programa. Seleções de amostras podem ser guiadas por diferentes objetivos: isso é somente na luz do objetivo procurado que a efetividade do conjunto de teste pode ser avaliada. [Per95:c21] (Fra98)

c) Teste para identificação de defeito

Nos testes para identificação de defeito, um teste de sucesso é um que causa a falha do sistema. Isso é bastante diferente dos testes para demostrar que o software satisfaz suas especificações ou outras propriedades desejadas, nesse caso, o teste é de sucesso se nenhuma falha (significante) é observada. [Kan99:c1]

d) O problema do oráculo

Um oráculo é qualquer agente (humano ou mecânico) que decide se um programa se comportou corretamente dentro de um determinado teste, e consequentemente produz um veredito de “passe” ou “falhe”. Existem muitos tipos de oráculos e a automação de oráculos pode ser muito difícil e cara. [Bei90:c1] (Ber96, Wey83)

e) Limitações teóricas e práticas de teste

A teoria de teste adverte contra atribuir nível injustificado de confiança para uma série de testes passados. Infelizmente, a maioria dos resultados estabelecidos de teoria de teste são negativos, na condição de que um teste pode nunca alcançar ao invés de que eles atualmente alcançam. A citação mais famosa nessa consideração é o provérbio de Dijkstra que “teste de programa pode ser usado para provar a presença de bugs, mas nunca para mostrar sua ausência”. A razão óbvia é que o teste completo de software não é possível em software real. Por causa disto, o teste precisa ser dirigido baseado no risco e pode ser visto como uma estratégia de administração de risco. [Kan99:c2] (How76)

f) O problema dos caminhos inviáveis

Caminhos inviáveis, os caminhos de controle de fluxo que não podem ser exercitados por qualquer dados de entrada, são um significante problema no teste de caminho orientado, e particularmente na derivação automatizada de testes de entrada para técnicas de teste código baseadas. [Bei90:c3]

g) Testabilidade

O termo “testabilidade de software” tem dois significados diferentes mas relacionados: por um lado, se refere ao grau a que se é fácil para o software cumprir um critério de cobertura de teste, como em (Bac90); por outro lado, está definida como a probabilidade, possivelmente medida estatisticamente, que o software irá expor uma falha sob o teste, se está defeituoso, como em (Voa98, Ber96a). Ambos os significados são importantes. [Bei90:c3, c13] (Bac90; Ber96a; Voa95)

h) Relações de teste com outras atividades

Teste de software é relacionado a, mas diferente de técnicas estáticas de administração de qualidade de software, provas de exatidão, depuração e programação. Porém, é informativo considerar os testes do ponto de vista de analistas de qualidade de software e de certificadores. Teste versus técnicas estáticas de administração da qualidade de software. Veja também a Área de Conhecimento Qualidade de Software, sub área 2. Processos de administração da qualidade do software. Per95:c17 (IEEE1008-87). Teste versus provas de exatidão e verificação formal Pfl01:c8. Teste versus depuração. Veja também a Área de Conhecimento Construção de Software, tópico 3.4 Teste de construção. [Bei90:c1s2.1] (IEEE1008-87). Teste versus programação. Veja também a Área de Conhecimento Construção de Software, tópico 3.4 Teste de construção. [Bei90:c1s2.3]. Teste e certificação (Wak99).

2.1.3.4.2 Níveis de Teste

a) alvo do teste

O teste de software é executado geralmente a níveis diferentes ao longo dos processos do desenvolvimento e da manutenção. Isso é para dizer, o alvo do teste pode variar: um único módulo, um grupo de tais módulos (relacionados pela finalidade, uso, comportamento, ou estrutura), ou um sistema inteiro. [Bei90: c1; Jor02: c13; Pfl01: c8] Três estágios grandes de teste podem ser conceitualmente distinguidos, são eles: unidade, integração, e sistema. O modelo do processo não é significativo, nem é suposto que algum daqueles três estágios tenha maior importância do que os outros dois.

- teste de unidade - [Bei90: c1; Per95: c17; Pfl01: c8s7.3] (IEEE1008-87) O teste de unidade verifica o funcionamento isoladamente das partes do software que são testáveis separadamente. Dependendo do contexto, estas podiam ser os subprogramas ou os grandes componentes feitos de unidades firmemente relacionadas. Um teste de unidade é definido mais precisamente no padrão IEEE para Teste de unidade de software (IEEE1008-87), que também descreve uma aproximação integrada para testes de unidade sistemáticos e documentados. Tipicamente, o teste de unidade ocorre com acesso ao código que está sendo testado e com o suporte de ferramentas de depuração de erros, e força o envolvimento dos programadores que

escreveram o código.

- teste de integração - [Jor02: c13, 14; Pfl01: c8s7.4] O teste da integração é o processo de verificar a interação entre os componentes de software. Estratégias clássicas de teste de integração, tais como de cima pra baixo ou de baixo para cima, são usadas com software tradicional, hierarquicamente estruturado. As estratégias modernas de integração sistemáticas são um pouco guiadas pela arquitetura, implica que a integração dos componentes de software ou subsistemas baseiam-se em linhas de identificação funcional. O teste da integração é uma atividade contínua, em cada estágio no qual os engenheiros de software devem abstrair perspectivas de baixo nível distantes e concentrar nas perspetivas do nível que estão integrando. À exceção de software pequeno, simples, sistemático, as estratégias de teste de integração incremental, são geralmente preferíveis para colocar todos os componentes juntos imediatamente, que é chamado pictorialmente de teste do“big bang”.
- teste de sistema - Pfl01:c9 O teste do sistema é estado relacionado com o comportamento de um sistema inteiro. A maioria das falhas funcionais deve já ter sido identificada durante os testes de unidade e de integração. O teste do sistema é considerado geralmente apropriado para comparar o sistema aos requisitos de sistema não funcionais, tais como a segurança, a velocidade, a exatidão, e a confiabilidade. As interfaces externas a outras aplicações, as utilidades, os dispositivos de hardware, ou o ambiente operacional são avaliadas neste nível também. Ver a Área de Conhecimento requisitos de software para mais informação sobre requisitos funcionais e não funcionais

b) objetivos de teste - Pfl01:c9s8.3

O teste é conduzido na perspectiva de um objetivo específico, que é fixado mais ou menos explicitamente, e com vários graus de precisão. Fixando o objetivo em termos precisos e quantitativos, permite que o controle seja estabelecido sobre o processo do teste. O teste pode ser apontado na verificação de propriedades diferentes. Os casos de teste podem ser projetados para a verificação da correta execução das especificações funcionais, que é referido variadamente na literatura como teste de conformidade, teste de exatidão, ou teste funcional. Entretanto, diversas outras propriedades não funcionais podem ser testadas também, incluindo

o desempenho, a confiabilidade, e a usabilidade, entre muitas outras. Outros objetivos importantes para o teste incluem (mas não se limitam) a medida da confiabilidade, a avaliação da usabilidade, e a aceitação, para que as aproximações diferentes sejam tomadas. Note que o objetivo do teste varia com o alvo de teste; geralmente, finalidades diferentes que estão sendo endereçadas a diferentes níveis de teste. As referências recomendadas acima para este tópico descrevem o conjunto de potenciais objetivos de teste. As sub-tópicos listados abaixo são aqueles mencionados mais frequentemente na literatura. Note que alguns tipos de teste são mais apropriados para pacotes de software feitos sob medida, teste de instalação, por exemplo; e outros para produtos genéricos, como teste beta.

- Teste de aceitação e qualificação - Pfl01:c9s8.5 (IEEE12207.0-96:s5.3.9) O teste de aceitação verifica o comportamento do sistema de encontro aos requisitos do cliente, porém estes devem ter sido expressados; os clientes empreendem, ou especificam, tarefas típicas para verificar se seus requisitos estão sendo cumpridos ou que a organização tenha identificado estes (requisitos) para o objetivo mercadológico do software. Esta atividade de teste pode ou não pode envolver os desenvolvedores do sistema.
- teste de instalação - Pfl01:c9s8.6 Geralmente após a conclusão do teste do software e de aceitação, o software pode ser verificado em cima da instalação no ambiente alvo. O teste de instalação pode ser visto como teste do sistema conduzido mais uma vez de acordo com requisitos de configuração de hardware. Os procedimentos de instalação também podem ser verificados.
- teste alfa e beta - [Kan99:c13] Antes que o software seja liberado, às vezes ele é fornecido a um grupo pequeno mas representativo de potenciais usuários para o uso experimental, interno (teste alfa) ou o externo (teste beta). Estes usuários relatam problemas com o produto. O uso alfa e beta é frequentemente descontrolado, e não é referido sempre dentro de um plano de teste.
- teste de conformidade/teste funcional/teste de exatidão - Per95:c8 (Wak99) O teste de conformidade tem por objetivo validar se o comportamento do software testado conforma-se a suas especificações.
- Realização e avaliação da confiabilidade - Pfl01:c9s.8.4 (Pos96) Ajudando na identificação de falhas, testar é um meio de melhorar a confiabilidade. Pelo

contraste, pela geração aleatória de casos de teste de acordo com o perfil operacional, as medidas estatísticas de confiabilidade podem ser derivadas. Usando modelos incrementais de confiabilidade, ambos os objetivos podem ser levados a cabo juntos (ver o sub tópico 4.1.4 teste de vida, avaliação da confiabilidade).

- teste de regressão - Per95:c11, c12; Pfl01:c9s8.1 (Rot96) De acordo com (IEEE610.12-90), o teste de regressão é “reteste seletivo de um sistema ou de um componente para verificar que modificações não causaram efeitos não intencionais…” Na prática, a ideia é mostrar que esse software que passou previamente nos testes ainda passa. Beizer (Bei90) define como qualquer repetição de teste que pretende mostrar que o comportamento do software é inalterado, exceto quando necessário. Obviamente um trade-off deve ser feito entre a garantia dada pelo teste de regressão cada vez que uma mudança é feita e os recursos exigiram fazer isso. O teste de regressão pode ser conduzido em cada um dos níveis do teste descritos no tópico 2.1 alvo do teste e pode aplicar-se ao teste funcional e não funcional
- teste de performance - Pfl01:c9s8.3 (Wak99) Este tem por objetivo específico verificar se o software cumpre as exigências específicas de desempenho, por exemplo, por instância, capacidade e tempo de resposta. Um tipo específico do teste de desempenho é o teste de volume (Per95: p185, p487; Pfl01: p401), em que as limitações internas do programa ou do sistema são experimentadas.
- teste de stress - Pfl01:c9s8.3 O teste de stress força o software na carga máxima de projeto, assim como além dela.
- teste back-to-back - Um único conjunto de teste é executado em duas versões implementadas de um produto de software, e os resultados são comparados.
- teste de recuperação - Pfl01:c9s8.3 O teste de recuperação tem o objetivo de verificar as capacidades de reinício do software após um “desastre”.
- teste de configuração - Pfl01:c9s8.3 Nos casos onde o software é construído para servir usuários diferentes, o teste da configuração analisa o software sob as várias configurações específicas.
- teste de usabilidade - Pfl01:c9s8.3 Este processo avalia o quanto é fácil para

que os utilizadores finais usem e aprendam o software, incluindo a documentação de usuário; como efetivamente o software funciona no apoio de tarefas do usuário; e, finalmente, sua habilidade de se recuperar de erros do usuário.

- teste dirigido de desenvolvimento - [Bec02] O teste dirigido de desenvolvimento não é uma técnica de teste por si só, promovendo o uso de testes como um substituto para um documento de especificação de requisitos, como uma verificação independente se o software implementou corretamente os requisitos.

2.1.3.4.3 Técnicas de Teste

Um dos objetivos do teste é revelar tanto potencial para a falha como possível, e muitas técnicas foram desenvolvidas para fazer isso, que tentam “parar” o programa, rodando um ou vários testes esboçados das classes identificadas de execuções julgadas equivalentes. O princípio fundamental que é a base de tais técnicas é ser tão sistemático como possível em identificar um conjunto representativo de comportamentos do programa; por exemplo, considerando subclasses do domínio de entrada, cenário, estados, e fluxo de dados. É difícil encontrar uma base homogênea para classificar todas as técnicas, e essa usada aqui deve ser vista como um compromisso. A classificação é baseada em como os testes são gerados da intuição e da experiência do engenheiro de software, as especificações, a estrutura do código, (real ou artificial) falhas a serem descobertas, o uso de campos, ou, finalmente, a natureza da aplicação. Às vezes estas técnicas são classificadas como caixa branca, igualmente chamada caixa transparente, se os testes confiam na informação sobre como o software foi projetado ou codificado, ou como caixa-preta se os casos de teste confiam somente no comportamento de entrada/saída. Uma última categoria trata o uso combinado de duas ou mais técnicas. Obviamente, estas técnicas não são usadas muitas vezes igualmente por todos os engenheiros. São incluídas na lista aquelas que um engenheiro de software deve saber.

a) baseadas na intuição e na experiência da engenheiro de software

- teste ad hoc [Kan99: c1] Talvez a técnica mais extensamente praticada

permaneça sendo o teste ad hoc: os testes são derivados confiando na habilidade do engenheiro de software, intuição, e experiência com programas similares. O teste ad hoc pode ser útil para identificar testes especiais, aqueles não facilmente capturados por técnicas formalizadas.

- Teste exploratório - O teste exploratório é definido simultaneamente como aprendizagem, o projeto do teste, e a execução do teste; isto é, os testes não são definidos antecipadamente em um plano de teste estabelecido, mas são dinamicamente projetados, executados, e modificados. A eficácia do teste exploratório confia no conhecimento do engenheiro de software, que pode ser derivado de várias fontes: comportamento observado do produto durante o teste, familiaridade com a aplicação, a plataforma, o processo da falha, o tipo de falhas e fracassos possíveis, o risco associado com um produto em particular, e assim por diante. [Kan01: c3]

b) técnicas baseadas na Especificação

- * 3.2.1. Divisão equivalente [Jor02: c7; Kan99: c7] - O domínio de entrada é subdividido em uma coleção de subconjuntos, ou classes equivalentes, que são julgadas equivalente de acordo com uma relação específica, e um conjunto representativo dos testes (às vezes somente um) são tomados de cada classe.
- Análise do valor limite [Jor02: c6; Kan99: c7] - Os casos de teste são escolhidos próximo e nos limites do domínio da entrada das variáveis, com a razão subjacente que muitas falhas tendem a se concentrar perto dos valores extremos de entradas. Uma extensão desta técnica é teste de vigor, onde as situações de teste são escolhidas igualmente fora do domínio da entrada das variáveis, para testar o vigor do programa para entradas inesperadas ou errôneas.
- Tabela de decisão [Bei90: c10s3] (Jor02) - As tabelas de decisão representam relacionamentos lógicos entre condições (aproximadamente, entradas) e ações (aproximadamente, saídas). Os casos de teste são derivados sistematicamente considerando cada combinação possível de condições e de ações. Uma técnica relacionada é a representação gráfica de causa-efeito. [Pfl01: c9]
- Baseada em máquina de estado finito Jor02:c8 - Modelando um programa

como uma máquina de estado finito, os testes podem ser selecionados a fim cobrir estados e transições nela.

- Teste das especificações formais [Zhu97: s2.2] (Ber91; Dic93; Hor95) - Dar as especificações em uma linguagem formal permite a derivação automática de casos de teste funcionais, e, ao mesmo tempo, fornece uma saída de referência, um oráculo, para verificar os resultados do teste. Os métodos existem derivando casos de teste de especificações modelo baseadas (Dic93, Hor95) ou algébricas. (Ber91)
- Teste aleatório [Bei90: c13; Kan99: c7] - Os testes são gerados puramente em aleatório, para não ser confundidos com o teste estatístico do perfil operacional como descritos no sub tópico 3.5.1 perfil operacional. Esta forma de teste cai sob o título da entrada de especificação baseada, desde que pelo menos o domínio da entrada seja conhecido, para poder escolher pontos aleatórios dentro dele.

c) técnicas código baseadas

- Critérios ipecaconhas-de-flor-branca [Bei90: c3; Jor02: c10] (Zhu97) - Os critérios de cobertura ipecaconhas-de-flor-branca tem o objetivo de cobrir todas as indicações ou blocos de indicações em um programa, ou combinações específicas deles. Diversos critérios de cobertura foram propostos, como a cobertura circunstância/decisão. O mais forte dos critérios controle de fluxo baseados é o teste do trajeto, que tem por objetivo executar todos os trajetos do fluxo de controle de entrada a saída no gráfico de fluxo. Desde que o teste do trajeto não seja geralmente praticável por causa dos laços, outros critérios menos estritos tendem a ser usados na prática, como o teste de declaração, o teste de ramificação, e o teste de condição/decisão. A suficiência de tais testes é medida nas porcentagens; por exemplo, quando todas as ramificações foram executadas pelo menos uma vez pelos testes, a cobertura da ramificação de 100% é dita ter sido conseguida.
- Critérios fluxo de dados baseados [Bei90: c5] (Jor02; Zhu97) - No teste fluxo de dados baseado, o gráfico de fluxo de controle é anotado com informação sobre como as variáveis do programa são definidas, usadas, e destruídas (indefinidas). O critério mais forte, trajetos de uso definidos, necessita que,

para cada variável, todo segmento do trajeto do fluxo de controle de uma definição de qual variável para um uso de qual definição seja executada. A fim de reduzir o número de trajetos exigidos, estratégias mais fracas tais como todas as definições e todos osusos são empregadas.

- Modelos de referência para o teste código baseado (gráfico de fluxo, grafo chamado) [Bei90: c3; Jor02: c5]. - Embora não seja uma técnica em si, a estrutura de controle de um programa é representada graficamente usando um gráfico de fluxo nas técnicas de teste código baseadas Um gráfico de fluxo é um grafo dirigido de nós e arcos que correspondem aos elementos do programa. Por exemplo, os nós podem representar declarações ou sequências ininterruptas de declarações, e arcos a transferência de controle entre os nós.

d) Técnicas falha baseadas

Com graus diferentes de formalização, as técnicas de teste falha baseadas planejam as situações de teste visadas especificamente revelando categorias de falhas predefinidas ou prováveis. (Mor90)

- Suposição do erro [Kan99: c7] - No suposição do erro, as situações de teste são projetadas especificamente pelas engenheiros de software que tentam externar as falhas mais plausíveis em um programa dado. Uma boa fonte de informação é a história das falhas descobertas em projetos prematuros, assim como a perícia do engenheiro de software.

- Teste de mutação [Per95: c17; Zhu97: s3.2-s3.3] - Um mutante é uma versão ligeiramente modificada do programa sob o teste, diferindo dele por uma mudança pequena, sintática. Cada caso de teste exercita ambos o original e todos os mutantes gerados: se um caso de teste é bem sucedido em identificar a diferença entre o programa e um mutante, o último seria “destruído”. Concebido originalmente como uma técnica para avaliar um caso do teste (ver 4.2), teste de mutação é igualmente um critério de teste por si só: outros testes são gerados aleatoriamente até que mutantes suficientes estejam destruídos, ou testes são projetados especificamente para destruir mutantes sobreviventes. No último caso, o teste da mutação pode igualmente ser categorizado como uma técnica código baseada A

suposição subjacente do teste de mutação, o efeito do acoplamento, é que procurando as falhas sintáticas simples, mais falhas complexas porém reais serão encontradas. Para que a técnica seja eficaz, um grande número de mutantes deve ser automaticamente derivado de uma maneira sistemática.

e) Técnicas uso baseadas

- Perfil operacional [Jor02: c15; Lyu96: c5; Pfl01: c9] - No teste para a avaliação da confiabilidade, o ambiente de teste deve reproduzir o ambiente operacional dos software tão próximo como possível. A ideia é para inferir, dos resultados da análise observados, a confiabilidade futura do software quando no uso real. Para fazer isto, entradas são atribuídas numa distribuição de probabilidade, ou num perfil, de acordo com sua ocorrência na operação real.
- Teste projetado da confiabilidade de software [Lyu96: c6] - O teste projetado da confiabilidade de software (SRET) é um método de teste que abrange o processo de desenvolvimento inteiro, por meio de que o teste “é projetado e guiado por objetivos de confiabilidade e uso relativo e criticidade previstos de funções diferentes no campo.”

f)Técnicas baseadas na natureza da aplicação

As técnicas acima aplicam-se a todos os tipos de software. Entretanto, para alguns tipos das aplicações, algum "know-how" adicional é exigido para a derivação do teste. Uma lista de alguns campos de teste especializados é fornecida aqui, baseada na natureza da aplicação sob o teste:

- Teste orientado ao objeto [Jor02: c17; Pfl01: c8s7.5] (Bin00)
- Teste componente baseado
- Teste web baseado
- Teste de GUI [Jor20]
- Teste dos programas simultâneos (Car91)
- Teste de conformidade do protocolo (Pos96; Boc94)
- Teste de sistemas de tempo real (Sch94)
- Teste dos sistemas segurança críticos (IEEE1228-94)

g) Selecionando e combinando técnicas

- Funcional e estrutural [Bei90: c1s.2.2; Jor02: c2, c9, c12; Per95: c17] (Pos96) - As técnicas de teste baseadas na especificação e código baseadas são contrastadas frequentemente como testes funcionais versus estruturais. Estas duas aproximações para seleção de teste não devem ser vistas como alternativas mas como bastante complementares; de fato, usam fontes de informação diferentes e tem demonstrado destacar tipos diferentes de problemas. Poderiam ser usadas em combinação, dependendo das considerações orçamentais.
- Determinísticas versus aleatórias (Ham92; Lyu96: p541-547) - Os casos de teste podem ser selecionados de uma maneira determinística, de acordo com uma das várias técnicas listadas, ou selecionados aleatoriamente de alguma distribuição de entradas, como é feito geralmente no teste de confiabilidade. Diversas comparações analíticas e empíricas foram conduzidas para analisar as condições que fazem uma aproximação mais eficaz do que a outra.

2.1.3.4.4 Medidas Relacionadas ao Teste

Às vezes, as técnicas de teste são confundidas com os objetivos do teste. As técnicas de teste devem ser vistas como auxílio que ajudam a assegurar a realização de objetivos do teste. Por exemplo, a cobertura da ramificação é uma técnica de teste popular. Conseguir uma medida específica da cobertura da ramificação não deve ser considerada o objetivo do teste por si só: é um meio para melhorar as possibilidades de encontrar falhas sistematicamente exercitando cada ramificação do programa fora de um ponto de decisão. Para evitar tais enganos, uma distinção clara deve ser feita entre as medidas teste relacionadas, que fornecem uma avaliação do programa sob o teste baseado nas saídas observadas do teste, e as aquelas que avaliam a eficácia do conjunto do teste. A informações adicionais em medida de programas é fornecida na Área de conhecimento da gerência da engenharia de software, sub área 6, medição da engenharia de software. A informações adicionais em medidas pode ser encontrada na Área de conhecimento do processos de engenharia de software, na sub área 4, processos e medida do produto. A medida é considerada geralmente instrumental à análise da qualidade. A medida pode igualmente ser usada para aperfeiçoar o planejamento e execução dos testes. A gerência de teste pode usar diversas medidas de processos

para monitorar o progresso. As medidas relativo ao processo do teste para finalidades de gerência são consideradas no tópico 5.1 considerações práticas.

a) Avaliação do programa sob o teste

- Medidas do programa para auxiliar no planejamento e no teste do projeto [Bei90: c7s4.2; Jor02: c9] (Ber96; IEEE982.1-88) - As medidas baseadas no tamanho do programa (por exemplo, linhas de código fonte ou pontos de função) ou na estrutura do programa (como a complexidade) são usadas para guiar o teste. As medidas estruturais podem igualmente incluir medidas entre os módulos de programa nos termos da frequência com que os módulos chamam um ao outro. (IEEE982.1-98)
- Tipos, classificação, e estatísticas de falha [Bei90: c2; Jor02: c2; Pfl01: c8] (Bei90; IEEE1044-93; Kan99; Lyu96) - A literatura do teste é rica nas classificações e nas taxonomias das falhas. Para fazer o teste mais efetivo, é importante saber que tipos de falhas poderiam ser encontrados no software sob o teste, e a frequência relativa com que estas falhas ocorreram no passado. Esta informação pode ser muito útil em fazer predições de qualidade, assim como para a melhoria do processo. Mais informação pode ser encontrada na Área de conhecimento qualidade de software, no tópico 3.2 caraterização do defeito. Existe um padrão IEEE em como classificar “anomalias” do software (IEEE1044-93).
- Densidade de falha [Per95: c20] (IEEE982.1-88; Lyu96: c9) - Um programa sob teste pode ser avaliado pela contagem e classificação das falhas descobertas pelos seus tipos. Para cada classe de falha, a densidade da falha é medida como a relação entre o número de falhas encontradas e o tamanho do programa.
- Teste de vida, avaliação da confiabilidade [Pfl01: c9] (Pos96: p146-154) - Uma estimativa estatística da confiabilidade de software, que possa ser obtida pela realização e pela avaliação da confiabilidade (ver sub tópico 2.2.5), pode ser usada para avaliar um produto e para decidir se o teste pode ser parado ou não.
- Modelos de crescimento da confiabilidade [Lyu96: c7; Pfl01: c9] (Lyu96: c3,

c4) Os modelos de crescimento da confiabilidade fornecem uma predição da confiabilidade baseada nas falhas observadas sob a realização e a avaliação da confiabilidade (ver o sub tópico 2.2.5). Eles assumem, geralmente, que as falhas que causaram os fracassos observados foram reparadas (embora alguns modelos igualmente aceitam reparos imperfeitos), e assim, em média, a confiabilidade do produto exibe uma tendência de aumento. Existem agora dúzias de modelos publicados. Muitos são colocados em algumas suposições comuns, enquanto outros diferem. Notavelmente, estes modelos são divididos em modelos de contagem de fracassos e modelos tempo entre fracassos.

b) Avaliação dos testes executados

- Medidas de cobertura/eficácia [Jor02: c9; Pfl01: c8] (IEEE982.1-88) - Diversos critérios de suficiência de teste exigem que os casos de teste exercitem sistematicamente um conjunto de elementos identificados no programa ou nas especificações (ver a sub tópico 3). Para avaliar a eficácia dos testes executados, os verificadores podem monitorar os elementos cobertos, de modo que possam dinamicamente medir a relação entre os elementos cobertos e seu número total. Por exemplo, é possível medir a porcentagem de ramificações cobertas no gráfico de fluxo do programa, ou dos requisitos funcionais exercidos entre aquelas listadas no documento das especificações. Os critérios código baseados de suficiência exigem a instrumentação apropriada do programa sob o teste.
- Semeando falhas [Pfl01: c8] (Zhu97: s3.1) - Algumas falhas são introduzidas artificialmente no programa antes do teste. Quando os testes são executados, alguma destas falhas semeadas serão reveladas, e possivelmente algumas falhas que já haviam lá também serão. Na teoria, dependendo de qual das falhas artificiais são descobertas, e de quantas, o teste de eficácia pode ser avaliado, e o número restante de falhas genuínas pode ser estimado. Na prática, os estatísticos questionam a distribuição e a representatividade das falhas semeadas relativas às falhas genuínas e do pequeno tamanho de amostra em que todas as extrapolações são baseadas. Alguns igualmente discutem que esta técnica deve ser usada com grande cuidado, desde que a

introdução de falhas no software envolve o risco óbvio de deixá-las lá.

- Contagem da mutação [Zhu97: s3.2-s3.3] - No teste da mutação (ver o sub tópico 3.4.2), a relação de mutantes destruídos com o número total de mutantes gerados pode ser uma medida da eficácia do conjunto de teste executado.
- Comparação e eficácia relativa das diferentes técnicas [Jor02: c9, c12; Per95: c17; Zhu97: s5] (Fra93; Fra98; Pos96: p64-72) - Diversos estudos foram conduzidos para comparar a eficácia relativa de diferentes técnicas de teste. É importante ser preciso a respeito da propriedade contra a qual as técnicas estão sendo avaliadas; que, por exemplo, é dado o significado exato ao termo "eficácia"? As interpretações possíveis são: o número de testes necessários para encontrar o primeiro fracasso, a relação do número de falhas encontradas por meio do teste para todas as falhas encontradas durante e após o teste, ou quanto a confiabilidade foi melhorada. As comparações analíticas e empíricas entre técnicas diferentes foram conduzidas de acordo com cada um das noções da eficácia especificadas acima.

2.1.3.4.5 Processo de Teste

Os conceitos, as estratégias, as técnicas, e as medidas do teste precisam de ser integrados em um processo definido e controlado que funcione através de pessoas. O processo de teste suporta atividades de teste e fornece a orientação às equipes do teste, do plano de teste para o teste de avaliação da saída, de modo a oferecer a garantia justificada de que os objetivos do teste sejam encontrados de maneira rentável.

a) Considerações práticas

- Atitudes/programação sem ego [Bei90: c13s3.2; Pfl01: c8] - Um componente muito importante do teste bem sucedido é uma atitude colaborativa para o teste e as atividades de garantia da qualidade. Os gerentes têm um papel chave em promover uma recepção geralmente favorável para a descoberta da falha durante o desenvolvimento e a manutenção; por exemplo, impedindo uma atitude mental da posse de código entre programadores, de modo que não sintam responsáveis pelas falhas reveladas por seu código.

- Guias de teste [Kan01] - As fases de teste podiam ser guiadas por vários alvos, por exemplo: no teste risco baseado, que usa os riscos do produto para dar a prioridade e focalizar a estratégia do teste; ou no teste cenário baseado, em que os casos de teste são definidos baseados em cenários específicos do software.
- Teste da gestão de processo [Bec02: III; Per95: c1-c4; Pfl01: c9] (IEEE1074-97; IEEE12207.0-96: s5.3.9, s5.4.2, s6.4, s6.5) - O teste das atividades conduzidas a níveis diferentes (ver sub tópico 2 níveis de teste) deve ser organizado, junto com as pessoas, ferramentas, políticas, e medidas, em um processo bem definido que seja uma parte integrante do ciclo de vida. No IEEE/EIA Standard 12207.0, o teste não é descrito como um processo autônomo, mas os princípios para atividades do teste são incluídos junto com os cinco processos preliminares do ciclo de vida e o processo de suporte. No IEEE STD 1074, o teste é agrupado com outras atividades da avaliação como integrante ao ciclo de vida inteiro.
- Teste de documentação e produtos do trabalho de [Bei90: c13s5; Kan99: c12; Per95: c19; Pfl01: c9s8.8] (IEEE829-98) - A documentação é uma parte integrante da formalização do processo de teste. O padrão IEEE para teste de documentação de Software (IEEE829-98) fornece uma boa descrição de documentos de teste e de seus relacionamentos uns com os outros e com o processo do teste. Os documentos de teste podem incluir, entre outros, o plano de teste, a especificação do projeto do teste, a especificação do procedimento de teste, a especificação dos casos de teste, o registro de teste, e o relatório do incidente ou problema do teste. O software sob o teste é documentado como item do teste. A documentação de teste deve ser produzida e continuamente atualizada, ao mesmo nível de qualidade que outros tipos de documentação na engenharia de software.
- Equipe de teste interna versus independente [Bei90: c13s2.2-c13s2.3; Kan99: c15; Per95: c4; Pfl01: c9] - a Formalização do processo de teste pode envolver formalizar a organização da equipe de teste também. A equipe do teste pode se compor de membros internos (isto é, na equipe de projeto, envolvido ou não na construção do software), de membros externos, na esperança de trazer em uma perspetiva imparcial, independente, ou,

finalmente, de membros internos e externos. As considerações de custos, de agendamento, de níveis da maturidade das organizações envolvidas, e de criticidade da aplicação podem determinar a decisão.

- A estimação do custo/esforço e outras medidas do processo [Per95: c4, c21] (Per95: Apêndice B; Pos96: p139-145; IEEE982.1-88) - Diversas medidas relativas aos recursos gastos no teste, assim como à eficácia relativa da busca de falhas das várias fases de teste, são usadas por gerentes para controlar e melhorar o processo do teste. Estas medidas do teste podem cobrir tais aspetos como número de casos de teste específicos, número de casos de teste executados, número de casos de teste passados, e número de casos de teste que falharam, entre outros. A avaliação de relatórios da fase de teste pode ser combinada com a análise origem causa para avaliar a eficácia do processo de teste em encontrar falhas o mais cedo possível. Tal avaliação pode ser associada com a análise dos riscos. Além disso, os recursos que gastam valor no teste devem ser proporcionais com o uso/criticidade da aplicação: as técnicas diferentes têm custos diferentes e rendem níveis diferentes de confiança na confiabilidade do produto.
- Término [Bei90: c2s2.4; Per95: c2] - Uma decisão deve ser tomada se a quantidade de teste é suficiente e quando um estágio de teste pode ser terminado. Medidas de eficácia, como a cobertura conseguida do código ou a integralidade funcional, assim como estimativas da densidade das falhas ou da confiabilidade operacional, fornecem sustentação útil, mas não são suficientes em si mesmas. A decisão igualmente envolve considerações sobre os custos e os riscos incorridos pelo potencial para falhas restantes, ao contrário dos custos implicados na continuidade do teste. Ver também sub tópico 1.2.1 critérios de seleção de testes /suficiência de testes.
- Reuso de teste e padrões de teste [Bei90: c13s5] - Para realizar o teste ou a manutenção de uma maneira organizada e custo efetiva, os meios usados para testar cada parte do software devem ser reusados sistematicamente. Este repositório de materiais do teste deve estar sob o controle da gerência de configuração do software, de modo que as mudanças nos requisitos do software ou o projeto possam ser refletidos nas mudanças no escopo dos testes conduzidos. As soluções de teste adotadas no teste de alguns tipos de

aplicação sob determinadas circunstâncias, com as motivações atrás das decisões tomadas, dão forma a um padrão de teste que pode ser documentado para reuso posterior em projetos similares.

b) Atividades de teste

Sob este tópico, é dada uma breve vista geral de atividades de teste; como implicado frequentemente pela seguinte descrição, a gerência bem sucedida de atividades do teste depende fortemente do processo da gerência de configuração do software.

- Planejamento [Kan99: c12; Per95: c19; Pfl01: c8s7.6] (IEEE829-98: s4; IEEE1008-87: s1-s3) - Como todos os outros aspetos do gerenciamento do projeto, as atividades de teste devem ser planejadas. Os aspectos chave do planeamento de teste incluem a coordenação dos pessoal, a gerência das facilidades de teste e dos equipamentos disponíveis (que podem incluir meios magnéticos, planos de teste e procedimentos), e o planejamento para possíveis resultados indesejáveis. Se mais de uma linha de base do software está sendo mantida, então uma consideração principal do planeamento é o momento e o esforço necessários para assegurar-se de que o ambiente de teste esteja ajustado à configuração apropriada.
- Geração de casos de teste [Kan99: c7] (Pos96: c2; IEEE1008-87: s4, s5) - A geração de casos de teste é baseada no nível de teste a ser executado e das técnicas de teste particulares. As situações de teste devem estar sob o controle da gerência de configuração do software e incluir os resultados previstos para cada teste.
- Desenvolvimento do ambiente de testes [Kan99: c11] - O ambiente usado para o teste deve ser compatível com as ferramentas da tecnologia de programação. Deve facilitar o desenvolvimento e o controle dos casos de teste, assim como o registro e a recuperação de resultados previstos, de manuscritos, e de outros materiais do teste.
- Execução [Bei90: c13; Kan99: c11] (IEEE1008-87: s6, s7) - A execução dos testes deve personificar um princípio básico de experimentação científica: tudo feito durante o teste deve ser executado e documentado bem claramente

para que uma outra pessoa possa reproduzir os resultados. Consequentemente, o teste deve ser executado de acordo com procedimentos documentados usando uma versão claramente definida do software sob o teste.

- Avaliação dos resultados da análise [Per95: c20, c21] (Pos96: p18-20, p131-138) - Os resultados do teste devem ser avaliados para determinar se o teste foi mesmo bem sucedido ou não. Na maioria dos casos, “bem sucedido” significa que o software executou como esperado e não teve nenhum resultado principal inesperado. Não todos os resultados inesperados são necessariamente falhas, entretanto, mas poderia ser julg para ser simplesmente o ruído. Antes que uma falha possa ser removida, uma análise e um esforço na eliminação de erros é necessário para isolá-la, identificá-la, e descrevê-la. Quando os resultados do teste são particularmente importantes, um conselho de revisão formal pode ser reunido para avaliá-los.
- Relatório do problema/registro do teste [Kan99: c5; Per95: c20] (IEEE829-98: s9-s10) - As atividades do teste podem ser incorporadas a um registro do teste para identificar quando um teste foi conduzido, quem executou o teste, que configuração de software foi a base para o teste, e outras informações de identificação relevantes. Os resultados da análise inesperados ou incorretos podem ser gravados em um sistema de relatório de problemas, os dados que formam a base para uma posterior depuração e reparação de problemas que foram observados como falhas durante o teste. Também, as anomalias não classificadas como falhas poderiam ser documentadas no caso de se mostrarem posteriormente mais sérias do que pensado inicialmente. Os relatórios de teste também são uma entrada ao processo de pedido de gerência da mudança (ver Área de conhecimento gerenciamento de configuração de software, sub tópico 3, de controle de configuração do software).
- Procurando por defeito [Kan99: c6] – Os fracassos observadas durante o teste são mais frequentemente devido a falhas ou defeitos no software. Tais defeitos podem ser analisados para determinar quando foram introduzidos no software, que tipo de erro fez com que eles fossem criados (requisitos mal definidos, declaração incorreta de variáveis, estouro de memória, erro na

sintaxe de programação, por exemplo), e quando poderiam primeiramente ter sido observados no software. A informação da procura por defeito é usada para determinar que aspetos da engenharia de software precisam melhorar e como foram eficazes as análises e testes precedentes.

	[Bec02]	[Bei90]	[Jor02]	[Kan99]	[Kan01]	[Lyu96]	[Per95]	[Pfl01]	[Zhu97]
1. Fundamentos de teste de software									
1.1 Terminologia relacionada a teste									
1.1.1 Definição de teste e terminologia relacionada		c1	c2			c2s2.2			
1.1.2 Falhas versus fracassos			c2			c2s2.2	c1	c8	
1.2 Assuntos chave									
1.2.1 Critérios de seleção de testes/critérios de suficiência de testes								c8s7.3	s1.1
1.2.2 Efetividade do teste		c1s1.4					c21		
1.2.3 Teste para identificação de defeito		c1		c1					
1.2.4 O problema do oráculo		c1							
1.2.5 Limitações teóricas e práticas de teste				c2					
1.2.6 O problema dos caminhos inviáveis		c3							
1.2.7 Testabilidade		c3,c13							
1.3 Relações de teste com outras atividades									
Teste versus técnicas estáticas de administração da qualidade de software		c1					c17		
Teste versus provas de exatidão e verificação formal		c1s5						c8	
Teste versus depuração		c1s2.1							
Teste versus programação		c1s2.3							
Teste e certificação									
2. Níveis de teste									
2.1 Alvo do teste		c1	c13					c8	
Teste de unidade		c1					c17	c8s7.3	
Teste de integração			c13,c14					c8s7.4	
Teste de sistema			c15					c9	
2.2 Objetivos de teste							c8	c9s8.3	
Teste de aceitação e qualificação							c10	c9s8.5	
Teste de instalação							c9	c9s8.6	
Teste alfa e beta				c13					
Teste de conformidade/Teste funcional/Teste de exatidão				c7			c8		
Realização e avaliação da confiabilidade						c7		c9s8.4	
Teste de regressão				c7			c11, c12	c9s8.1	
Teste de performance							c17	c9s8.3	
Teste de stress							c17	c9s8.3	
Teste back-to-back									
Teste de recuperação							c17	c9s8.3	
Teste de configuração								c9s8.3	
Teste de usabilidade				c8			c8	c9s8.3	
Teste dirigido de desenvolvimento	III								

Figura 16: Tópicos x referências (4)

	[Bec02]	[Bei90]	[Jor02]	[Kan99]	[Kan01]	[Lyu96]	[Per95]	[Pfl01]	[Zhu97]
3. Técnicas de teste									
3.1 Baseadas na intuição e na experiência do engenheiro de software									
Teste ad hoc				c1					
Teste Exploratório					c3				
3.2 Técnicas baseadas na especificação									
Divisão equivalente			c7	c7					
Análise do valor limite			c6	c7					
Tabela de decisão		c10s3						c9	
Baseada em máquina de estado finito		c11	c8						
Teste das especificações formais									s2.2
Teste aleatório		c13		c7					
3.3 Técnicas código-baseadas									
Critérios controle-de-fluxo baseados		c3	c10					c8	
Critérios fluxo-de-dados baseados		c5							
Modelos de referência para o teste código-baseado		c3	c5						
3.4 Técnicas falha-baseadas									
Suposição do erro				c7					
Teste de mutação							c17		s3.2, s3.3
3.5 Técnicas uso-baseadas									
Perfil operacional			c15			c5		c9	
Teste projetado da confiabilidade de software						c6			
3.6 Técnicas baseadas na natureza da aplicação									
Teste orientado ao projeto			c17					c8s7.5	
Teste componente-baseado									
Teste web-baseado									
Teste de GUI			c20						
Teste dos programas simultâneos									
Teste de conformidade do protocolo									
Teste de sistemas distribuídos									
Teste de sistemas de tempo real									
3.7 Selecionando e combinando técnicas									
Funcional e estrutural		c1s2.2	c1, c11s11.3				c17		
Determinísticas versus aleatórias									

Figura 17: Tópicos x referências (5)

	[Bec02]	[Bei90]	[Jor02]	[Kan99]	[Kan01]	[Lyu96]	[Per95]	[Pfl01]	[Zhu97]
4. Medidas relacionadas ao teste									
4.1 Avaliação do programa sob o teste									
Medidas do programa para auxiliar no planejamento e no teste do projeto		c7s4.2	c9						
Tipos, classificação e estatísticas de falha		c2	c1					c8	
Densidade de falha							c20		
Teste de vida, avaliação da confiabilidade								c9	
Modelos de crescimento da confiabilidade						c7		c9	
4.2 Avaliação dos testes executados									
Medidas de cobertura/eficácia			c9					c8	
Semeando falhas								c8	
Contagem da mutação									s3.2, s3.3
Comparação e eficácia relativa das diferentes técnicas			c8,c11				c17		s5
5. Processo de teste									
5.1 Considerações práticas									
Atitudes/programação sem ego		c13s3.2						c8	
Guias de teste	III				C5				
Teste da gestão de processo							c1-c4	c9	
Teste de documentação e produtos do trabalho		c13s5		c12			c19	c9s8.8	
Equipe de teste interna versus independente		c13s2.2, c1s2.3		c15			c4	c9	
A estimação do custo/esforço e outras medidas do processo							c4, c21		
Término		c2s2.4					c2		
Reuso de teste e padrões de teste		c13s5							
5.2 Atividades de teste									
Planejamento				c12			c19	c87s7.6	
Geração de casos de teste				c7					
Desenvolvimento do ambiente de testes				c11					
Execução		c13		c11					
Avaliação dos resultados da análise							c20, c21		
Relatório do problema/registro do teste				c5			c20		
Procurando por defeito				c6					

Figura 18: Tópicos x referências (6)

2.1.3.5 Manutenção de Software

Os esforços no desenvolvimento de software resultam em uma entrega de um produto de software que satisfaça as necessidades dos utilizadores. Consequentemente, os produtos de software devem mudar ou evoluir. Uma vez em operação, defeitos são descobertos, as operações de ambientes mudam, e as novas necessidades aparecem. A fase de manutenção do ciclo de vida começa após o período de garantia ou após a implementação da entrega do suporte, mas as atividades de manutenção ocorrem muito mais cedo. A manutenção de software é uma parte integral do ciclo de vida do software. Entretanto, ela não tem, historicamente, recebido o mesmo grau de atenção que as outras fases têm. Historicamente, o desenvolvimento de software tem tido um perfil muito mais elevado do que a manutenção de software, na maioria das organizações. Isto agora está mudando, como as organizações se esforçando para espremer ao o máximo o seu investimento em desenvolvimento de software para manter a manutenção de software em operação o maior tempo possível. Preocupações sobre o Ano 2000 (Y2K) geraram uma significativa atenção focada sobre a fase manutenção de software, e os paradigmas dos Códigos Abertos trouxeram ainda mais atenção à questão dos artefatos de manutenção de software desenvolvida pelos outros. No Guia, manutenção de software é definida como a totalidade das atividades necessárias para fornecer o melhor custo-benefício ao suporte de software. As atividades são realizadas durante a fase pré entrega, bem como durante o período após a entrega. As atividades pré entrega incluem o planejamento para as operações pós-entrega, operações estas de manutenção, logística e de determinação de transição de atividades. As atividades pré entrega incluem modificação de software, formação e operação da interface de suporte. A área de conhecimento de Manutenção de Software está relacionada a todos os outros aspectos da engenharia de software. Portanto, essa descrição de área de conhecimento está ligada a todos os outros capítulos do Guia.

Repartição dos tópicos de Manutenção de software

A repartição dos tópicos da área de conhecimento de Manutenção de Software é apresentada na Figura 19.

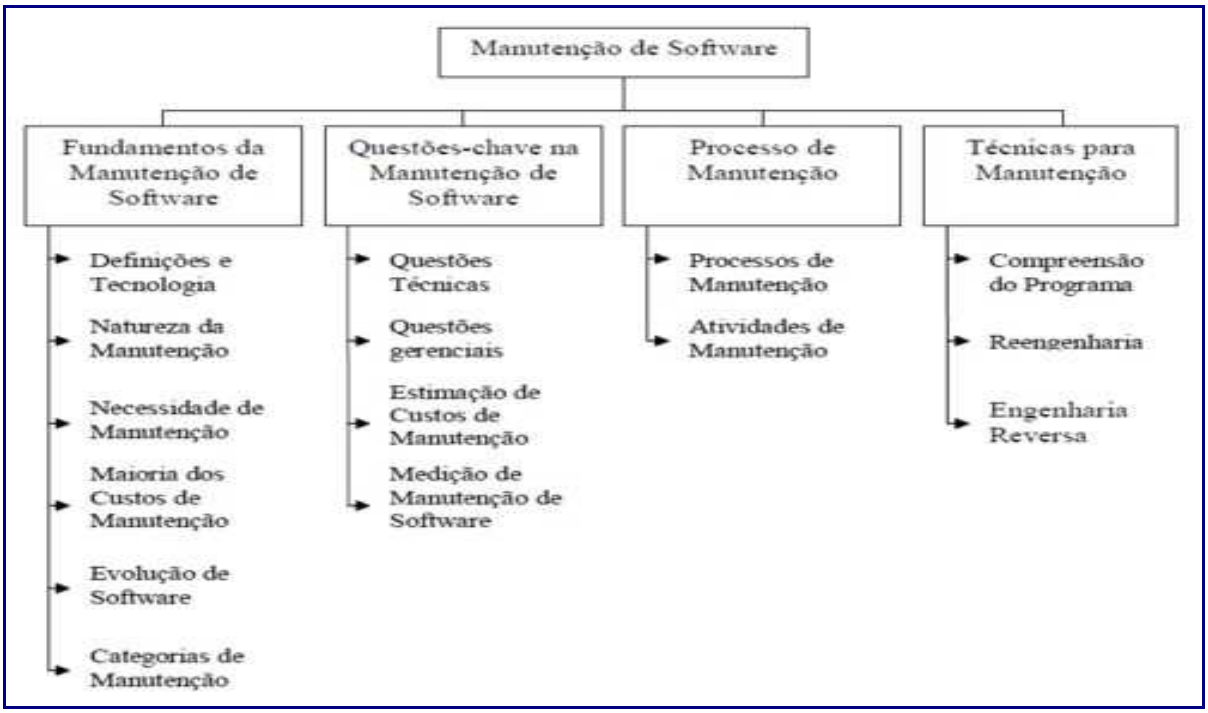


Figura 19: Tópicos da Manutenção de software

2.1.3.5.1 Fundamentos da Manutenção de Software

Essa primeira seção introduz os conceitos e terminologia subjacente, que formam uma base para a compreensão do papel e o alcance da manutenção de software. Os tópicos fornecem definições e enfatizam o porquê da necessidade de manutenção. As categorias de manutenção de software são críticas para a compreensão do seu significado subjacente.

a) Definição e Terminologia

Manutenção de Software é definida na Norma IEEE para Manutenção de Software, IEEE 1219, como a modificação de um produto de software após a entrega para corrigir falhas, para melhorar o desempenho ou outros atributos, ou adaptar o produto para um ambiente modificado. A norma também arrola as atividades de manutenção antes da entrega do de produtos de software, mas apenas em um apêndice da informação da norma.

O IEEE/EIA 12207 é uma norma para o processo do ciclo de vida de software que retrata essencialmente a manutenção como um dos processos primários do ciclo de vida, e descreve a manutenção como o processo que sofre um produto de software “de modificação de código e documentação associada devido a um problema ou à necessidade de melhoria. O objetivo é modificar o software atual

enquanto o produto preserve a sua integridade." ISO/IEC 14764, o padrão internacional para a manutenção de software, define manutenção nos mesmos termos do IEEE/EIA 12207, e enfatiza os aspectos pré entrega de manutenção, como, por exemplo, o planejamento.[IEEE1219-98:s3.1.12; IEEE12207.0-96:s3.1,s5.5; ISO14764-99:s6.1]

b) Natureza da Manutenção

A manutenção de Software sustenta o produto de software ao longo do seu ciclo de vida operacional. Os requisitos de modificação são registrados e monitorados, o impacto das propostas de mudanças é determinado, código de software e outros artefatos são modificados, o teste é realizado, e uma nova versão é lançada. Além disso, a formação e o diário de apoio são oferecidos aos usuários. Pfleeger [Pfl01] afirma que "a manutenção tem um objetivo maior, com maior monitoramento e controle" do que o desenvolvimento. Um mantenedor é definido pelo IEEE/EIA 12207 como uma organização que realiza atividades de manutenção [IEEE12207.0-96]. Este KA, muitas vezes, refere-se aos indivíduos que desempenham essas atividades, contrapondo-os com os desenvolvedores. IEEE/EIA 12207 identifica as principais atividades de manutenção de software como: processo de execução; problema com análise e modificação; modificação, implementação, manutenção revisão/aceitação; migração; e reformas. Estas atividades serão discutidas no tópico 3.2 – Manutenção de Atividades. Os mantenedores podem aprender a partir do conhecimento do desenvolvedor de software. O contato precoce com os desenvolvedores ajuda a reduzir os esforços de manutenção do mantenedor. Em alguns casos, o engenheiro de software pode não ser encontrado ou se mudou para outras tarefas, o que cria um desafio adicional para os mantenedores. Os mantenedores devem ter os produtos do desenvolvimento, como os códigos ou documentação, por exemplo, e mantê-los ou evoluí-los progressivamente ao longo do ciclo de vida do software.[Pfl01:c11s11.2]

c) Necessidade da manutenção

A manutenção é necessária para assegurar que o software continue a satisfazer os requisitos dos usuários. A manutenção é aplicável a qualquer software desenvolvido utilizando o ciclo de vida modelo (por exemplo, em espiral). O sistema

é modificado devido às ações de mudanças corretivas e não – corretivas. A sua manutenção deve ser realizada, a fim de: [Pig97: c2s2.3; Tak97:c1]

- Corrigir falhas;
- Melhorar o design;
- Implementar melhorias;
- Interface com outros sistemas;
- Adaptar programas para que diferentes hardware, software, sistema de recursos e de telecomunicações possam ser utilizados;
- Migrar software legado
- Aposentar software

As atividades de manutenção compreenderão quatro características principais, de acordo com Pfleeger [Pfl01]:

- Manter o controle sobre o software no dia-a-dia das funções;
- Manter o controle sobre as modificações de software;
- Aperfeiçoar funções existentes;
- Impedir o desempenho degradante do software para um nível inaceitável;

d) A Maioria dos Custos de Manutenção

A manutenção consome uma parte significativa dos recursos financeiros do ciclo de vida de software. A percepção comum de manutenção de software é que ela apenas corrige falhas. No entanto, estudos e pesquisas ao longo dos anos têm indicado que a maioria, mais de 80%, do esforço da manutenção de software é usada para ações não - corretivas. [Abr93, Pig97, Pre01] Jones (Jon91) descreve a maneira pela qual os grupos de gerentes de manutenção de software frequentemente apresentam melhorias e correções na gestão de seus relatórios. A inclusão de problemas acessórios aos relatórios contribui para algumas das ideias erradas sobre o elevado custo de correções. Compreender as categorias de manutenção de software ajuda a compreender a estrutura dos custos de manutenção do software. Também, entender os fatores que influenciam a manutenção de um sistema pode ajudar na contenção de despesas. Pfleeger [Pfl01] apresenta alguns dos fatores técnicos e não técnicos que afetam os custos de manutenção de software, como segue: [Pfl01:c11s11.3;Pig97:c3;Pre01:c30s2.1, c30s2.2]

- Tipo de aplicação
- Novidade em Software
- Disponibilidade de pessoal para a manutenção de software
- Duração de vida do Software
- Características do Hardware
- Qualidade de design de software, construção, documentação e testes.

e) Evolução do Software

Lehman foi o primeiro estudioso (addressed) da manutenção de software e evolução dos sistemas, em 1969. Ao longo de um período de vinte anos, suas pesquisas levaram à formulação de oito "Leis de Evolução". [Leh97]. As principais conclusões incluem o fato de que a manutenção é evolutiva, e que as decisões sobre a manutenção são auxiliadas pela compreensão do que acontece com os sistemas (e software) ao longo do tempo. Outra conclusão é que o estado de manutenção é um desenvolvimento continuado, exceto quando há uma entrada extra (ou constrangimento)- ou seja, quando o software nunca está concluído e continua a evoluir. Como ele evolui, ele cresce mais complexo, a menos que se tomem providências para reduzir esta complexidade. Desde que os softwares regulares demonstram comportamento e tendências, estes podem ser medidos. As tentativas de desenvolver modelos para estimar os esforços com a manutenção têm sido feitos, e, como resultado, instrumentos úteis de gestão têm sido desenvolvidos. [Art88], (Bel72)[Art88:c1s1.0,s1.1,s1.2,c11s1.1,s1.2; Leh97:108-124],(Bel72).

f) Categorias de Manutenção

Lientz & Swanson inicialmente definidam três categorias de manutenção: corretiva, adaptável e perfectiva. [Lie78;IEEE1219-98] Esta definição foi posteriormente atualizada pela Norma para a Manutenção de Software, Engenharia de Software, ISO/IEC 14764, para incluir quatro categorias, como segue:[Lie78; Dor02:v1c9s1.5; IEEE1219- 98:s3.1.1,s3.1.2,s3.1.7,A.1.7; ISO14764-99:s4.1,s4.3,s4.10, s4.11,s6.2; Pig97:c2s2.3]

- Manutenção corretiva: modificação reativa realizada após a entrega de produtos de software para corrigir problemas detectados.

- Manutenção Adaptativa: Alteração de um produto de software realizada após a entrega e visa manter o software utilizável em um produto modificado ou que fora alterando seu ambiente.
- Manutenção Perfectiva: Alteração de um produto de software após a entrega, para melhorar o desempenho ou capacidade de manutenção.
- Manutenção Preventiva: Alteração de um produto de software após a entrega para detectar e corrigir latente falhas no software.

A ISO/IEC 14764 classifica a manutenção adaptativa e perfectiva como acessórias. Também reúne as categorias de manutenção preventiva e corretiva na categoria de correção, conforme mostrado na Tabela 1. A manutenção preventiva, a mais nova categoria, a maior parte das vezes é realizada sobre produtos de software onde a segurança é crítica.

2.1.3.5.2 Problemas chave na manutenção de software

Um número de problemas chave deve ser tratado para garantir a manutenção efetiva do software. É importante entender que manutenção de software fornece desafios técnicos e gerenciais únicos para engenheiros de software. Tentar achar uma falha em software contendo 500 mil linhas de código que o engenheiro de software não desenvolveu é um bom exemplo. Semelhantemente, competir com desenvolvedores de software por recursos é uma batalha constante. Planejar uma versão futura enquanto se desenvolve a próxima versão e se envia atualizações de segurança para a versão atual também fornece um desafio. A seção seguinte apresenta alguns problemas técnicos e gerenciais relacionados à manutenção de software. Eles têm sido agrupados como: Problemas técnicos, Problemas gerenciais, estimativa de custo e Medidas

a)Problemas técnicos

Compreensão limitada refere-se a como o engenheiro de um software, pode, rapidamente, perceber onde se deve fazer uma mudança ou uma correção no software que este indivíduo não desenvolveu. Pesquisas indicam que cerca de 40% a 60% dos esforços de manutenção são dedicados ao entendimento do software a ser modificado. Assim, o tema da compreensão do software é de grande interesse para os engenheiros de software.

A compreensão é mais difícil na representação do texto orientador, no código-fonte, por exemplo, onde muitas vezes é difícil rastrear a evolução do software através das suas releases / versões quando as alterações não são documentadas e quando os desenvolvedores não estão disponíveis para explicá-las, o que muitas vezes é o caso. Assim, engenheiros de software podem inicialmente ter um compreensão limitada do software, e muito tem de ser feito para remediar isto.

Testando. O custo da repetição de ensaios em uma grande peça de software pode ser significativo em termos de tempo e dinheiro. Regressão de testes, é a seleção de uma reanálise de software ou componente para verificar se as modificações não tenham causado efeitos não intencionais, que é importante para a manutenção. Encontrar tempo para testar é muitas vezes difícil. Há também a desafio de coordenar diferentes testes quando os membros da equipe da manutenção estão trabalhando em diversos problemas ao mesmo tempo. [Plf01] Quando se executa um software com as funções críticas, pode ser impossível trazê-lo para testar offline. O Teste de Software KA fornece informações adicionais e as referências sobre o assunto, em seu sub-tópico 2.2.6 Regressão de testes.

Estudos de impacto Dor02:v1c9s1.10; Pfl01: c11s11.5. Estudo de impacto descreve como a conduta, de forma eficaz, pode analisar completamente o impacto de uma mudança no software atual. Os mantenedores devem possuir um conhecimento aprofundado da estrutura do software e de seu conteúdo [Pfl01]. Eles usam esse conhecimento para realizar análises de impacto, o que identifica todos os sistemas e produtos de software afetado por uma solicitada mudança de software, e desenvolve uma estimativa dos recursos necessários para realizar a mudança. [Art88] Além disso, o risco de fazer a mudança é determinado. A mudança pedida, às vezes é chamada de Pedido de Modificação (MR) e, muitas vezes, requerem um relatório problema (PR), que deve, em primeiro lugar, ser analisado e traduzido em termos de software. [Dor02] É realizado após a entrada de um pedido de mudança de software no gerenciamento de configuração processo. Arthur [Art88] afirma que os objetivos de análise de impacto são:.....

- Determinação do âmbito de uma mudança no sentido de planejar e executar trabalhos;
- Desenvolvimento de estimativas precisas dos recursos necessários para executar o trabalho;

- Análise dos custos/benefícios das alterações solicitadas;
- Comunicação a terceiro da complexidade de uma dada mudança

A gravidade de um problema é frequentemente utilizada para decidir como e quando um problema será corrigido. O engenheiro de software, então, identifica os componentes afetados. Várias soluções potenciais são apresentadas, e, em seguida, é feita uma recomendação quanto ao melhor curso de ação. O Software projetado com a manutenção em mente facilita e muito a avaliação de impacto. Mais informações podem ser encontradas na Gerência de Configuração de Software KA.

Capacidade de Manutenção [ISO14764-99:s6.8s6.8.1; Pfl01: c9s9.4; Pig97:c16]. Como é que um acompanhamento pode promover as questões da capacidade de manutenção durante o desenvolvimento? O IEEE [IEEE610.12-90] define capacidade de manutenção como a facilidade com que pode o software ser mantido, reforçado, adaptado, corrigido ou satisfazer requisitos especificados. ISO/IEC define capacidade de manutenção como uma característica de qualidade (ISO9126-01). As subcaracterísticas da capacidade de manutenção devem ser especificadas, revista, e controladas durante o desenvolvimento das atividades de software, a fim de reduzir os custos de manutenção. Se for feito com êxito, a manutenção do software irá melhorar. Isso é muitas vezes difícil de ser concretizado, porque as características da capacidade de manutenção não são um importante foco durante o processo de desenvolvimento de software. Os desenvolvedores estão preocupados com muitas outras coisas e frequentemente ignoram as exigências do mantenedor. Este fato, por sua vez, pode, e muitas vezes o faz, resultar em uma falta de documentação do sistema, o que é uma das principais causas das dificuldades no programa de compreensão e avaliação do impacto. Também tem foi observado que a presença de sistemas, de processos maduros, técnicas e ferramentas ajudam a reforçar a manutenção de um sistema.

Estudos de impacto Dor02:v1c9s1.10; Pfl01: c11s11.5. Estudo de impacto descreve como a conduta, de forma eficaz, pode analisar completamente o impacto de uma mudança no software atual. Os mantenedores devem possuir um conhecimento aprofundado da estrutura do software e de seu conteúdo [Pfl01]. Eles usam esse conhecimento para realizar análises de impacto, o que identifica todos os sistemas e produtos de software afetado por uma solicitada mudança de software, e desenvolve uma estimativa dos recursos necessários para realizar a mudança.

[Art88] Além disso, o risco de fazer a mudança é determinado. A mudança pedida, às vezes é chamada de Pedido de Modificação (MR) e, muitas vezes, requerem um relatório problema (PR), que deve, em primeiro lugar, ser analisado e traduzido em termos de software. [Dor02] É realizado após a entrada de um pedido de mudança de software no gerenciamento de configuração processo. Arthur [Art88] afirma que os objetivos de análise de impacto são:

- Determinação do âmbito de uma mudança no sentido de planejar e executar trabalhos
- Desenvolvimento de estimativas precisas dos recursos necessários para executar o trabalho
- Análise dos custos/benefícios das alterações solicitadas
- Comunicação a terceiro da complexidade de uma dada mudança

A gravidade de um problema é frequentemente utilizada para decidir como e quando um problema será corrigido. O engenheiro de software, então,identifica os componentes afetados.

Várias soluções potenciais são apresentadas, e, em seguida, é feita uma recomendação quanto ao melhor curso de ação. O Software projetado com a manutenção em mente facilita e muito a avaliação de impacto. Mais informações podem ser encontradas na Gerência de Configuração de Software KA.

Manutenibilidade [ISO14764-99:s6.8s6.8.1; Pfl01: c9s9.4; Pig97:c16]. Como é que um acompanhamento pode promover as questões de manutenibilidade durante o desenvolvimento? O IEEE [IEEE610.12-90] define manutenibilidade como a facilidade com que pode o software ser mantido, reforçado, adaptado, corrigido ou satisfazer requisitos especificados. ISO/IEC define manutenibilidade como uma característica de qualidade (ISO9126-01). As subcaracterísticas da manutenibilidade devem ser especificadas, revista, e controladas durante o desenvolvimento das atividades de software, a fim de reduzir os custos de manutenção. Se for feito com êxito, a manutenção do software irá melhorar. Isso é muitas vezes difícil de ser concretizado, porque as características da manutenibilidade não são um importante foco durante o processo de desenvolvimento de software.

Os desenvolvedores estão preocupados com muitas outras coisas e frequentemente ignoram as exigências do mantenedor. Este fato, por sua vez, pode,

e muitas vezes o faz, resultar em uma falta de documentação do sistema, o que é uma das principais causas das dificuldades no programa de compreensão e avaliação do impacto. Também tem foi observado que a presença de sistemas, de processos maduros, técnicas e ferramentas ajudam a reforçar a manutenção de um sistema.

b) Gestão de Problemas

Alinhamento com os objetivos organizacionais [Ben00: c6sa; Dor02: v1c9s1.6]. Os objetivos organizacionais descrevem a forma de demonstrar o retorno sobre os investimentos das atividades de manutenção de software. Bennett [Ben00] afirma que " o objetivo é normalmente desenvolver software baseados em projetos, com uma escala definida no tempo e orçamento.

O principal destaque é a entrega a tempo e dentro do orçamento para satisfazer as necessidades do utilizador. Em contrapartida, a manutenção de software frequentemente tem o objetivo de estender a vida de um software para o maior tempo quanto possível. Além disso, pode ser motivado pela procuro de utilizador por atualizações de software e acessórios. Em ambos os casos, o retorno sobre o investimento não é claro, de modo que a visão a nível da direção é muitas vezes uma das principais atividades que consomem recursos significativos, sem benefícios quantificáveis claros para a organização."

Pessoalidade [Dek92 :10-17; Dor02: v1c9s1.6; Par86: c4s8-c4s11] (Lie81). Pessoalidade refere-se a forma de atrair e manter software o pessoal da manutenção. Manutenção não é frequentemente visto como um trabalho glamoroso. Deklava fornece uma lista de problemas com os recursos humanos relacionados com a base em levantamento de dados. [Dek92] Como resultado, o pessoal da manutenção de software são frequentemente vistos como "Cidadãos de segunda classe" (Lie81), com a moral prejudicada.

Processo [Pau93; Ben00: c6sb; Dor02: v1c9s1.3]. O processo de Software é um conjunto de atividades, métodos, práticas e transformações que as pessoas usam para desenvolver e manter o software e produtos associados. [Pau93] Pelo nível do processo, algumas atividades de manutenção são comuns as do desenvolvimento de software (por exemplo, gerência de configuração de software é uma atividade crucial para ambos). [Ben00] A Manutenção exige também várias

atividades que não são encontrados no desenvolvimento de software. Essas atividades apresentam desafios para a gestão. [Dor02]

Aspectos organizacionais da manutenção [Pfl01: c12s12.1-c12s12.3; Par86: c4s7; Pig97: c2s2.5; Tak97: C8]. Os aspectos organizacionais descrevem a forma de identificar qual organização e/ou função será responsável pela manutenção de software. A equipe que desenvolve o software não é necessariamente atribuída a manter o software quando este estiver operacional. Para decidir onde irá funcionar a manutenção de software, as organizações de engenharia de software podem, por exemplo, permanecer com o desenvolvedor original ou ir a uma equipe separada (ou mantenedores).

Muitas vezes, a opção mantenedor é escolhida de modo a garantir que o software funcione adequadamente e evolua para satisfazer necessidades evolutivas dos utilizadores. Uma vez que há muitos prós e os contras de cada uma destas opções [Par86,Pig97], a decisão deve ser feita com base em cada caso. O importante é a delegação ou cessão da responsabilidade pela manutenção a uma única pessoa ou grupo [Pig97], independentemente da estrutura da organização.

Terceirização [Dor02: v1c9s1.7; Pig97: c9s9.1, s9.2], (Car94; McC02) As indústrias terceirizadas estão a tornar-se uma grande indústria. As grandes corporações estão terceirizando portfólios inteiros de sistemas de software, incluindo a manutenção de software. Mas muitas vezes, a terceirização é uma opção selecionada de forma não absoluta, por empresas que não estão dispostas a perder o controle do software usado no seu core business. Carey (Car94) relata que alguns irão terceirizar apenas se poderem encontrar maneiras da manutenção de controle estratégico. No entanto, as medidas de controle são difíceis de encontrar. Um dos grandes desafios para os terceirizados é determinar o alcance dos serviços de manutenção necessários e os detalhes contratuais. McCracken (McC02) afirma que 50% dos terceirizados prestam serviços sem qualquer acordo claro de nível de serviço. As empresas de terceirização geralmente gastam muitos meses para avaliar o software que irá entrar em um relação contratual, antes de concretizá-la. [Dor02] Outro desafio identificado é a transição do software para o contratante. [Pig97]

c) Estimação do Custo de manutenção

Engenheiros de software deve compreender as diferentes categorias de

manutenção de software, acima discutidas, a fim de abordar a questão da avaliação do custo da manutenção de software. Para fins de planejamento, calcular custos é um aspecto importante de manutenção de software.

Estimação de Custo [Art88: c3; Boe81: C30; Jon98: C27; Pfl01: c11s11.3; Pig97: C8]. Foi mencionado, análise de impacto, que a avaliação de impacto identifica todos os sistemas e produtos de software afetados por uma mudança de software e desenvolve um pedido de estimativa dos recursos necessários para realizar essa mudança. [Art88] As estimativas dos custos de manutenção são afetadas por muitos fatores técnicos e não técnicos. ISO/IEC14764 afirma que "as duas abordagens mais populares para estimar os recursos para manutenção de software são o uso de modelos paramétricos e a utilização da experiência "[ISO14764-99: s7.4.1]. Na maior parte das vezes, é usada a combinação de ambas.

Modelos paramétricos [Ben00: S7; Boe81: C30; Jon98: C27; Pfl01: c11s11.3] Alguns trabalhos foram realizados na aplicação paramétrica para a modelagem do custo de manutenção de software. [Boe81, Ben00] É necessário frisar que os dados dos últimos projetos são necessários em a fim de se utilizar os modelos. Jones [Jon98] discute todos os aspectos de estimar os custos, incluindo os pontos das funções (IEEE14143.1-00), e oferece um detalhado capítulo sobre a estimação da manutenção.

Experiência [ISO14764-00: S7, s7.2, s7.2.1, s7.2.4; Pig97: C8; Sta94]. A experiência, na forma de perito acórdão (usando a técnica Delphi, por exemplo) e analogias, é um trabalho de desagregação que aborda a estrutura que deveria ser utilizada para aumentar os dados de modelos paramétrico. Claramente, a melhor abordagem para a estimativa da manutenção é combinar dados empíricos e experiência. Estes dados deverão ser fornecidos como resultado de um programa de medição.

d) Medição da Manutenção de Software

Grady e Caswell [Gra87] discutem a instituição de um programa para as grandes empresas para a medição da manutenção de software, onde formulários e dados coletados são descritos. O Sistema de Medição (PSM) é um software prático que descreve um projeto issuedriven processo de medição que é utilizado por muitas organizações e é bastante prático. [McG01] Os softwares possuem medidas que são

comuns a todos os empreendimentos, sendo que o Engineering Institute (SEI) tem identificadas as seguintes categorias de Software: tamanho; esforço; cronograma; e qualidade. [Pig97]m Estas medidas constituem um bom ponto de partida para o mantenedor. A discussão sobre o processo e o produto da medição é apresentada no Software Engineering Process KA. O programa é descrito na medição do Software Engenharia de Gestão KA.

Medidas Específicas IEEE1219-98:Table3; Sta94:p239-249. Abran [Abr93] apresenta técnicas de benchmarking para comparar as diferentes organizações da manutenção interna. O mantenedor deve determinar quais são as medidas adequadas para a organização em questão. [IEEE1219 - 98; ISO9126-01; Sta94] sugere medidas que sejam mais específicas para os programas de medição da manutenção de software. Esta lista inclui uma série de medidas para cada uma das quatro subcaracterísticas de durabilidade:

- Analisativas: Medidas do esforço do mantenedor ou recursos gastos na tentativa de diagnosticar deficiências ou causas de insucesso, ou na identificação de peças a serem modificadas.
- Inconstância: Medidas do mantenedor do esforço associados à implementação de uma determinada modificação.
- Estabilidade: Medidas do comportamento inesperado do software, incluindo a que surgir durante o ensaio.
- Testabilidade: Medidas do esforço do mantenedor e usuário para tentar testar o software modificado.

Algumas medidas de manutenção do software podem ser obtidas utilizando ferramentas comerciais disponíveis. (Lag96; Apr00)[IEEE1061-98:A.2; Pig97:c14s14.6; Gra87 ; Tak97: c6s6.1-c6s6.3]

2.1.3.5.3 Processo de manutenção

O Processo de Manutenção fornece referências de subzona de padrões utilizados para implementar o processo de manutenção de software. “Atividades de Manutenção” diferencia o tópico “Desenvolvimento da manutenção” e mostra a relação da engenharia de software para outras atividades. A necessidade da engenharia de software é um processo bem documentado. Os modelos CMMI se aplicam aos processos de manutenção de

software, e são semelhantes aos processos de desenvolvimento. [SEI01] Manutenção de Software Capability Maturity descreve os únicos processos de manutenção de software (Apr03, Nie02, Kaj01).

a) Processos de Manutenção

O processo de manutenção fornece atividades necessárias, detalhando as entradas/saídas para essas atividades, e são descritas na Manutenção em software IEEE 1219 e nas normas ISO/IEC 14764. O modelo de processo de manutenção descrito no Standard Manutenção de Software (IEEE1219) inicia-se com o esforço de manutenção de software durante a fase pós-entrega e aborda itens como planejamento para manutenção. Aquele processo é ilustrado na Figura 20.

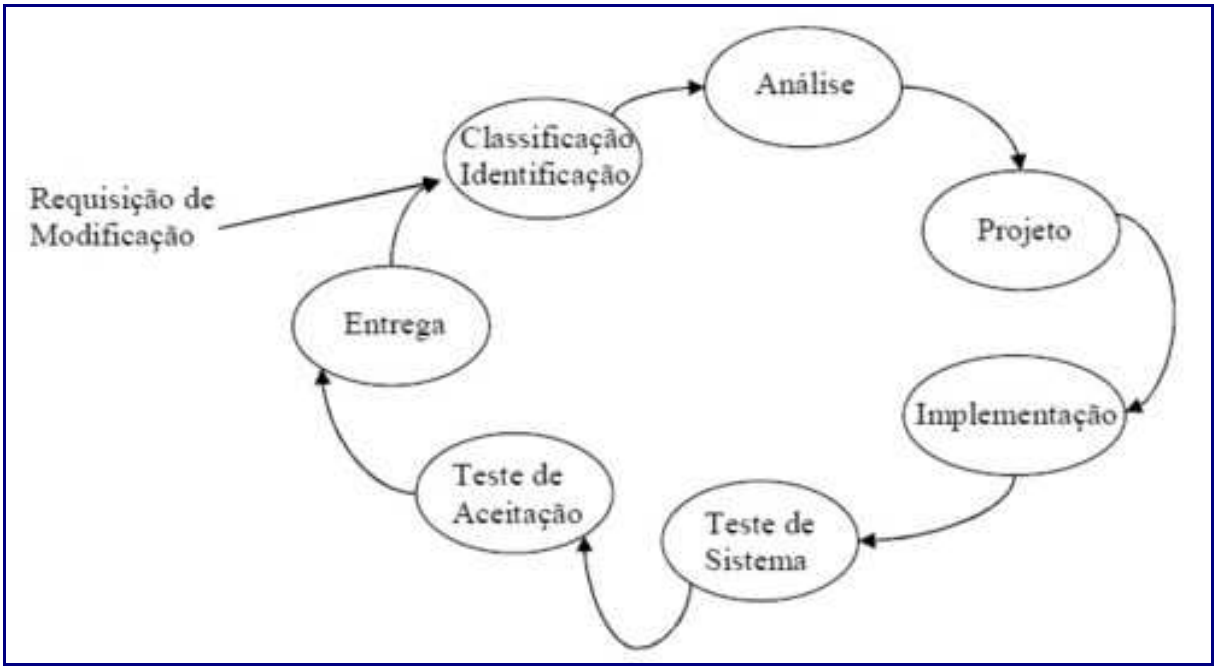


Figura 20: Atividades do Processo de Manutenção

A ISO/IEC 14764 [ISO14764-99] é um aprofundamento do processo de manutenção IEEE/EIA 12207.0-96. As atividades de manutenção da ISO/IEC é um processo semelhante à dos do IEEE, com a diferença de serem agregadas um pouco diferente. As atividades desenvolvidas no processo de manutenção pela ISO / IEC são mostradas na Figura 21.

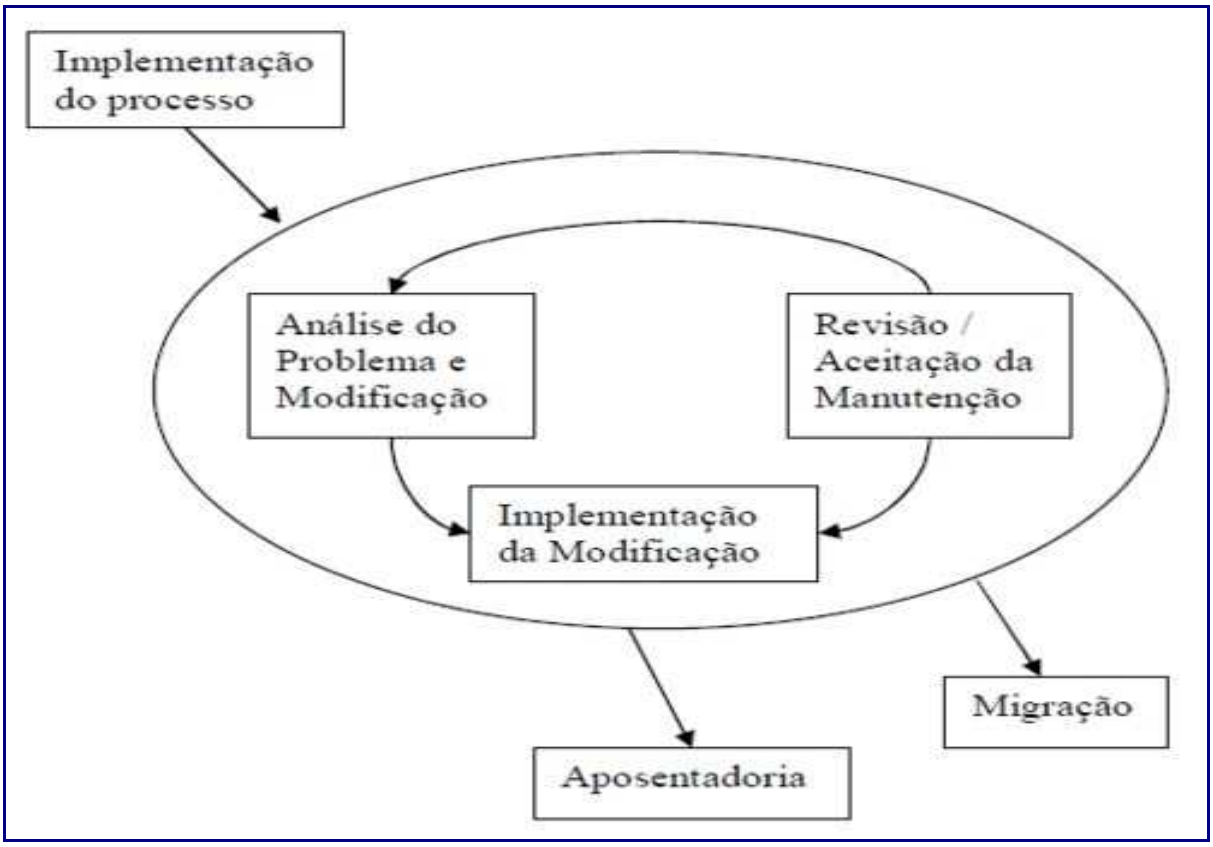


Figura 21: Processo de Manutenção de Software

Cada uma das atividades primárias de manutenção de software das ISO/IEC 14764 é subdividida em tarefas, como se segue:

- Processo de Execução
- Problema e Análise da Modificação
- Execução da Modificação
- Revisão da Manutenção / Aceitação
- Migrações
- Aposentadoria do Software

Grubb & Takang [Tak97] fornecem um histórico de modelos de processos de manutenção conducentes aos modelos de processos de desenvolvimento do IEEE e ISO/IEC. Parikh [Par86] também dá uma boa visão geral de um processo genérico de manutenção. Recentemente, metodologias ágeis foram criadas e promoveram a eficiência dos processos. Essa exigência decorre do da procura por fast turn-around de manutenção de serviços. Alguns experimentos com manutenção são apresentados em Extreme (Poo01).

b) Atividades de Manutenção

Como já foi dito, muitas atividades de manutenção são semelhantes com as de desenvolvimento de software. Os mantenedores executam análise, projeto, codificação, testes e documentação. Estes requisitos devem monitorar suas atividades como é feito no desenvolvimento, atualização a documentação como base na alteração. ISO/IEC14764 recomenda que, quando um mantenedor refere-se a um processo de desenvolvimento semelhante, ele tem que adaptá-lo para satisfazer suas necessidades específicas [ISO14764-99: s8.3.2.1, 2]. No entanto, para a manutenção de software, algumas atividades envolvem processos de software exclusivos para a manutenção.

Atividades Únicas Dor02:v1c9s1.9.1; IEEE1219-98:s4.1,s4.2; ISO14764 99:s8.2.2.1,s8.3.2.1;Pfl01:c11s11.2. Há uma série de processos, atividades e práticas que são exclusivas da manutenção de software, como por exemplo:

- Transição: uma sequência de atividades coordenadas e controladas onde o software é transferido progressivamente a partir do desenvolvedor para o mantenedor [Dek92, Pig97]
- Aceitação / Recusa do Pedido de Modificação: o pedido de modificação, por ter certo tamanho/esforço/complexidade, pode ser rejeitado pelo mantenedor e reencaminhado para um desenvolvedor [Dor02], (Apr01)
- Requisição de modificação e sua solicitação ao Help Desk: uma função de apoio aos usuários finais, o que desencadeia avaliação, priorização, e custeio do pedido de modificação [Ben00]
- Análise do impacto (ver ponto 2.1.3 para mais detalhes)
- Suporte de Software: ajuda e aconselhamento ao usuário relativo a um pedido de informação (por exemplo, regras empresariais, a validação de dados, significado e pedidos ad-hoc/reports) (Apr03)
- Acordos de nível de serviço (SLAs) especificam que o domínio dos contratos de manutenção é de responsabilidade dos mantenedores (Apr01),

As atividades de apoio [IEEE1219-98:A.7,A.11; IEEE12207.0-96:c6,c7;ITI01; Pig97:c10s10.2,c18] ;(Kaj01). Os mantenedores podem também realizar atividades de apoio, tais como planejamento da manutenção de software, gestão da configuração de software, verificação e validação, garantia da qualidade de software, revisões e auditorias, treinamento ao usuário. Outra atividade de apoio do

mantenedor, a formação, é também necessária. [Pig97; IEEE12207.0-96] (Kaj01)

Atividades de Planejamento da Manutenção [IEEE1219-98: A.3; ISO14764-99: S7; ITI01; Pig97: C7, C8]. Uma atividade importante para a manutenção de software é o planejamento, e os mantenedores devem resolver as questões relacionadas com o planejamento com uma serie de perspectivas:

- Planejamento do negócio (nível organizacional)
- Planejamento da Manutenção (nível de transição)
- Planejamento da Release/versão (nível de software)
- Pedido individual de um processo de mudança de (nível solicitação).

No nível individual, o planejamento é realizado durante a análise de impacto. A atividade de planejamento da release/versão exige que o mantenedor [ITI01]:

- Colher as datas de disponibilidade dos pedidos individuais;
- Concorde com os usuários sobre o conteúdo das subsequentes releases/versões.
- Identifique conflitos potenciais e crie alternativas para saná-los.
- Avalie os riscos de uma determinada entrega e desenvolva um plano de back-out no caso de problemas surgirem.
- Informar todas as partes interessadas

Considerando que o desenvolvimento de projetos de software geralmente pode passar de alguns meses para um número reduzido de anos, a fase de manutenção geralmente dura muitos anos. Fazer a previsão de recursos é um elemento fundamental do planejamento de manutenção. Esses recursos devem ser incluídos nos orçamentos do planejamento dos projetos dos desenvolvedores. O planejamento da manutenção de Software deve começar com a decisão de desenvolver um novo sistema, e deverá considerar objetivos de qualidade (IEEE1061-98). Um conceito de documento deverá ser desenvolvido, seguido de um plano de manutenção. O conceito de documento de manutenção [ISO14764 - 99: s7.2] deve abordar:

- O âmbito da manutenção de software
- Adaptação do processo de manutenção de software
- Identificação da organização de manutenção do software
- Uma estimativa dos custos de manutenção de software

O próximo passo é desenvolver um software correspondente ao plano de manutenção. Este plano deverá ser elaborado durante o desenvolvimento de software, e deve especificar como os usuários irão solicitar as modificações ou relatar os problemas de software. O planejamento da manutenção de Software de [Pig97] é abordado no IEEE 1219 [IEEE1219-98] e ISO/IEC 14764. [ISO14764-99] ISO/IEC14764 fornece diretrizes para um plano de manutenção. Por fim, ao mais alto nível, a organização da manutenção terá de realizar atividades de planejamento empresarial (orçamental, financeira e recursos humanos), tal como todas as outras divisões da organização. A gestão de conhecimentos necessários para o fazer pode ser encontrada nas Disciplinas de Engenharia de Software Relacionadas ao final do capítulo.

Gerenciamento de configuração de Software [Art88:c2,c10; IEEE1219-98:A.11; IEEE12207.0-96:s6.2; Pfl01:c11s11.5; Tak97:c7]. O padrão IEEE para Manutenção de Software, IEEE 1219 [IEEE1219-98], descreve a gestão da configuração de software como um elemento crítico do processo de manutenção. Os procedimentos de gerenciamento de configuração do Software deverão prever a verificação, validação, e auditoria de cada passo necessário para identificar, autorizar, executar, e liberar os produtos de software. Ele não é suficiente simplesmente para monitorar a Modificação ou os Pedidos de relatórios dos problemas. O produto de software e quaisquer mudanças feitas a ele têm de ser controladas. Este controle é estabelecido pela implementação e aplicação um programa aprovado de gerenciamento de processo de configuração (SCM). O Software Configuration Management KA fornece detalhes de SCM e discute o processo pelo qual os pedidos de mudança de software sejam apresentados, avaliados e aprovados. O processo de SCM para manutenção de software é diferente do processo SCM para desenvolvimento de software, quanto ao número de pequenas mudanças que devem ser controladas no software operacional. O processo SCM é implementado através do desenvolvimento de um plano de gestão e procedimentos operacionais. Os mantenedores devem participar da configuração das câmaras de controle e determinar o conteúdo dos próximos release/versões.

Qualidade de software IEEE12207.0-96:s6.3; IEEE1219-98:A.7; ISO14764-99:s5.5.3.2. Não é suficiente quer o simples aumento da esperança de aumentar a

qualidade irá resultar na qualidade resultarão da manutenção de software. Ela deve ser planejada e implementada para apoiar os processos processo de manutenção. As atividades e técnicas de Segurança de Qualidade de (SQA), V & V, opiniões, e auditorias devem ser selecionadas em consonância com todos os outros processos para alcançar o nível desejado de qualidade. É também recomendado que os mantenedores adaptem os processos de desenvolvimento, técnicas e resultados, para instituir testes de documentação, e testes de resultados. [ISO14764-99] Mais detalhes podem ser encontrados no Quality Software KA.

2.1.3.5.4 Técnicas de Manutenção

Este subitem introduz alguns dos métodos geralmente aceitos pelas técnicas utilizadas na manutenção de software.

a) Programa de Compreensão

Os programadores gastam um tempo considerável na leitura e compreensão dos programas, a fim de implementar mudanças. Os Códigos Navegadores são instrumentos fundamentais para a compreensão do programa. A documentação clara e concisa pode auxiliar no processo de compreensão do programa. [Arn92: C14; Dor02: v1c9s1.11.4; Tak97: c3]

b) Reengenharia

Reengenharia é definida como a análise e alteração de software para reconstituí-lo em um novo formulário, e inclui a posterior aplicação do novo formulário. Dorfman e Thayer [Dor02] indicam que a reengenharia é o mais radical (e caro) formulário de alteração. Outros entendem que a reengenharia pode ser usada para pequenas alterações. Ela, muitas vezes, não compromete-se a melhorar a durabilidade, mas a substituir o envelhecido legado de software. Arnold [Arn92] prevê um compêndio abrangente de temas, como por exemplo: conceitos, ferramentas e técnicas, estudos de caso, e os riscos e benefícios associados à reengenharia.[Arn92: C1, C3-C6; Dor02: v1c9s1.11.4; IEEE1219-98: B.2], (Fow99)

c) Engenharia reversa

Engenharia reversa é o processo de análise de software que identifica os

componentes do software e suas inter-relações para criar representações do software sob outra forma ou em níveis mais elevados de abstração. A Engenharia Reversa é passiva, ela não muda o software, ou resultar em novos softwares. Ela produzirá esforços chamados grafos e gráficos de controle de fluxo para achar o código. Um tipo de engenharia reversa é a redocumentação. Outro tipo é a recuperação de design [Dor02]. Refactoring é um programa de transformação, que reorganiza um programa sem modificar o seu comportamento, e é uma forma de engenharia reversa que visa melhorar a estrutura do programa. (Fow99) Finalmente, os dados de engenharia reversa têm ganhado importância ao longo dos últimos anos, onde schemas lógicos foram recuperados físicos a partir de bases de dados. (Hen01)[Arn92: c12; Dor02: v1c9s1.11.3; IEEE1219-98: B.3; Tak97: c4, Hen01]

2.1.3.6 Gerência de Configuração de Software

Um *sistema* pode ser definido com uma conjunto de componentes organizados para realizar uma função especifica ou um conjunto de funções (IEEE 610.12-90). A configuração de um sistema consiste das características funcionais e/ou físicas de hardware, firmware, ou software ou a combinação destas, conforme estabelecida na documentação técnica e realização de um produto. Ela também pode ser pensada como uma coleção de versões específicas de hardware, firmware, software ou itens combinados de acordo com procedimentos de construção específicos para servir um propósito particular.

Gerenciamento de Configuração (CM), então, é a disciplina de identificação e configuração de um sistema em pontos distintos no tempo com a finalidade de controlar sistematicamente as alterações na configuração, e manutenção da integridade e da rastreabilidade da configuração durante todo o ciclo de vida do sistema (Ber97) È formalmente definido (IEEE610.12-90) como “Uma disciplina aplicando técnicas e administrando direção e fiscalização para: identificar e documentar as características funcionais e físicas de uma configuração de item, controle de mudanças para estas características, gravando e relatando mudanças processando e implementando status, e verificando conformidade com especificação de requisitos.” Gerência de Configuração de Software (SCM) da suporte ao ao processo de ciclo de vida do software (IEEE23307.096) beneficiando o gerenciamento de projeto, as atividades de desenvolvimento e manutenção,

atividades de garantia, e a cliente e usuários do produto final.
O conceito de gerência de configuração aplica a todos os itens a serem controlados, embora existam algumas diferenças na implementação entre gerência de configuração de hardware e gerência de configuração de software.
SCM está intimamente relacionado com a atividade de garantia de qualidade de software (SQA).

Como definido na Qualidade de Software KA, processos SQA fornecem garantia que os produtos de software e processos de ciclo de vida projeto estão em conformidade com suas especificações de requisitos pelo planejamento, articulação, e execução de um conjunto de atividade para fornecer confiança adequada que a qualidade está sendo construída no software. Atividades de SCM ajudam na realização das metas de SQA. Em alguns contextos projetos (veja, por exemplo, IEEE730-02), especificam requisitos de SQA descrevendo certas atividades de SCM.

As atividade de SCM são: gerência e planejamento dos processos de SCM, identificação e configuração de software, controle configuração de software, estimar o status da configuração de software, auditoria da configuração de software, e a gestão de lançamento e entrega do software.

2.1.3.6.1 Gerenciamento dos Processos SCM

SCM controla a evolução e integridade de um produto pela identificação de seus elementos, gerenciando e controlando mudanças, e verificando, registrando e apresentação informações sobre configuração. A partir da perspectiva do engenheiro de software, SCM facilita o desenvolvimento e a implementação de atividades de mudança. Uma implementação bem sucedida de SCM requer um cuidadoso planejamento e gerenciamento. Este, por sua vez, requer um entendimento do contexto organizacional para, e as restrições colocadas sobre, o design e implementação dos processos de SCM.

a) Contexto Organizacional para SCM

Para planear um processo de SCM para um projeto, é necessário compreender o contexto organizacional e os relacionamentos entre os elementos organizacionais. SCM interage com várias outras atividades ou elementos organizacionais. Os elementos organizacionais responsáveis pelo apoio aos

processos de engenharia de software podem ser estruturados de várias maneiras. Embora a responsabilidade de realizar certas tarefas SCM pode ser designada para outras partes da organização, como a organização de desenvolvimento, a responsabilidade global pelo SCM muitas vezes recai em um distinto elemento organizacional ou individuo designado. Software é frequentemente desenvolvido como parte de um sistema maior que contenham elementos de hardware e firmware. Neste caso, atividades SCM têm lugar em paralelo com atividades GC de hardware e firmware, e devem ser compatíveis com o nível de GC do sistema. Buckley [Buc96: c2] descreve SCM dentro deste contexto. Note que o firmware contém hardware e software, portanto os conceitos de GC são aplicados a ambos hardware e software.

SCM pode interagir com uma organização atividade garantia de qualidade em questões tais como gerenciamento de registros e itens em não conformidade Em relação ao anterior, alguns itens sob controle SCM podem também ser registros de projeto sujeito ao fornecimento garantia de qualidade do programa da organização. Gerenciamento itens sem conformidade é geralmente da responsabilidade da atividade de garantia de qualidade, entretanto, SCM pode ajudar com monitoramento e relatórios sobre configuração de software itens que se inserem nesta categoria.

Talvez a mais estreita relação com desenvolvimento e manutenção de software nas organizações.

É neste contexto que muitas tarefas de controle e configuração do software são executadas. Muitas vezes, as mesmas ferramentas apoiam o desenvolvimento, manutenção, propósitos de SCM.[Ber92 :c4; Dar90:c2; IEEE828-98:c4s2.1]

b)Limitações e orientações para o processo de SCM

As restrições e orientações para o processo de SCM provêm de várias de fontes. Políticas e procedimentos organizacionais podem influenciar a concepção e implementação dos processo de SCM para um determinado projeto. Além disso, contratos entre o compradores e o fornecedores podem conter disposições que afetam o processo de SCM. Por exemplo, a configuração de algumas auditorias poderia ser necessária, ou poderia especificar que certos itens sejam colocados sob CM. Quando os produtos de software a ser desenvolvidos têm o potencial de afetar a segurança pública, entidades reguladoras podem impor restrições externas (ver, por exemplo, USNRC1.169-97). Finalmente, o ciclo de vida de software escolhido

para um processo de software e as ferramentas de projeto selecionadas para executar o software afetam a concepção e implementação do processo de SCM. [Ber92]

Orientação para o desenvolvimento e a implementação de um processo de SCM pode também ser obtido a partir de "melhores práticas", o que se refletiu em normas da engenharia de software emitidas por diferentes organizações de padronização. Moore [Moo98] fornece um roteiro para estas organizações e as suas normas. As melhores práticas também se reflete na melhoria dos processos e modelos de processo de avaliação, tais como o Software Engineering Institute's Capability Maturity Model Integration (SEI / CMMI) (SEI01) e ISO/IEC15504 Software Engineering Process-Avaliação (ISO / IEC 15504-98). IEEE828-98:c4s1,c4s2.3; Moo98

c) Planejamento para SCM

O planejamento de um processo de SCM para um determinado projeto deve ser coerente com o contexto organizacional, respeitar as restrições e seguir as orientações e a natureza do projeto (por exemplo, o tamanho e criticidade). As principais atividades abrangidas são: Identificação e Configuração de Software, Controle de Configuração de Software, Contagem de Status de Configuração de Software, Audição da Configuração de Software, e Gerenciamento de Release e Entrega do. Além disso, questões como a organização e as responsabilidades, recursos e cronogramas, seleção e utilização de ferramenta, controle de fornecedores e subcontratados, e controle de interface são normalmente considerados.

Os resultados do planejamento das atividades são gravados em um Plano de SCM (SCMP), que normalmente está sujeitos a revisão e auditoria da SQA. Para evitar confusão sobre quem irá realizar determinada atividade ou tarefas de SCM, as organizações envolvidas no processo de SCM precisam ser claramente identificadas. Responsabilidades específicas para determinadas atividades ou tarefas de SCM também precisam ser atribuídas a entidades organizacionais, quer pelo título ou pelo elemento organizacional. A autoridade global e relatórios de critérios para SCM também devem ser identificados, embora isto possa ser realizado no gerenciamento de projetos ou na fase de Planejamento da Garantia da

Qualidade.

O planejamento para SCM identifica os funcionários e as ferramentas envolvidas na realização de atividades e tarefas de SCM. Aborda questões da programação, estabelecendo as sequências necessárias para a criação de tarefas de SCM e à identificação das suas relações com os cronogramas do projeto e das metas estabelecidas na fase de gerenciamento de planejamento do projeto. Qualquer requerimento de treinamento necessário para a implementação dos planos e programas de formação para novos funcionários também são especificados.

Diferentes tipos de ferramenta de capacidades, e os procedimentos para a sua utilização, são apoiados pelas atividades de SCM. Dependendo da situação, a capacidade destas ferramentas podem ser disponibilizadas com alguma combinação de ferramentas manuais, ferramentas automatizadas proporcionando uma capacidade única de SCM, ferramentas automatizadas integrando uma série de SCM (e talvez outras) capacidades, ou ferramenta integrada para os ambientes que servem as necessidades dos vários participantes no processo de engenharia de software (por exemplo, SCM, o desenvolvimento, V & V). Ferramenta automatizada torna-se o apoio cada vez mais importante e cada vez mais difícil de estabelecer, como projetos que crescem em tamanho e como ambientes projetos se tornam mais complexos. Estas capacidades das ferramentas fornecer apoio para:

- A Biblioteca do SCM
- O pedido de mudança no software (SCR) e procedimentos de aprovação
- Código (e produtos relacionados com o trabalho) e mudança das tarefas de gerenciamento
- Relato de Status Configuração de Software e coleção de métricas de SCM
- Auditoria e Configuração de Software
- Gerenciamento e monitoramento da documentação de software
- Performance da construção do software
- Gerenciamento monitoramento da versões de software e da sua entrega

As ferramentas utilizadas nessas áreas também podem fornecer métricas para o processo de melhoria. Royce [Roy98] descreve sete medidas fundamentais de valor no gerenciamento dos processos de engenharia de software. Informações disponíveis, a partir das diversas ferramentas de SCM, refere-se Royce ao trabalho e progresso de indicadores de gerenciamento e sua qualidade de indicadores de

mudança de Tráfico e estabilidade, a Quebra e Modularidade, Retrabalho e Adaptabilidade, e MTBF (tempo médio entre falhas) e maturidade. Elaboração de relatórios sobre estes indicadores podem ser organizados de várias maneiras, por exemplo, item configuração de software ou pelo tipo de alteração solicitada.

Outras ferramentas podem fornecer dados e relatórios de gerenciamento de capacidades para o gerenciamento do desenvolvimento, e atividades de garantia de qualidade. Como foi acima referido, a capacidade dos diversos tipos de ferramenta pode ser integrada com sistemas de SCM, o qual, por sua vez, está intimamente ligado às várias atividades de outros softwares.

No planejamento, engenharia de software SCM seleciona ferramentas adequadas para o trabalho. Planejamento considera questões que possam surgir na implementação destas ferramentas, especialmente se algum tipo de cultura é necessário que mudar. Uma visão geral dos sistemas de SCM e de seleção das considerações é dada em [Dar90: C3, apare], e um estudo de caso sobre a seleção de um sistema de SCM é dada em [Mid97]. As informações complementares sobre ferramentas SCM podem ser encontradas em Engenharia de Software Ferramentas e Métodos KA.

Um projeto de software pode adquirir ou utilizar a aquisição de produtos de software, tais como compiladores e outras ferramentas. SCM considera o planejamento e como estes itens serão adquiridas sob o controle de configuração (por exemplo, integração das bibliotecas no projeto) e como mudanças e atualizações serão avaliadas e gerenciadas.

Considerações semelhantes aplicam-se à terceirização de software. Neste caso, os requisitos SCM devem ser impostos ao subcontratante da SCM, como parte do processo de terceirização e os meios de controlar o cumprimento, também são necessários para ser estabelecido. Este último inclui a consideração de que informações de SCM devem estar disponíveis para o cumprimento da monitoração efetiva

Quando um item de software irá interagir com outros itens de hardwares ou softwares, uma mudança em qualquer item pode afetar outros. O planejamento para SCM considera o processo de como os itens de interface serão identificados e como as mudanças para os itens serão gerenciadas e comunicadas. O papel SCM pode fazer parte de um maior, processo de níveis de sistema para especificação de

interface e controle, e podem incluir especificações da interface, planos de controle de interfaces, documentos de controle de interfaces. Nesse caso, planejamento SCM para controle de interface tem lugar no contexto do nível de sistema do processo. A discussão da performance das atividades do controle de interface é dada em [Ber92: C12]. IEEE 12207.0-96 :c6.s2.1; Som01:c29

d) Plano SCM

Os resultados do planejamento SCM para um determinado projeto são gravados em um plano de gerenciamento de configuração de software (SCMP), um "documento vivo" que serve de referência para o processo de SCM. Ele é mantido (isto é, atualizado e aprovado), conforme necessário durante o ciclo de vida do software. Na implementação do SCMP, é normalmente é necessário desenvolver uma série de definição mais detalhada dos procedimentos para a subordinação de forma específica os requisitos que serão realizados durante as atividades do dia-a-dia.

Orientação sobre a criação e a manutenção de um SCMP, com base nas informações produzidas pelas atividades de planejamento, está disponível a partir de uma variedade de fontes, tais como [IEEE828-98: c4]. Esta referência estabelece requisitos de informação a ser incluída em uma SCMP. Também define e descreve seis categorias de SCM informações a serem incluídas em um SCMP: Buc96:c3; Pau93:L2-81

- Introdução (objetivos, escopo, termos usados)
- Gerenciamento de SCM (organização, responsabilidades, autoridades, políticas aplicadas, diretrizes e procedimentos)
- Atividades de SCM (identificação da configuração, controle de configuração, e assim por diante)
- Cronograma de SCM (coordenação com as outras atividades do projeto)
- Recursos de SCM (ferramentas, recursos físicos e humanos)
- Manutenção de SCMP

e) Monitoramento do Gerenciamento de configuração de software

Depois que o processo de SCM é implementado é necessário certo grau de monitoramento para garantir que as disposições do SCMP sejam corretamente

executadas (veja, por exemplo, [Buc96]). É provável que existam requisitos específicos de SQA para garantir o cumprimento de processos e procedimentos especificados no SCM. Isto poderia envolver uma autoridade de SCM garantindo que as pessoas com responsabilidades atribuídas desempenhem corretamente as tarefas definidas no SCM. A autoridade da garantia da qualidade de software, como parte de uma atividade de auditoria e observância, poderia também desempenhar esse monitoramento.

O uso de ferramentas SCM integrado com capacidade para processar o controle pode tornar a tarefa mais fácil controlar. Algumas ferramentas para facilitar o cumprimento de transformação ao mesmo tempo proporcionar flexibilidade para o engenheiro de software para adaptar os procedimentos. Outras ferramentas para implementar o processo, deixando o engenheiro de software com menos flexibilidade. Acompanhamento e exigências ao nível da flexibilidade de ser fornecido a um engenheiro de software são considerações importantes na ferramenta de seleção.

SCM medidas podem ser projetadas para fornecer informações específicas sobre a evolução do produto, ou fornecer informações sobre o funcionamento dos processos de SCM. Um dos objetivos de acompanhar os processos de SCM é descobrir as oportunidades para melhoria dos processos. Medições dos processos de SCM fornecem um bom meio para monitorar a eficácia das atividades de SCM de forma continua. Essas medições são úteis para caracterizar o estado atual do processo, bem como fornecer uma base para comparações ao longo do tempo. Análises das medições podem produzir conhecimento importante para mudanças e atualizações correspondentes ao SCMP.

Bibliotecas de software e as diversas ferramentas de capacidades do SCM fornecem as fontes para extrair informações sobre as características do processo de SCM (bem como, dispor de projetos e gerenciamento da informação). Por exemplo, informações sobre o tempo necessário para executar diversos tipos de mudanças seriam úteis numa avaliação dos critérios para determinar quais são os níveis de autoridade são ótimas para autorizar certos tipos de mudanças.

Cuidados devem ser tomados para manter o foco no monitoramento sobre os conhecimentos que podem ser derivados a partir das medições, não sobre as medições em si. Discussão de medição do processo e do produto é apresentado no

Processo de Engenharia de Software KA. O Programa de medição de software é descrito no Gerenciamento da Engenharia de Software KA.

As auditorias podem ser realizadas durante o processo de engenharia de software para investigar o estado atual de elementos específicos da configuração ou para avaliar a implementação dos processos de SCM. Auditoria no processo de SCM fornece mais um mecanismo formal para o monitoramento de determinados aspectos do processo e pode ser coordenados com a função de SQA. Veja também subárea 5 Auditoria configuração de Software. [Pau93:L2-87]

2.1.3.6.2 Identificação da Configuração de Software

A atividade de identificação de configuração de software, identifica itens a serem controlados, estabelece esquemas de identificação para os itens e versões deles, e estabelece as técnicas e ferramentas para serem usadas em aquisição e gerenciamento de itens controlados. Estas atividades fornecem o básico para outras atividades SCM.

a) Identificando itens a serem controlados

Um primeiro passo no controle de mudanças é identificar os itens de software a serem controlados. Isto envolve a compreensão da configuração de software dentro do contexto do sistema de configuração, selecionando itens de configuração de software, desenvolvendo uma estratégia para classificação dos itens de software e descrever as suas relações e identificação das linhas de base a ser utilizado, juntamente com o procedimento para a aquisição de uma linha de base sobre os itens .

Uma configuração de software é o conjunto de características físicas e funcionais de software, conforme estabelecido na documentação técnica ou alcançado em um produto (IEEE610.12-90). Ele pode ser visto como parte de uma configuração geral do sistema.

Um item de configuração de software (SCI) é uma agregação de software designado para o gerenciamento de configuração e é tratada como uma entidade única no processo de SCM (IEEE610.12-90). Uma variedade de itens, além do próprio código, normalmente é controlada pelo CSM. itens de software com potencial para se tornar SIC incluem planos, especificações e documentação do projeto,

ensaios de materiais, ferramentas de software de código-fonte e executáveis, bibliotecas de código, dados e dicionários de dados, e documentação para instalação, manutenção, operação e uso de software.

Seleção de sítios de importância comunitária é um importante processo no qual um equilíbrio deve ser alcançado entre dar visibilidade adequada para fins de controle do projeto e fornecendo um número razoável de itens controlados. Uma lista de critérios para a seleção SCI é dada em [Ber92].

As relações estruturais entre a SIC selecionado, e suas partes constituintes, afetam as atividades de SCM ou outras tarefas, como a construção de software ou análise do impacto das mudanças propostas. Adequado controle dessas relações também é importante para apoiar a rastreabilidade. A concepção do sistema de identificação de sítios de importância comunitária devem considerar a necessidade de mapear os elementos identificados com a estrutura de software, bem como a necessidade de apoiar a evolução dos itens de software e seus relacionamentos.

Uma versão de um item de software é um item específico identificado e especificado. Ele pode ser pensado como um estado de um item em evolução. [Con98: C3-C5] Uma revisão é uma nova versão de um item que se destina a substituir a antiga versão do item. A variante é uma nova versão de um item que serão adicionados à configuração sem substituir a versão antiga.

Uma linha de base de software é um conjunto de itens de configuração de software formalmente designado e fixado em um tempo específico durante o ciclo de vida do software. O termo também é usado para se referir a uma versão específica de um item de configuração de software que foi acordado. Em ambos os casos, a base só pode ser alterada através de procedimentos formais de controle de mudança. Uma linha de base, juntamente com todas as alterações aprovadas para a linha de base, representa a configuração atual aprovado. Comumente bases utilizadas são as bases funcionais, atribuídas, de desenvolvimento e do produto (ver, por exemplo, [Ber92]). A linha de base funcional corresponde aos requisitos do sistema são revisados. A linha de base atribuído corresponde à revista especificação de requisitos de software e interface de software especificação de requisitos. A base de desenvolvimento representa a configuração de software em desenvolvimento, por vezes, selecionados durante o ciclo de vida do software. Alterar autoridade para

essa linha de base geralmente cabe principalmente à organização de desenvolvimento, mas pode ser compartilhada com outras organizações (por exemplo, SCM ou de teste). A base do produto corresponde ao produto de software entregue preenchido para integração de sistemas. As linhas de base a ser usada para um determinado projeto, juntamente com seus associados níveis de autoridade necessário para a aprovação da mudança, são geralmente identificadas no SCMP.

Itens de configuração de software são colocados sob controle SCM em momentos diferentes, ou seja, eles são incorporados a uma base particular em um determinado ponto do ciclo de vida do software. O fato gerador é a realização de algum tipo de tarefa a aceitação formal, como uma revisão formal. A Figura 2 caracteriza o crescimento de itens de linha de base como o produto do ciclo de vida. Este valor é baseado no modelo em cascata, para fins de ilustração, os índices usados na figura indicam as versões dos itens de evolução. Após a aquisição de um SCI, alterações para o item deve ser formalmente aprovado como adequado para a SCI e a linha de base em causa, tal como definido na SCMP. Após a aprovação, o item será incorporado no software de base de acordo com o procedimento adequado.

b) Biblioteca de Software

Uma biblioteca de software é um conjunto controlado de software e documentação relacionada com a intenção de auxiliar no desenvolvimento de software, uso e manutenção (IEEE610.12-90). É também fundamental no gerenciamento de software de lançamento e entrega de atividades. Vários tipos de bibliotecas podem ser utilizadas, cada uma correspondendo a um determinado nível de maturidade do item de software. Por exemplo, uma biblioteca de trabalho poderá apoiar codificação e uma biblioteca de apoio ao projeto poderia apoiar testes, enquanto uma biblioteca de mestre pode ser utilizado para produtos acabados. Um nível adequado de controle de SCM (associadas de base e nível de autoridade para a mudança) é associado a cada biblioteca. Segurança, em termos de controle de acesso e as instalações de backup, é um aspecto fundamental da gestão da biblioteca. Um modelo de uma biblioteca de software é descrito em [Ber92: C14].

A ferramenta usada (s) para cada biblioteca deve apoiar o controle SCM precisa dessa biblioteca, tanto em termos de SIC controlar e controlar o acesso à

biblioteca. Ao nível da biblioteca de trabalho, esta é uma capacidade de gerenciamento de código que serve os desenvolvedores, mantenedores, e SCM. É focado na gestão de versões de itens de software, apoiando as atividades de vários desenvolvedores. Em níveis mais altos de controle, o acesso é mais restrito e SCM é o principal usuário.

Essas bibliotecas são também uma importante fonte de informação para a medição do trabalho e do progresso. [Bab86: c2; c5; Buc96: c4; IEEE828-98: c4s3.1; Pau93: L2-82; Som01: C29]

2.1.3.6.3 Software de controle de configuração

Software de controle de configuração está preocupado com o gerenciamento de mudanças durante o ciclo de vida do software. Abrange o processo para determinar quais as mudanças a fazer, a autoridade para aprovar as mudanças, o apoio para a implementação dessas mudanças, e o conceito de desvios formais de requisitos do projeto, bem como a renúncia a eles. Informações resultantes dessas atividades é útil para medir o tráfego de mudança e ruptura, e aspectos de retrabalho.

a) Requerente, Avaliação e aprovação de alterações de software

O primeiro passo para gerir as mudanças de produtos controlados é determinar quais as mudanças a fazer. O processo de solicitação de alteração de software (ver Figura 5), prevê procedimentos formais para a apresentação e gravação de solicitações de mudança, avaliando o custo potencial e o impacto de uma mudança proposta, e aceitando, modificar ou rejeitar as alterações propostas. Os pedidos de alteração de itens de configuração de software pode ser originada por qualquer pessoa em qualquer ponto do ciclo de vida do software e pode incluir uma proposta de solução e a prioridade solicitada. Uma fonte de solicitações de mudança é o início de ação corretiva em resposta a relatórios de problemas. Isso fornece uma oportunidade para acompanhar os defeitos e coletar medidas de atividade se alteram por tipo de alteração. Depois de um SCR é recebida, uma avaliação técnica (também conhecida como análise de impacto) é realizada para determinar a extensão das modificações que seriam necessárias caso o pedido de mudança ser aceitas. Uma boa compreensão das relações entre software (e, possivelmente,

hardware) itens é importante para esta tarefa. Finalmente, uma autoridade estabelecida, compatível com a linha de base afetado, o SCI envolvidos, bem como a natureza da mudança, irá avaliar os aspectos técnicos e gerenciais da solicitação de mudança e quer aceitar, modificar, recusar ou adiar a mudança proposta.

A autoridade para aceitar ou rejeitar as alterações propostas recai sobre uma entidade tipicamente conhecido como um controle de configuração (CCB). Em projetos menores, esta autoridade pode realmente residir com o líder ou um indivíduo atribuído ao invés de uma placa multi pessoa Pode haver vários níveis de autoridade mudar dependendo de uma variedade de critérios, tais como a criticidade do item em causa, a natureza da mudança (por exemplo, o impacto sobre o orçamento e cronograma), ou o ponto atual do ciclo de vida. A composição das CCBs utilizado para um determinado sistema varie em função desses critérios (um representante da SCM estaria sempre presente). Todos os interessados, adequadas ao nível da CCB, estão representados. Quando o escopo da autoridade da CCB é estritamente software, que é conhecido como comitê de controle configuração de software (SCCB). As atividades do CCB são normalmente sujeitas à auditoria da qualidade de software ou de revisão.

Uma solicitação de mudança efetiva de software (SCR) processo requer o uso de ferramentas de apoio e os procedimentos que vão de formulários de papel e um procedimento documentado para uma ferramenta eletrônica para efetuar pedidos de mudança, reforçando o fluxo do processo de mudança, captando as decisões do CCB, e relatar processo de mudança da informação. A ligação entre essa capacidade da ferramenta e do sistema de notificação de problemas pode facilitar o acompanhamento das soluções para os problemas relatados. descrições de processo de mudança e as formas de apoio (informação) são dados em uma variedade de referências, por exemplo [Ber92: c9].

b) Implementação de mudanças de software

Aprovado SCRs são implementados usando os procedimentos definidos por software, em conformidade com os requisitos calendário aplicável. Desde que um número de SCRs aprovados poderão ser implementadas simultaneamente, é necessário fornecer um meio para controlar quais SCRs são incorporadas em versões de software específico e linhas de base. Como parte do encerramento do

processo de mudança, completou mudanças podem sofrer auditorias de configuração e verificação da qualidade de software. Isto inclui assegurar que apenas as alterações aprovadas foram feitas. O processo de solicitação de alteração descrita acima, normalmente o documento SCM (e outras) informações de aprovação para a mudança.

A implementação efetiva de uma mudança é suportada pela capacidade biblioteca de ferramentas, que fornecem o gerenciamento de versões e suporte repositório de código. No mínimo, essas ferramentas fornecem recursos de controle de versão Check-In/Check-Out e associados. Ferramentas mais poderosas podem apoiar o desenvolvimento paralelo e ambientes geograficamente distribuídos. Essas ferramentas podem se manifestar como separar aplicações especializadas sob o controle de um grupo de SCM independente. Eles também podem aparecer como uma parte integrada do ambiente de engenharia de software. Finalmente, podem ser tão elementar como um sistema rudimentar de mudança de controle equipado com um sistema operacional.

c) Desvios e Dispensas

As restrições impostas a um esforço de engenharia de software ou com as especificações produzidas durante as atividades de desenvolvimento podem conter disposições que não podem ser satisfeitas no ponto designado no ciclo de vida. Um desvio é uma autorização para afastar-se uma disposição prévia para o desenvolvimento do item. A renúncia é uma autorização para utilizar um item, a seguir o seu desenvolvimento, que derroga a disposição de alguma forma. Nestes casos, um processo formal é usado para obter a aprovação dos desvios ou isenções de, as disposições. [Ber92: C9; Buc96: c12]

2.1.3.6.4 Software de contabilidade do Status da Configuração

Software de contabilidade status de configuração (SCSA) é o registo e comunicação de informações necessárias para uma gestão eficaz da configuração do software.

a) Status de Configuração de Software da Informação

A atividade SCSA projetos e opera um sistema de captura e transmissão de

informações necessárias como o produto do ciclo de vida. Como em qualquer sistema de informação, as informações de status de configuração a ser gerenciado para a evolução das configurações devem ser identificadas, coletadas e mantidas. Várias informações e as medidas são necessárias para apoiar o processo de SCM e satisfazer as necessidades de status de configuração de relatórios de gestão, engenharia de software, e outras atividades relacionadas. Os tipos de informações disponíveis incluem a identificação configuração aprovada, bem como a identificação e o atual estágio de implantação das mudanças, desvios e isenções. Uma lista parcial dos elementos de informação importante é dada em [Ber92: c10].

Alguma forma de apoio ferramenta automatizada é necessária para realizar a coleta de dados SCSA e tarefas de comunicação. Este poderia ser um recurso de banco de dados, ou pode ser uma ferramenta autônoma ou a capacidade de um ambiente maior ferramenta, integrada. [Buc96: C13; IEEE828-98: c4s3.3]

b) Software de Status de Configuração Reporting

Informações relatadas podem ser usados por vários elementos da organização e do projeto, incluindo a equipe de desenvolvimento, a equipe de manutenção, gerenciamento de projetos e atividades de qualidade de software. Relatórios podem assumir a forma de consultas ad hoc para responder a questões específicas ou a produção periódica de relatórios predefinidos. Algumas informações produzidas pela atividade contábil status no decurso do ciclo de vida pode tornar-se registos de qualidade.

Além de relatar o status atual da configuração, as informações obtidas pela SCSA pode servir como base para várias medidas de interesse para o desenvolvimento, gestão e SCM. Exemplos incluem o número de solicitações de mudança por SCI e o tempo médio necessário para executar uma solicitação de mudança. [Ber92: C10; Buc96: c13]

2.1.3.6.5 Auditoria de Configuração de Software

A auditoria de software é uma atividade realizada de forma independente avaliar a conformidade de produtos de software e processos de regulamentos, normas, diretrizes, planos e procedimentos (IEEE1028-97). As auditorias são realizadas de acordo com um processo bem definido composto de vários papéis e

responsabilidades do auditor. Consequentemente, cada auditoria deve ser planejada com cuidado. Uma auditoria pode exigir um número de indivíduos para executar uma variedade de tarefas ao longo de um período relativamente curto de tempo. Ferramentas para apoiar o planeamento e a condução de uma auditoria pode contribuir para facilitar o processo. Orientação para a realização de auditorias de software está disponível em diversas referências, como [Ber92: C11; Buc96: C15] e (IEEE1028-97).

O software da catividade de auditoria de configuração determina o grau em que um item satisfaça as características exigidas física e funcional. Informal auditorias deste tipo podem ser realizados em pontos-chave do ciclo de vida. Dois tipos de auditorias formais pode ser exigido pelo contrato que rege (por exemplo, em contratos de cobertura de software crítico): a Auditoria de Configuração Funcional (FCA) e da Auditoria de Configuração Física (PCA). A conclusão bem sucedida destas auditorias pode ser um pré-requisito para o estabelecimento da linha de base do produto. Buckley [Buc96: c15] contrasta os efeitos da FCA e PCA em contextos de hardware versus software, e recomenda uma avaliação cuidadosa da necessidade de um software FCA e PCA antes de executá-las.

a) Software de Auditoria de Configuração Funcional

O objetivo do software FCA é garantir que o item de software auditado está em conformidade com as especificações aplicáveis. A saída das catividades de verificação e validação é um contributo essencial para esta auditoria.

b) Software de Auditoria de Configuração Física

O objetivo da auditoria de configuração de software física (PCA) é garantir que a documentação do projeto e de referência é consistente com o produto de software "como construído".

c) Auditorias de processo de uma Baseline Software

Como mencionado acima, as auditorias podem ser realizadas durante o processo de desenvolvimento para investigar o estado atual de elementos específicos da configuração. Neste caso, a auditoria pode ser aplicada a itens de linha de base da amostra para assegurar que o desempenho é consistente com as

especificações ou para assegurar a evolução documentação continua a ser consistente com o ponto inicial de desenvolvimento.

2.1.3.6.6 Software de Gerenciamento de Liberação e Entrega

A "libertação" é utilizado neste contexto para se referir à distribuição de um item de configuração de software fora da atividade de desenvolvimento. Isto inclui a libertação interna, bem como a distribuição aos clientes. Quando as versões diferentes de um item de software estão disponíveis para entrega, como versões para diferentes plataformas ou em versões com capacidades diferentes, é frequentemente necessário para recriar versões específicas de pacotes e os materiais corretos para a entrega da versão. A biblioteca de software é um elemento-chave na realização de tarefas de lançamento e entrega.

a) Construindo software

Construção de software é a atividade de combinar as versões corretas dos itens de configuração de software, utilizando os dados de configuração adequada, em um programa executável para a entrega a um cliente ou outro destinatário, como a atividade de teste. Para sistemas com hardware ou firmware, o programa executável é entregue à atividade de construção do sistema. Construir instruções garantir que o bom construir sejam tomadas medidas e na sequência correta. Além da construção de software para novos lançamentos, geralmente é necessário também para SCM ter a capacidade de reproduzir versões anteriores para a recuperação, testes, manutenção, ou para fins de liberação adicional.

Software é construído usando uma versão particular de ferramentas de apoio, tais como compiladores. Pode ser necessário para reconstruir uma cópia exata de um item previamente construída configuração de software. Neste caso, as ferramentas de apoio e instruções de construção associados precisam estar sob controle de SCM para garantir a disponibilidade das versões corretas dos instrumentos.

A capacidade de ferramenta é útil para selecionar as versões corretas dos itens de software em um ambiente determinado objectivo e para automatizar o processo de construção do software a partir das versões selecionada e os dados de configuração apropriada. Para grandes projetos, com desenvolvimento paralelo ou

em ambientes de desenvolvimento distribuído, esta capacidade ferramenta é necessária. A maioria dos ambientes de engenharia de software fornecem esse recurso. Essas ferramentas variam em complexidade de exigir o engenheiro de software para aprender uma linguagem de script especializada para abordagens de gráficos orientado que escondem muito da complexidade de um "inteligente" construir instalações.

O processo de construção e produtos estão frequentemente sujeitos a verificação da qualidade de software. As saídas do processo de compilação pode ser necessário para referência futura e pode tornar-se registros de garantia da qualidade. [Bab86: C6; Som05: C29]

b) Software de Gerenciamento de Liberação

Software de gerenciamento de liberação compreende a identificação, embalagem e entrega dos elementos de um produto, por exemplo, o programa executável, documentação, notas de lançamento, e os dados de configuração. Dado que as alterações do produto pode ocorrer em uma base contínua, uma preocupação para a gestão de liberação é determinar quando a emitir um comunicado. A gravidade dos problemas abordados pela liberação e medições das densidades de falha de versões anteriores afetar essa decisão. (Som01) A tarefa embalagens devem identificar quais os itens de produtos devem ser entregues, em seguida, selecione as variantes correta desses itens, devido à aplicação prevista para o produto. As informações documentar o conteúdo físico de uma liberação é conhecido como um documento de descrição de versão. As notas de lançamento normalmente descrevem novos recursos, problemas conhecidos, e os requisitos de plataforma necessária para a operação adequada do produto. O pacote a ser lançado também contém instruções de instalação ou atualização. Este último pode ser complicada pelo fato de que alguns usuários atuais pode ter versões que são várias versões antigas. Finalmente, em alguns casos, a atividade de gerenciamento de liberação pode ser necessária para controlar a distribuição do produto para vários clientes ou sistemas de destino. Um exemplo seria um caso em que o fornecedor era obrigado a notificar o cliente de novos problemas relatados.

A capacidade de ferramenta é necessária para suportar essas funções de gerenciamento de liberação. É útil ter uma conexão com a capacidade ferramenta de

apoio ao processo de solicitação de alteração de conteúdo, a fim de mapear a libertação para o SCR que tenham sido recebidos. Esta capacidade ferramenta pode também manter informações sobre as plataformas alvo diferentes e em ambientes de vários clientes.[Som05: C29]

2.1.3.7 Gerência de Engenharia de Software

Gerência de Engenharia de Software pode ser definida como a aplicação das atividades de gerenciamento - planejamento, coordenação, mensuração, monitoração, controle e informação - para garantir que o desenvolvimento e manutenção do software seja sistemático, disciplinado e quantificado (IEEE610.12-90).

A área de conhecimento Gerência da Engenharia de Software, portanto, trata o gerenciamento e a mensuração da engenharia de software. Embora a mensuração seja um aspecto importante de todas as KAs, é aqui que o tópico de mensuração de programas é apresentado.

Embora seja verdade que, em um sentido, deveria ser possível gerenciar a engenharia de software da mesma forma que qualquer outro processo (complexo), existem aspectos específicos aos produtos de software e aos processos do ciclo de vida do software que complicam o gerenciamento efetivo - apenas alguns dos quais são os seguintes:

- A percepção dos clientes é tal que frequentemente existe uma falta de reconhecimento da complexidade inerente à engenharia de software, particularmente em relação ao impacto das mudanças de requisitos.
- É quase inevitável que o processo de engenharia de software por ele próprio gerará a necessidade de requisitos de cliente novos ou mudados.
- Como resultado, o software é muitas vezes construído em um processo iterativo ao invés de em uma sequência de tarefas fechadas.
- Engenharia de software necessariamente incorpora aspectos de criatividade e disciplina - a manutenção de um apropriado equilíbrio entre as duas é muitas vezes difícil.
- O grau de novidade e complexidade do software é muitas vezes extremamente elevado.
- Existe um ritmo rápido de mudanças na tecnologia subjacente.

Com respeito à engenharia de software, atividades de gerenciamento ocorrem em três níveis: gerenciamento organizacional e de infraestrutura, gerência de projetos, e planejamento e controle do programa de mensurações. As duas últimas são cobertas em detalhes na descrição desta KA. Entretanto, isto não é para diminuir a importância das questões de gerenciamento organizacional.

Como o link às disciplinas relacionadas - obviamente gerenciamento - é importante, ele será descrito em maior detalhe do que nas descrições das outras KA. Aspectos de gerência organizacional são importantes em termos de seu impacto na engenharia de software - sobre a política de gestão, por exemplo: padrões e políticas organizacionais fornecem a estrutura na qual a engenharia de software é realizada. Essas políticas podem precisar ser influenciadas pelos requisitos do efetivo desenvolvimento e manutenção do software, e um número específico de políticas de engenharia de software podem ter que ser estabelecidas para o efetivo gerenciamento da engenharia de software no nível organizacional. Por exemplo, políticas normalmente são necessárias para estabelecer processos específicos em escala organizacional ou procedimentos para tais tarefas de engenharia de software como projeto, implementação, estimativas, monitoramento e "informes/relatórios". Tais políticas são essenciais para a gerência efetiva e de longo termo da engenharia de software, pela definição de uma base consistente sobre a qual analisar a performance passada e implementar as melhorias, por exemplo.

Outro aspecto importante do gerenciamento é a gestão de pessoal: políticas e procedimentos para contratar, treinar, e motivar pessoal e mentalizar o desenvolvimento da carreira são importantes não somente no nível do projeto mas, também, no sucesso de longo prazo de uma organização. Pessoal de engenharia de software pode ter treinamento único ou gerenciamento das mudanças de pessoal (por exemplo, mantendo a atualização num contexto onde a tecnologia subjacente sofre mudanças rápidas e contínuas. Gerência de comunicação é também muitas vezes mencionada como um aspecto despercebido porém importante de performance dos indivíduos num campo onde o entendimento preciso das necessidades do usuário e dos requisitos e projetos complexos é necessário. Finalmente, a gerência de portfólio, que é a capacidade de ter uma visão geral não somente do conjunto de software em desenvolvimento mas, também, do software já em uso em uma organização, é necessária. Além do mais, reuso de software é o

fator chave na manutenção e melhoria da produtividade e competitividade. Reuso efetivo requer uma visão estratégica que reflete a força única e as exigências desta técnica.

Em adição ao entendimento dos aspectos do gerenciamento que são unicamente influenciados pelo software, engenheiros de software devem ter algum conhecimento dos aspectos mais genéricos, mesmo nos quatro primeiros anos após a graduação que são *rotulados* no Guia.

Comportamento e cultura organizacional, e gerência funcional da empresa em termos de aquisições, gestão da cadeia de fornecedores, marketing, vendas, e distribuição, todos têm influência, mesmo que indireta, no processo de engenharia de software da organização.

Relevante para esta KA é a noção da gerência de projeto, como 'a construção de artefatos de software úteis' normalmente é gerida na forma de (talvez programas de) projetos individuais. Neste sentido, encontramos amplo apoio no Guia para o Conjunto de Conhecimentos da Gerência de Projetos (PMBOK) (PMI00), o qual por si só inclui as seguintes KA: gerência de integração do projeto, gerência de escopo do projeto, gerência de tempo do projeto, gerência de custo do projeto, gerência da qualidade do projeto, gerência de recursos humanos do projeto, e gerência de comunicações do projeto. Evidentemente, todos esses tópicos têm relevância direta para a área de conhecimento de Gerência de Engenharia de Software. Tentar duplicar o conteúdo do Guia do PMBOK aqui seria tanto impossível quanto inapropriado. Ao contrário, sugerimos que o leitor interessado na gerência de projeto além do que é específico para os projetos de engenharia de software consulte o PMBOK. Gerência de projetos é também encontrada no capítulo das Disciplinas Relacionadas à Engenharia de Software.

A KA Gerência de Engenharia de Software consiste tanto do processo de gerência de projeto de software, nas suas primeiras cinco subáreas, quanto da mensuração da engenharia de software na sua última subárea Embora essas duas disciplinas sejam muitas vezes consideradas como sendo separadas, e sem dúvida elas possuem muitos aspectos únicos, suas estreitas relações levaram a um tratamento combinado nesta KA. Infelizmente, uma percepção comum na indústria de software é que ela entrega produtos com atraso, além do custo, de baixa qualidade e de funcionalidade incerta. Gerência de mensuração informada - um

princípio assumido de qualquer verdadeira disciplina de engenharia - pode ajudar a mudar essa percepção. Na essência, gerenciamento sem mensuração, qualitativa e quantitativamente, sugere uma falta de rigor, e mensuração sem gerenciamento sugere uma falta de finalidade ou contexto. Da mesma forma, entretanto, gerenciamento e mensuração sem conhecimento técnico especializado é inútil, por isso, devemos ter cuidado para evitar o excesso de ênfase nos aspectos quantitativos da Gerência de Engenharia de Software (GES). A gerência efetiva requer a combinação de ambos: números e experiência.

As seguintes definições de trabalho são adotadas aqui:

- Gerência de processo refere-se às atividades que são feitas com o fim de garantir que os processos de engenharia de software são realizados de uma maneira consistente com as políticas, objetivos e padrões da organização.
- Mensuração refere-se à atribuição de valores e rótulos aos aspectos de engenharia de software (produtos, processos e fontes são definidos por [Fen98]) e os modelos que são derivados a partir deles, se esses modelos são desenvolvidos usando estatística, conhecimento técnico especializado ou outras técnicas.

As subáreas da gerência de projetos da engenharia de software fazem uso extensivo da subárea de medição de engenharia de software.

Não inesperadamente, esta KA está intimamente relacionada a outras no Guia SWEBOK e ler as descrições das seguintes KAs em conjunto com esta seria particularmente útil:

- *Requisitos de Software*, onde algumas das atividades a serem realizadas durante a definição da fase de Iniciação e Escopo do projeto são descritas.
- *Gerência de Configuração de Software*, como esta trata a identificação, controle, status de contabilização e auditoria da configuração do software junto com a gerência de lançamento e entrega.
- *Processo de Engenharia de Software*, porque processos e projetos são intimamente ligados (esta KA também descreve a medição de processos e produtos).
- *Qualidade de Software*, como qualidade é uma meta constante da gerência e é um objetivo de muitas atividades que devem ser gerenciadas.

Como a KA de Gerência de Engenharia de Software é vista aqui como um processo organizacional o qual incorpora a noção de gerência de processo e de projetos, nós criamos uma estrutura que é tanto baseada em tópicos quanto baseada no ciclo de vida. Entretanto, a principal base para a estrutura de tópico de nível mais superior é o processo de gerência de um projeto de engenharia de software. Existem seis grandes subáreas As cinco primeiras subáreas seguem largamente o Processo de Gerenciamento da IEEE/EIA 12207. As seis subáreas são:

- *Iniciação e definição de escopo*, a qual vai de encontro com a decisão de iniciar um projeto de engenharia de software.
- *Planejamento do projeto de software*, a qual orienta as atividades empreendidas para preparar para o sucesso a engenharia de software a partir de uma perspectiva de gerenciamento.
- *Formalização do projeto de software*, a qual aborda as atividades de gerência de engenharia de software geralmente aceitas que ocorrem durante a engenharia de software.
- *Análise e avaliação*, a qual trata da garantia de que o software seja satisfatório.
- *Fechamento* (encerramento), que trata das atividades de pós-realização de um projeto de engenharia de software.
- *Mensuração da engenharia de software*, a qual aborda o desenvolvimento e implementação efetiva de programas de mensuração nas organizações de engenharia de software (IEEE12207.0-96).

A estrutura de tópicos para a KA de Gerência de Engenharia de Software é mostrada na Figura 22.

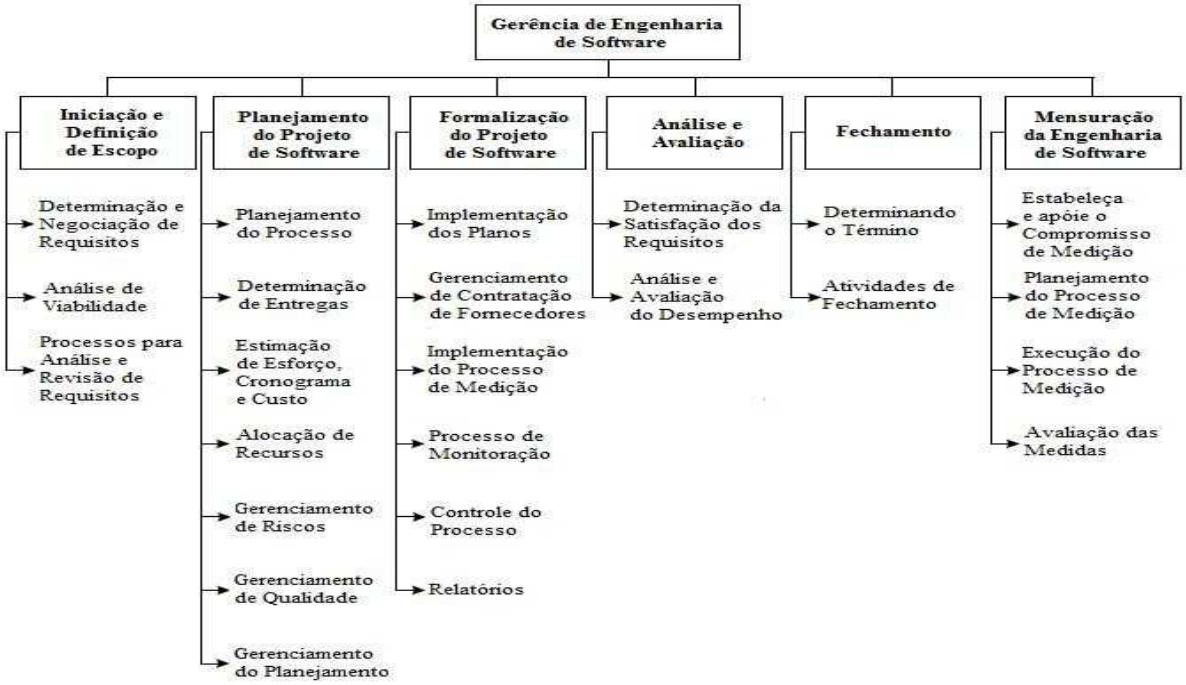


Figura 1 Estrutura de tópicos para a área de conhecimento de Gerência de Engenharia de Software.

Figura 22: Tópicos da KA Gerência de Engenharia de Software

2.1.3.7.1 Iniciação e Definição de escopo

O foco deste conjunto de atividades está na eficaz determinação de exigências de software via métodos de elicitação e avaliação do desenvolvimento da prática do projeto, olhando de vários pontos de vista. Uma vez a prática sendo estabelecida, as tarefas restantes dentro deste processo passam a ser a especificação de requisitos e modificação dos procedimentos (ver também as Exigências de Software KA).

a) Determinação e Negociação de Requisitos

Métodos de requerimentos de software para elicitação de requisitos (por exemplo, observação), análise (por exemplo, modelagem de dados, modelagem de casos de uso), especificação, e validação (por exemplo, prototipagem) deve ser selecionado e aplicado, considerando-se as perspectivas de todas partes interessadas. Isto leva à determinação de alcance de projeto, objetivos e constrangimentos. Isto exige, muitas vezes alguns "campos" de estimação, esforço e custo baseados em métodos adequados (por exemplo, técnicas de analogia informadas por perito).[Dor02: v2c4; Pfl01: c4; Pre04: c7; Som05: c5]

b) Análise de Viabilidade (técnica, operacional, financeira, Política/Social)

Engenheiros de software devem estar seguros sobre a capacidade adequada e se os recursos estão disponíveis (pessoas, especialização, instalações, infraestrutura e apoio) quer interna ou externamente, para garantir que o projeto tenha êxito em tempo oportuno e eficaz em termos de custos (usando, por exemplo, um requisito capacidade matriz). Isto muitas vezes necessita alguma estimativa "aproximada" de esforço e preço baseado em métodos apropriados (por exemplo, informação de perito e analogia técnica).[Pre04: c6; Som05: c6]

c) Processo para Análise e Revisão de Requisitos

Dada a inevitabilidade da mudança, é fundamental que o acordo entre as partes interessadas seja realizado no início. Para isso as exigências e o acordo devem ser revisados (por exemplo, através de mudanças na gestão dos procedimentos). Isto implica, claramente, que os requisitos não serão "gravados na pedra", mas podem e devem ser revistos em pontos como o processo se desdobra (por exemplo, na revisão de desenhos, revisão de gerenciamentos). Se mudanças são aceitas, então alguma forma de rastreabilidade, análise de risco (ver tópico 2.5 Gestão de Riscos) deveria ser utilizada para aferir o impacto dessas mudanças. A gestão de abordagem das mudanças deveria ser igualmente útil se na hora de analisar o resultado do projeto, o escopo e requisitos formam uma base para a avaliação do sucesso. [Som05: c6] Ver também as subáreas de configuração de controle de software em Software Configuration Management KA.

2.1.3.7.2 Planejamento de Projeto de Software

O processo de planejamento iterativo é informado pelo alcance e exigências e pelo estabelecimento de praticabilidade. Nesse ponto, os processos de ciclo de vida de software são avaliados e a parte mais apropriada (dado a natureza do projeto, o seu grau de novidade, a sua complexidade funcional e técnica, as suas exigências de qualidade, e assim por diante) é selecionada. Onde relevante, o próprio projeto é então planejado na forma da decomposição hierárquica de tarefas, os resultados finais de cada tarefa são especificados e caracterizados em termos de qualidade e outros atributos de acordo com determinadas exigências, e esforço detalhado, horário, e preço a estimativa é empreendida. Os recursos então são alocados às

tarefas para otimizar produtividade de pessoal (no individual, equipe, e níveis organizacionais), equipamento e utilização de materiais, e aderência para planejar. A gerência de riscos detalhada é empreendida e “o perfil de riscos” do projeto é discutido e aceito por todas as partes relevantes interessadas. Na gerência de qualidade de software abrangente os processos são determinados como parte do processo de planejamento na forma de procedimentos e responsabilidades por software asseguramento da qualidade, verificação e validação, revistas, e auditorias (ver a Qualidade de Software KA). Como um processo iterativo, é vital que os processos e responsabilidades pela gerência de plano contínuo, a análise, e a revisão sejam, também claramente afirmadas e aceitas.

a) Planejamento de Processo

Seleção do modelo de ciclo de vida de software apropriado (por exemplo, espiral, prototipação evolutiva) e a adaptação e implantação de software de ciclo de vida por processos apropriados, são realizadas em função das especialidades dos requisitos do projeto. Métodos relevantes e os instrumentos também são selecionados. [Dor02: v1c6, v2c8; Pfl01: c2; Pre04: c2; Rei02: c1, c3, c5; Som05: c3; Tha97: c3] Ao nível do projeto, ferramentas e métodos apropriados são utilizados para decompor o projeto em funções, com os insumos, resultados, conclusão e condições de realização (por exemplo, estrutura de emergência de trabalho). [Dor02: v2c7; Pfl01: c3; Pre04: c21;Rei02: c4,c5; Som05: c4; Tha97: c4,c6] Este, por sua vez, influencia as decisões na programação e organização estrutural do projeto.

b) Determinar os Resultados Finais

O(s) produto(s) de cada tarefa (por exemplo, desenho arquitetônico, relatório de inspeção) são especificados e caracterizados. [Pfl01: c3; Pre04: c24; Tha97: c4] Oportunidades de reutilizar componentes de software previamente desenvolvidos ou a utilização de produtos de software são avaliados. Uso de terceiros, bem como adquirir software é planejado e fornecedores são selecionados.

c) Esforço, Programação, e Estimativa de Preço

Com base na repartição de tarefas, entradas, saídas e, a variedade necessária de esforço esperada para cada tarefa é determinada usando uma

estimativa baseada no modelo histórico do tamanho do esforço quando os dados disponíveis e relevantes, ou outros métodos como peritos são avaliados. As dependências de tarefa são estabelecidas e os gargalos potenciais são identificados usando métodos convenientes (por exemplo, análise de caminho crítico). Os gargalos são resolvidos onde possível e o tempo estimado de tarefas com tempos de partida projetados, durações, e tempo limite para produção são produzidos (por exemplo, um gráfico PERT). Requerimentos de recursos (pessoas, ferramentas) são traduzidos em estimativas de custos. [Dor02: v2c7; Fen98: c12; Pfl01: c3; Pre04: c23,c24; Rei02: c5,c6; Som05: c4,c23; Tha97: c5] Isto é uma atividade altamente iterativa que deve ser negociada e revisada até que o consenso seja conseguido entre todas as partes afetadas (principalmente engenharia e gerência).

d) Alocação de Recurso

Equipamentos, instalações, e as pessoas estão associados à tarefas agendadas, incluindo a atribuição de responsabilidades para a conclusão (utilizando, por exemplo, um gráfico Gantt). Esta atividade é informada e condicionada pela disponibilidade de recursos e a sua utilização otimizada, nestas condições, bem como pelas questões relativas ao pessoal (por exemplo, produtividade dos indivíduos / equipes, equipe dinâmica e estruturas organizacionais).

e) Gestão de Riscos

Identificação e análise de risco (o que pode dar errado, como e por quê, e quais são as consequências prováveis), avaliação crítica de riscos (quais são os riscos mais significativos em termos de exposição, o que podemos fazer em termos de iniciação), compensação de risco e contingência de planejamento (formulação de uma estratégia para lidar com riscos e para gerenciar o perfil de risco) são todas realizadas. Métodos de avaliação de riscos (por exemplo, árvores de decisão e simulações de processo) devem ser usados para destacar e avaliar os riscos. A política de abandono de projeto deve ser determinada neste ponto na discussão com todas partes interessadas. [Dor02: v2c7; Pfl01: c3; Pre04: C25; Rei02: C11; Som05: c4; Tha97: c4]. Aspectos de riscos de software, como a tendência de engenheiros de software em acrescentarem recursos indesejáveis ou os riscos ligados à natureza intangível do software, devem influenciar a gestão de risco do projeto.

f) Gestão de Qualidade

A qualidade é definida em termos de atributos pertinentes ao projeto específico e qualquer produto(s) associado(s), possivelmente tanto em requisitos quantitativos quanto qualitativos. Essas características de qualidade serão determinadas na especificação dos requerimentos detalhados do software. Veja também os Requerimentos de Software KA. Os limiares para aderência à qualidade são definidos para cada indicador de acordo com as expectativas dos intervenientes com relação ao software. Procedimentos relativos à SQA em curso ao longo do processo e à verificação e validação de produto (realizações) são também especificadas neste estágio (por exemplo, revisões e inspeções técnicas) (veja também Qualidade de Software KA).

g) Gestão de Plano

Como o projeto e o plano serão geridos também deve ser planejado. Reportando, monitorando e controlando o projeto deve estar de acordo com o processo de engenharia de software selecionado e às realidades do projeto, e devem ser refletidos nos variados artefatos que serão usados para geri-lo. Mas, num ambiente onde a mudança é esperada ao invés de um choque, é vital que os planos sejam geridos por si sós. Isso exige que a aderência aos planos seja sistematicamente dirigida, monitorada, revista, reportada, e, onde apropriado, revisada. Planos associados a outros processos de suporte orientados para gestão (por exemplo, documentação, gestão de configuração de software, e solução de problema) também precisam ser geridos da mesma maneira.

2.1.3.7.3 Formalização do Projeto de Software

Os planos são, então, implementados e os processos consubstanciados nos planos são formalizados.

Em tudo, há o foco na aderência aos planos, com uma expectativa prioritária de que tal aderência irá levar à bem sucedida satisfação dos requisitos do stakeholder e ao alcance dos objetivos do projeto. Fundamental para a formalização são as atuais atividades de mensuração, monitoração, controle e relatórios (divulgação de informações).

a) Implementação dos Planos

O projeto é iniciado e as atividades do projeto são realizadas de acordo com o cronograma. No processo, recursos são utilizados (por exemplo, esforço pessoal, financiamento) e entregas são produzidas (por exemplo, documentos de projeto da arquitetura, casos de teste).

b) Gerência de Contrato de Fornecedores

Prepare e execute contratos com fornecedores, monitore o desempenho dos fornecedores, e aceite o fornecedor de produtos, incorporando-os como for apropriado.

c) Implementação da Mensuração de Processos

O processo de mensuração é formalizado junto com o projeto de software, assegurando que os dados relevantes e úteis são coletados.

d) Acompanhamento do Processo

A aderência aos variados planos é continuamente avaliada em intervalos pré-determinados. Resultados e condições de conclusão de cada tarefa são analisados. Entregas são avaliadas em termos de suas características requeridas (por exemplo, através de revisões e auditorias). Esforço despendido, adesão ao cronograma e custos para entrega em dia são investigados e o recurso utilizado é examinado. O perfil de risco do projeto é revisto e a aderência aos requisitos de qualidade é avaliada.

Medições de dados são modeladas e analisadas. Análise de variações baseada nos desvios dos resultados e valores atuais e esperados é efetuada. Isto pode ser feito na forma de custos extrapolados, cronograma ultrapassado e coisas parecidas. Identificações atípicas, análise da qualidade e outras mensurações de dados são realizadas (por exemplo, análise da densidade dos defeitos). A influência de e a exposição a riscos são recalculadas e árvores de decisão, simulações e outros mais são refeitos à luz dos novos dados. Essas atividades habilitam a detecção de problemas e a identificação de exceções com base nos limites excedidos. Os resultados são relatados à medida do é preciso e, com certeza, onde

os limites aceitáveis são superados.

e) Controle do Processo

Os resultados das atividades de acompanhamento do processo fornecem as bases nas quais as ações de decisão são tomadas. Onde for apropriado, e onde o impacto e os riscos associados forem modelados e gerenciados, as mudanças podem ser feitas no projeto. Isto pode ter a forma de ações corretivas (por exemplo, retestar certos componentes), ela pode envolver a incorporação das contingências de forma que ocorrências similares sejam evitadas (por exemplo, a decisão de usar prototipação para ajudar na validação dos requisitos do software), e/ou ela pode acarretar a revisão de vários planos e documentos do projeto (por exemplo, especificação de requisitos) para acomodar os resultados inesperados e suas implicações.

Em alguns casos, ela pode levar ao abandono do projeto. Em todos os casos, procedimentos de controle de mudanças e gerência de configuração de software são cumpridos (veja também a KA de Gerência de Configuração de Software) para que as decisões sejam documentadas e comunicadas a todos os interessados, planos são revistos e revisados quando preciso e dados importantes são gravados na base de dados central.

f) Relatório

Nos períodos especificados e acordados, o cumprimento dos planos é relatado, tanto internamente para a organização (por exemplo, para o comitê diretor de portfólios de projeto) quanto externamente para os stakeholders (por exemplo, clientes, usuários). Relatórios desta natureza devem focar-se no cumprimento geral em oposição aos relatórios detalhados frequentemente exigidos pela equipe interna do projeto.

2.1.3.7.4 Análise e Avaliação

Em pontos críticos do projeto, o progresso geral voltado para o alcance dos objetivos definidos e para a satisfação dos requisitos do stakeholder é avaliado. Analogamente, avaliações da efetividade do processo como um todo para prazos, pessoal envolvido e ferramentas e métodos empregados também são realizados nos

marcos particulares.

a) Determinando a Satisfação dos Requisitos

Desde que a obtenção da satisfação do stakeholder (usuário e cliente) é um dos nossos principais objetivos, é importante que o progresso deste objetivo seja formal e periodicamente avaliado. Isto ocorre no atingimento dos principais marcos do projeto (por exemplo, confirmação da arquitetura do projeto de software, análise técnica da integração do software). Variações das expectativas são identificadas e as ações adequadas são tomadas. Como acima, na atividade de controle do processo (veja tópico *3.5 Controle do Processo*), em todos os casos procedimentos de controle de mudanças e gerência de configuração de software são cumpridos (veja também a KA de Gerência de Configuração de Software) para que as decisões sejam documentadas e comunicadas a todos os interessados, planos são revistos e revisados quando preciso e dados importantes são gravados na base de dados central (veja também 6.3 Execução do Processo de Medição/'Mensuração'). Mais informações podem ser encontradas na KA de Teste de Software, tópico 2.2 Objetivos de Testes e na KA de Qualidade de Software, tópico 2.3 Revisões e Auditorias.

b) Analisando e Avaliando o Desempenho/Performance

Revisões periódicas de desempenho para o pessoal do projeto fornecem esclarecimentos quanto a probabilidade de cumprimento dos planos tanto quanto a possíveis áreas de dificuldade (por exemplo, conflitos entre membros da equipe). Os vários métodos, ferramentas, e técnicas empregados são avaliados por sua efetividade e adequação, e o processo por si só é sistemática e periodicamente avaliado por sua relevância, utilidade e eficácia no contexto do projeto Onde apropriado, mudanças são feitas e gerenciadas.

2.1.3.7.5 Fechamento

O projeto alcança o fim quando todos os planos e processos envolvidos tenham sido formalizados e completados. Neste estágio, o critério para o sucesso do projeto é revisto. Uma vez que o fechamento/término seja estabelecido, atividades post mortem, de melhoria do processo e de arquivamento são realizadas.

a) Determinando o Fechamento

As tarefas como especificada nos planos são completadas e o alcance satisfatório da completude dos critérios é confirmado. Todos os produtos planejados são entregues com características aceitáveis. Requisitos são verificados e confirmados como satisfeitos, e os objetivos do projeto são alcançados. Esses processos normalmente envolvem todos os stakeholders e resultam na documentação da aceitação do cliente e quaisquer relatos de problemas remanescentes reconhecidos.

b) Atividades de Fechamento

Após o fechamento ser confirmado, o arquivamento dos materiais do projeto toma lugar alinhado com a concordância do stakeholder quanto a métodos, localização e duração. A base de dados de mensurações/medições da organização é atualizada com os dados finais do projeto e análises pós-projeto são realizadas. O post mortem de um projeto é feito de forma que questões, problemas e oportunidades encontradas durante o processo (particularmente através de revisões e avaliações, veja subárea *4 Análises e Avaliações*) são analisadas, e lições são desenhadas a partir do processo e alimentam o aprendizado organizacional e melhoram os empreendimentos. (veja também a KA Processo de Engenharia de Software).

2.1.3.7.6 Mensuração/Medição de Engenharia de Software

A importância da mensuração e sua função nas melhores práticas de gerenciamento é vastamente reconhecida e, assim, sua importância tende a crescer nos anos vindouros. Mensuração efetiva tem se tornado uma das pedras fundamentais da maturidade organizacional.

Termos chave na medição de software e nos métodos de medição tem sidos definidos na [ISO 15939-02] com base no vocabulário ISO internacional de mensuração [ISO93]. Contudo, leitores irão encontrar diferenças de terminologia na literatura; por exemplo, o termo métrica é algumas vezes usado no lugar de medidas.

Este tópico segue o padrão internacional ISO/IEC 15939, o qual descreve um processo que define as atividades e tarefas necessárias para implementar um processo de mensuração de software bem como inclui um modelo de informações

de mensuração.

a) Estabeleça e Sustente o Compromisso de Medições

Aceite as exigências por mensurações/medições. Cada empreendimento de medição deve ser guiado por objetivos organizacionais e orientado por um conjunto de requisitos de medições estabelecidos pela organização e pelo projeto. Por exemplo, um objetivo organizacional deve ser "ser o primeiro no mercado com novos produtos." [Fen98: c3,c13; Pre04: c22] Isto, por sua vez, poderia gerar um exigência de que os fatores que contribuam para esse objetivo sejam medidos de forma que os projetos possam ser gerenciados para ir de encontro a esse objetivo.

Defina o escopo da mensuração. A unidade organizacional na qual cada exigência de medição é para ser aplicada deve ser definida. Isso pode consistir de uma área funcional, um único projeto, um único site ou mesmo a empresa inteira. Todas as tarefas subsequentes relacionadas a esse requisito devem estar dentro do escopo definido. Adicionalmente, os stakeholders devem ser identificados.

Compromisso da gerência e equipe para medições. O compromisso deve ser formalmente estabelecido, comunicado e apoiado por recursos (veja o próximo item).

Comprometa/confirme recursos para as medições. O compromisso da organização para as medições é um fator essencial de sucesso, como evidenciado pela alocação de recursos para implementar o processo de mensuração/medição. A alocação de recursos inclui a designação de responsabilidade para as várias tarefas do processo de mensuração (tais como usuário, analista e documentador) e o fornecimento de financiamento adequado, treinamento, ferramentas, e apoio para conduzir o processo como uma moda duradoura.

b) Planeje o Processo de Mensuração/Medição

Caracterize a unidade organizacional. A unidade organizacional fornece o contexto para a mensuração/medição, então ela é importante para tornar esse contexto explícito e para articular as proposições que ela incorpora e as restrições que a ela são impostas. A caracterização pode ser em termos de processos organizacionais, domínios de aplicações, tecnologia e fronteiras organizacionais. Um modelo de processo organizacional também é tipicamente um elemento de caracterização da unidade organizacional [ISO 15939-02: 5.2.1].

Identifique as informações necessárias. Informações necessárias são baseadas nas metas, restrições, riscos e problemas da unidade organizacional. Elas podem ser derivadas a partir do negócio, da organização, regulamentação e/ou objetivos do produto. Elas devem ser identificadas e priorizadas. então, um subconjunto a ser tratado deve ser selecionado e seus resultados documentados, comunicados e revisados pelos stakeholders [ISO 15939-02: 5.2.2].

Selecione as medidas. Medidas candidatas devem ser selecionadas, com vínculos claros às informações necessárias. Medidas devem, então, ser selecionadas baseado nas prioridades das informações necessárias e outros critérios tais como custo da coleta, grau de ruptura do processo durante a coleta, facilidade de análise, facilidade de obtenção de acurácia, consistência dos dados, e outros mais [ISO 15939-02: 5.2.3 e Apêndice C].

Defina os procedimentos de coleta de dados, análise e relatórios. Isso engloba os procedimentos de coleta e cronograma, armazenamento, verificação, análise, divulgação de relatórios e gerência de configuração dos dados [ISO 15939-02: 5.2.4].

Defina os critérios para a avaliação dos produtos de informação. Critérios para avaliação são influenciados pelos objetivos técnicos e de negócio da unidade organizacional. Produtos de informação incluem aqueles associados com o produto sendo produzido, bem como aqueles associados com os processos em uso para gerenciar e medir o projeto [ISO 15939-02: 5.2.5 e Apêndices D e E].

Revise, aprove e forneça recursos para as tarefas de mensuração/medição. O plano de medições deve ser revisado e aprovado pelos stakeholders apropriados. Isso inclui todos os procedimentos de coleta de dados, armazenamento, análise e procedimentos de divulgação de relatórios; critérios de avaliação; cronograma e responsabilidades. Critérios para avaliação desses artefatos devem ser estabelecidos no nível da unidade organizacional ou nível superior e devem ser usados como base para essas revisões. Tais critérios devem considerar as experiências anteriores, a disponibilidade de recursos e o potencial de interrupção do projeto quando mudanças nas práticas atuais são propostas. A aprovação demonstra o compromisso com o processo de medição [ISO 15939-02: 5.2.6.1 e Apêndice F].

Recursos devem ser disponibilizados para a implementação das tarefas de

mensuração/medição planejadas e aprovadas. A disponibilidade de recursos pode ser especulada/experimentada nos casos onde as mudanças estão para ser praticadas antes de um amplo emprego dessas mudanças. Considerações devem ser feitas quanto aos recursos necessários para o emprego com sucesso dos novos procedimentos ou medidas [ISO 15939-02: 5.2.6.2].

Adquira e empregue tecnologias de apoio. Isto inclui a avaliação de tecnologias de apoio disponíveis, seleção das tecnologias mais apropriadas, aquisição e emprego dessas tecnologias [ISO 15939-02: 5.2.7].

c) Execute o Processo de Mensuração/Medição

Integre os procedimentos de medição com os processos relevantes. Os procedimentos de medição, tais como coleta de dados, devem ser integrados nos processos que eles estão medindo. Isto pode envolver a mudança dos processos atuais para acomodar a coleta de dados ou geração de atividades. Ela pode envolver também a análise dos processos atuais para minimizar os esforços adicionais e a avaliação do efeito sobre os empregados para garantir que os procedimentos de medição serão aceitos. Questões morais e outros fatores humanos precisam ser considerados. Adicionalmente, os procedimentos de medidas devem ser comunicados àqueles que fornecem os dados, pode ser preciso providenciar treinamento, e apoio/suporte deve tipicamente ser providenciado. Análise de dados e procedimentos de divulgação de relatórios devem tipicamente ser integrados aos processos e/ou projetos organizacionais de uma forma similar [ISO 15939-02:5.3.1].

Colete dados. Os dados devem ser coletados, verificados e armazenado [ISO 15939-02: 5.3.2].

Analise os dados e desenvolva as informações de produtos. Dados podem ser agregados, transformados ou registrados como parte do processo de análise, usando um nível de rigor adequado à natureza dos dados e informações necessárias. Os resultados dessas análises são tipicamente indicadores que devem ser interpretados, resultando em conclusões iniciais a serem apresentadas aos stakeholders. Os resultados e conclusões devem ser revisados, usando um processo definido pela organização (o qual pode ser formal ou informal). Fornecedores dos dados e usuários das medidas devem participar na revisão dos

dados para garantir que elas são significativas e acuradas/precisas e que elas podem resultar em ações razoáveis [ISO 15939-02: 5.3.3 e Apêndice G].

Comunique os resultados. As informações dos produtos devem ser documentadas e comunicadas aos usuários e stakeholders [ISO 15939-02: 5.3.4].

d) Avalie as Medidas

Avalie as informações dos produtos. Avalie as informações dos produtos contra os critérios de avaliação especificados e determine os pontos fortes e fracos das informações dos produtos. Isto pode ser realizado por um processo interno ou por uma auditoria externa e deve incluir o feedback (retorno) dos usuários das medições. Registre as lições aprendidas numa base adequada [ISO 15939-02: 5.4.1 e Apêndice D].

Avalie o processo de mensuração/medição. Avalie o processo de medição contra os critérios de avaliação especificados e determine os pontos fortes e fracos do processo. Isto pode ser realizado por um processo interno ou por uma auditoria externa e deve incluir o feedback (retorno) dos usuários das medições. Registre as lições aprendidas numa base adequada [ISO 15939-02: 5.4.1 e Apêndice D].

Identifique melhorias potenciais. Tais melhorias podem ser mudanças nas formas dos indicadores, mudanças nas unidades de medida ou reclassificação das categorias. Determine os custos e benefícios das melhorias potenciais e selecione as ações de melhoria adequadas. Comunique as proposições de melhoria aos proprietários do processo de medição e aos stakeholders para revisão e aprovação. Também comunique a falta de melhorias potenciais caso a análise falhe ao identificar as melhorias [ISO 15939-02: 5.4.2].

2.1.3.8 Processo de Engenharia de Software

O processo de engenharia de software KA pode ser estudado em 2 níveis. O primeiro nível engloba as atividades técnicas e gerenciais dentro do processo do ciclo de vida do software que são realizados durante aquisição, desenvolvimento, manutenção e retirada. O segundo é o meta nível, que se concentra na definição, implementação, avaliação, mensuração, gerência, mudança, melhoria do processo de ciclo de vida software em si. O primeiro nível é coberto por outra guia KA. Este KA está concentrado no segundo.

O termo “processo de engenharia de software” pode ser interpretado de diferentes maneiras, e isto pode causar confusão. A primeira maneira, onde a palavra “o” é usada, como “o” processo de engenharia de software, poderia implicar que existe somente uma maneira correta de se executar tarefas de desempenho de engenharia de software. Este significado é evitado neste Guia, porque não existe tal processo. Padrões como IEEE12207 falam sobre o processo de engenharia de software, de maneira que existem muitos processos envolvidos, tais como processo de desenvolvimento, ou gerencia de processo de configuração.

Uma segunda maneira refere-se à discussão geral do processo para engenharia de software relatado. Esta é a maneira entendida nesta KA, e sua maior intenção.

Finalmente, a terceira maneira poderia significar a atual forma das atividades realizadas dentro das organizações, a qual pode ser visualizado como um processo, especialmente dentro de uma organização. Esta interpretação é usada no KA em poucos exemplos.

A KA se aplica a qualquer parte do gerenciamento do processo de ciclo de vida do software onde mudanças de processos ou de tecnologia são inicialmente introduzidas através da melhoria de processos ou produtos.

Processo de engenharia de software é relevante não somente para grandes organizações. Pelo contrário, as atividades relacionadas com o processo podem, e tem sido executadas com sucesso por pequenas organizações, equipes e indivíduos.

O objetivo do gerenciamento do ciclo de vida do processo de software é implementar um novo ou melhores processos nas atuais práticas, sejam elas individuais, de projeto ou organizacional.

Esta KA não especifica o gerenciamento de recursos humanos (HRM), por exemplo, como os baseados em pessoas CMM (Cur02) e processo de engenharia de sistemas [ISO1528-028;IEEE 1220-981].

Também se deve reconhecer que muitos problemas de processo de engenharia de software estão intimamente relacionados com outras disciplinas, tais como o gerenciamento, embora tecnologias diferentes sejam usadas.

2.1.3.8.1 Processo de implementação e mudança

Este subárea concentra-se na mudança organizacional. Ela descreve a infraestrutura, atividades, modelos, e considerações práticas para o processo de implementação e mudança.

É descrita aqui a situação em que os processos são implantados pela primeira vez (por exemplo, introdução de um processo de inspeção em um projeto ou um método que abrange todo o ciclo de vida), e onde os processos atuais estão mudando (por exemplo, introdução de uma ferramenta, ou otimização de um procedimento). Isto também pode ser denominado processo de evolução. Em ambos os casos, existe práticas que devem ser modificadas. Se as modificações são grandes, então mudanças na cultura da organização podem ser necessárias.

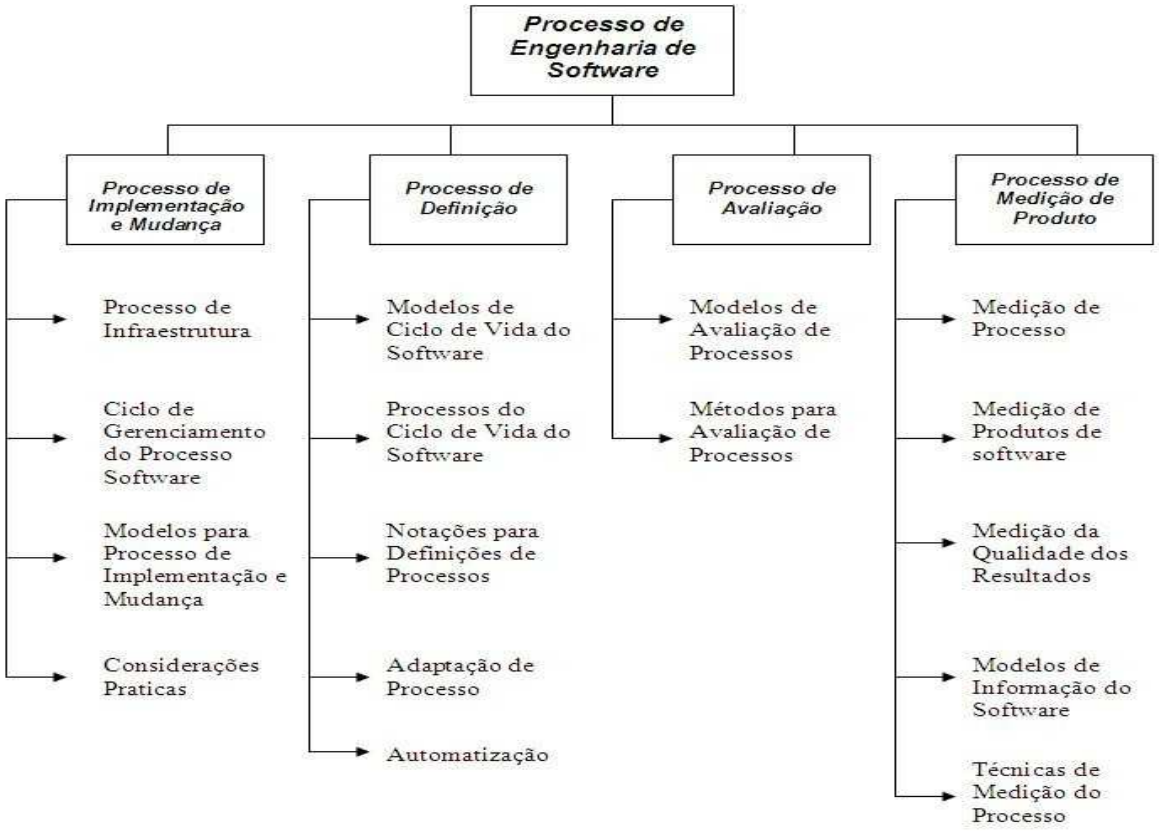


Figura1 - Distribuição dos temas do Processo KA para Engenharia de Software

Figura 23: Tópicos para a KA Processo de Engenharia de Software

a) Processo de Infraestrutura

Este tópico inclui o conhecimento relacionado para o processo de engenharia de infraestrutura de software.

Para estabelecer o ciclo de vida do processo de software, é necessário ter um

local com infraestrutura apropriada, o que significa que os recursos devem estar disponíveis (pessoal competente, ferramentas e recursos financeiros) e responsabilidades definidas. Quando as tarefas forem completadas, será a indicação de desempenho do gerenciamento para a apropriação, e desempenho do processo de engenharia de software. Varias diretorias poder ter que vir a ser criadas, tais como diretoria de supervisão do desempenho do processo de engenharia de software.

Uma descrição geral de uma infraestrutura para o processo de melhoria é mostrada no [MCF96]. Dois principais tipos de infraestrutura são usados na prática: o Grupo de Engenharia de Processo de Software e a Fábrica de Experiência.

- Grupo de Engenharia de Processo de Software (SEPG). O SEPG é destinado a ser o foco central da engenharia de melhoria de software, e ela tem uma série de responsabilidades em para iniciar e manter essa posição, os quais são descritas em [Fow90].
- Experiência Fábrica (EF). O conceito de EF separa o projeto organizacional (por exemplo, a organização de desenvolvimento de software,) da área de melhoria. A área de projeto organizacional se preocupa com o desenvolvimento e manutenção de software, enquanto EF se preocupa com a melhoria da engenharia do processo de software.

A EF é destinada a institucionalizar o aprendizado coletivo de uma área pelo desenvolvimento, atualizando, e entregando à área de projetos pacotes experientes (por exemplo, guias, modelos e cursos de treinamento), também referidos como processos ativos. A organização do projeto Exemplos de pacotes experientes são apresentados in [Bas92].

b) Gerenciamento do ciclo do Processo de software

O gerenciamento de processo de software consiste de quatro atividades sequenciadas o que permitindo um ciclo iterativo para um contínuo feedback e melhoria do processo de software:

- Atividade de Estabelecer o Processo Infraestrutura consiste em estabelecer compromisso para o processo de implementação e mudança (incluindo obtenção management buy-in) e na criação de uma infraestrutura adequada (recursos e responsabilidades) para as atividades.

- A meta da atividade Planejamento é entender os objetivos do negócio corrente e necessidades do indivíduo, projeto, ou organização, para identificar seus pontos fortes e fracos, e fazer um plano para processo de implementação e mudança.
- A meta da atividade de Processo de Implementação e Mudança é executar o plano, implantar os novos processos (os quais podem envolver, por exemplo, implantação de ferramentas e treinamento do grupo), e/ou mudar os processos existentes.
- Processo Avaliação se preocupa em descobrir o quão bem a implementação das mudanças atenderam ou não as expectativas e se os benefícios esperados foram alcançados. Os resultados são então usados como entrada para ciclos sequentes.

c) Modelos Para Processos de Implementação e Mudança

Dois modelos gerais que tem emergido para orientar processos de implementação e mudança são o Paradigma de Melhoria de Qualidade (QIP) [SEL96] e o Modelo IDEAL [McF96]. Os dois paradigmas são comparados em [SEL96]. Os Resultados da avaliação do processo de implementação e mudança podem ser qualitativos ou quantitativos.

d) Considerações Práticas

Processo de implementação e mudança constitui um exemplo de mudança organizacional. A mudança organizacional mais bem sucedia trata a mudança como um projeto próprio, com planos apropriados, monitoramento, e revisões.

A Orientações sobre o processo de implementação e mudança da engenharia de software dentro de organizações, inclui ações de planejamento, formação, gestão de patrocínio, desempenho, e a seleção de projetos piloto, que abrangem tanto processos como ferramentas, são apresentados em [Moi98; San98; Sti99]. Estudos empíricos sobre fatores de sucesso para processos de mudança são relatados em (EIE99a).

O papel dos agentes de mudança nesta atividade é discutido em (Hut94). O processo de implementação e mudança também pode ser visto como uma instância de consulta (seja interna ou externa).

Uma mudança organizacional também pode ser vista a partir da perspectiva da transferência de tecnologia (Rog83). Artigos de engenharia de software que discutem a transferência de tecnologia e as características dos destinatários da nova tecnologia (que pode incluir tecnologias relacionadas com o processo) estão em (Pfl99; Rag89).

Existem duas formas de abordar a avaliação do processo de implementação e mudança, quer em termos de alterações do próprio processo ou em termos de alterações do processo de resultados (por exemplo, medir o retorno sobre o investimento para realizar a alteração). Um olhar pragmático para o que pode ser alcançado a partir desses estudos de avaliação é dado em (Her98).

Um panorama das formas de avaliar processo de implementação e mudança, e os exemplos de estudos, pode ser encontrado em [Gol99], 9kit98; Kra99; McG94.

2.1.3.8.2 Definição de Processo

A definição de processo pode ser um procedimento, uma política, ou uma norma. Processos de ciclo de vida de software são definidos por uma série de razões, incluindo o incremento da qualidade do produto, melhorias da compreensão humana e comunicação, apoio ao processo de melhoria, apoio aos processos de gestão, orientação aos processos automatizados, e providenciando a execução de suporte automatizado. Os tipos de definições exigidas no processo dependerão, pelo menos parcialmente, do motivo da definição.

Também é importa notar que o contexto do projeto e da organização irão determinar a definição do tipo de processo que é mais útil. Variáveis importantes a considerar incluem a natureza do trabalho (por exemplo, a manutenção ou desenvolvimento), o domínio da aplicação, o modelo do ciclo de vida, e da maturidade da organização.

a) Modelos de Ciclo de Vida de Software

Os modelos de Ciclo de vida de software servem como um alto nível de definição das fases que ocorrem durante o desenvolvimento. Eles não são destinados a fornecer definições pormenorizadas, mas em destacar as principais atividades e suas interdependências. Exemplos de modelos de ciclo de vida software são os modelo cascata, o modelo prototipagem descartável, desenvolvimento evolucionário, entrega incremental/iterativo, o modelo espiral, o modelo de software

reutilizável, e uma síntese automatizado de software. A comparação destes modelos é fornecidos em [Com97], (Dav88), e um método de seleção dentre eles em (Ale91).

b) Ciclo de vida do processo de software

Definições do ciclo de vida de processos software tendem a ser mais detalhados do que os modelos ciclo de vida de software. No entanto, processos de ciclo de vida de software não devem tentar ordenar o seu processo em tempo útil. Isto significa que, em princípio, o ciclo de vida do processo software pode ser disposto para caber em qualquer dos modelos de ciclo de vida do software. A principal referência nesta área é IEEE / EIA 12207,0: Tecnologia da Informação – Software Life Cycle Processes [IEEE 12207.0-96].

O padrão IEEE 1074:1997 de desenvolvimento de processos de ciclo de vida também fornece uma lista de processos e atividades de desenvolvimento e manutenção de software [IEEE1074-97], bem como uma lista de atividades do ciclo de vida que possam ser mapeadas em processos e organizadas no mesmo de modo que todos os modelos de ciclo de vida do software. Além disso, identifica outros padrões de software ligados ao IEEE para estas atividades. Em princípio, IEEE Std 1074 pode ser usada para construir processos em conformidade com qualquer dos modelos de ciclo de vida. Normas que se concentram nos processos de manutenção são IEEE Std 1219-1998 e ISO 14764: 1998 [IEEE 1219-98].

Outros padrões importantes que fornecem a definições de processo incluem:

- IEEE Std 1540: Software Risk Management (IEEE1540-01)
- IEEE Std 1517: Software Reuse Processes (IEEE 1517-99)
- ISO/IEC 15939: Software Measurement Process [ISO15939-02]. Veja também o gerenciamento de engenharia de software KA para a descrição detalhada deste processo.

Em algumas situações, processos de engenharia de software devem ser definidos tendo em conta os processos organizacionais de gestão da qualidade. ISO 9001 [ISO9001-00] provem requisitos para gerenciamento de processos da qualidade, e ISO/IEC 90003 que interpretam esses requisitos para as organizações de desenvolvimento de software (ISO90003-04).

Alguns ciclos de vida de softwares enfatizam processos de entrega rápida e forte participação dos usuários, chamados de métodos ágeis, como Extreme

Programming (Bec99]. Uma forma do problema da seleção diz respeito à escolha, ao longo do plano de direção do método base Uma abordagem de risco de como tomar essa decisão é descrito em (Boe03a).

c) Notações pra definição de processos

Este processo pode ser definido em diferentes níveis de abstração (por exemplo, definições genéricas vs definições adaptadas, descritiva vs prescritivo vs proscritiva) [Pfl01].

Vários elementos de um processo podem ser definidos, por exemplo, atividades, produtos (artefatos), e os recursos. Quadros detalhados que estruturam os tipos de informações necessárias para definir os processos são descritos em (Mad94).

Há uma série de anotações sendo utilizadas para definir processos (SPC92). A grande diferença entre eles está no tipo de informações dos quadros mencionados acima definem, capturam e usam. O engenheiro de software deve estar ciente das seguintes abordagens: diagramas de fluxo de dados, em termos de processo objetivo e os resultados [ISO15504-98], como uma lista dos processos constituintes decomposto em atividades e tarefas definidas na linguagem natural [IEEE12207.0-96] , Esfatechar (Har98), ETVX (Rad85), modelagem Ator Dependência (Yu94), SADT notação (Mcg93), redes de Petri (Ban95); IDEFO (IEEE 1320.1-98), e a regra de base (Bar95). Mais recentemente, um padrão de modelagem de processo foi publicado pela OMG, que visa harmonizar notações de modelagem. Isto é denominado o SPEM (Software Engineering Process Meta-Model) especificação. [OMG02].

d) Processo de Adaptação

É importante notar que, mesmo processos predefinidos mais usados devem ser adaptados às necessidades locais, por exemplo, um contexto organizacional, a dimensão dos projetos, requisitos regulamentar, práticas da indústria, e cultura das organizações. Alguns padrões, como o IEEE / EIA 12207, contêm mecanismos e recomendações para a realização das adaptações.

e) Automação

Cada ferramenta automatizadas da suporte a execução da definições do processo ou fornecem orientações para os indivíduos exercerem processos definidos. Nos casos onde processo de análise é realizado, algumas ferramentas permitem diferentes tipos de simulações (por exemplo, a simulação eventos discretos).

Além disso, há ferramentas que suportam cada um dos referidos processo de definição notações. Essas ferramentas podem executar o processo automatizado de fornecer suporte as definições dos processos reais, ou automatizar totalmente em alguns casos. Uma visão geral das ferramentas de modelagem de processos pode ser encontrada em [Fin94] e do centro de processo de ambientação em (Gar96). Trabalho, sobre a aplicação da Internet para a provisão de orientação do processo em tempo real está descrito em (Kel98).

2.1.3.8.3 Avaliação de Processo

Processo de avaliação é realizada utilizando tanto uma modelo de avaliação ou um método de avaliação. Em alguns casos, o termo "avaliação" é usado no lugar de avaliação, bem como o prazo "Capacidade de avaliação" é utilizada quando a avaliação é para os efeitos da adjudicação de um contrato.

a) Modelos de Avaliação de Processos

O modelo de avaliação de processos utiliza o que é reconhecido como boas práticas. Essas práticas podem dizer respeito somente a atividades de técnicas de engenharia software, ou podem também referir-se, por exemplo, gestão, engenharia de sistemas, e humanos bem como as atividade de gerenciamento dos recursos.

ISO / IEC 15504 [ISO15504-98] define um modelo de avaliação exemplo e sobre outras exigências conforme modelo de avaliação. Modelos específicos de avaliação disponível e em uso são SW-CMM (SEI95), CMMI [SEI01], e Bootstrap [Sti99]. Capacidade de maturidade e de muitos outros modelos têm sido definidas, por exemplo, para a concepção, documentação, e métodos formais, para citar apenas alguns. ISO 9001 é outro modelo de avaliação comum, que tem sido aplicado por organizações de software (ISO9001-00).

Um modelo de maturidade para a engenharia de sistemas também tem sido desenvolvidos, o que seria mais útil quando um projeto ou organização está envolvida no desenvolvimento e manutenção de sistemas, incluindo software (EIA/IS731- 99).

A aplicabilidade dos modelos para avaliação pequenas organizações é abordado em [Joh99; San98]. Existem duas arquiteturas para uma avaliação geral modelo que faz diferentes suposições sobre a ordem em que processos devem ser avaliados: contínua e por fases (Pau94). Eles são muito diferentes, e devem ser avaliadas pela organização considerando-as para determinar qual seriam as mais apropriada para as suas necessidades e objetivos.

b) Métodos de avaliação de processos

A fim de realizar uma avaliação, um método avaliação específica deve ser seguido para produzir uma Pontuação quantitativa que caracteriza a potencialidade do processo (ou maturidade da organização).

O método de avaliação CBA-IPI, por exemplo, concentra-se em processo de melhoria (Dun96), o método SCE centra-se em avaliar as capacidades dos fornecedores (Bar95). Ambos foram desenvolvidos pelo SW-CMM. Ambos os tipos de métodos de requisitos refletem o que se acredita ser boas práticas de avaliação estão previstas em [ISO15504-98], (Mas95). Os métodos scampi são orientadas pela avaliações CMMI [SEI01]. As atividades realizadas durante uma avaliação, são a distribuição do esforço sobre estas atividades, bem como a atmosfera durante um avaliação são diferentes quando são para a melhoria do que para uma junção.

Tem havido críticas aos modelos de avaliação de processo e métodos, por exemplo (Fay97; Gra98). A maioria dessas críticas têm se preocupado com as evidências empíricas que dão suporte à utilização de modelos e métodos de avaliação. No entanto, desde a publicação desses artigos, verificaram-se de forma sistemática alguns elementos que sustentavam a eficácia dos processos de avaliação. (Cla97; Ele00; Ele00a; Kri99)

2.1.3.8.4 Processo e Medição de Produto

Embora a aplicação de medição da engenharia de software pode ser complexa, particularmente em termos de modelagem e análise de métodos, existem

vários aspectos de engenharia de medição de software, que são fundamentais e que são a base de muitos dos mais avançados processos de medição e análise. Além disso, a realização do processo e os esforços para a melhoria dos produtos só pode ser avaliado se um conjunto de medidas de base tiver sido estabelecido.
A medição pode ser executada para apoiar a iniciação ou para avaliar as consequências da implementação de processo. E também, esta medição pode ser executa no próprio produto.

Conceitos fundamentais sobre medidas de software e métodos de medição foram definidos na norma ISO / IEC 15939, com base nas ISO vocabulário internacional de metrologia. ISO / IEC 15359 também fornece um processo padrão de medir tanto os processos quanto as características de produto. [VIM93]
No entanto, os leitores vão encontrar termos divergentes na literatura, por exemplo, o termo "métrica" às vezes é usado no lugar de "medir".

a) Processo de Medição

O termo "processo de medição", conforme usado aqui significa que informações quantitativas sobre o processo são recolhidas, analisadas, e interpretadas. A medição é utilizada para identificar os pontos fortes e fracos dos processos e para avaliá-los depois que eles foram implementados e/ou modificados.
O processo de medição pode servir a outros objetivos também. Por exemplo, é útil para gerenciar um projeto de engenharia de software. Aqui, o foco está no processo de medição e o objetivo na implementação e modificação de processo.

O contexto afeta o relacionamento entre os processos e os resultados de processos. Isto significa que esta relação processo a processo, depende do contexto do relacionamento.

Nem todo processo terá um impacto positivo em todos os resultados. Por exemplo, a introdução de software pode reduzir os ensaios de esforço e custo, mas pode aumentar o tempo decorrido, se em cada inspeção introduzir atrasos devido à programação de inspeção de grandes reuniões. (Vot93) Portanto, é preferível utilizar múltiplos resultados de processos que são importantes para o negócio da organização.

Embora algum esforço pode ser feito para avaliar o aproveitamento das ferramentas, o principal recurso que deve ser geridos em engenharia de software é o

pessoal. Contudo, as principais medidas de interesse são aquelas relacionadas à produtividade de equipes ou de processos (por exemplo, usando a medida de pontos de função produzidos por unidade de personeffort) e seus respectivos níveis de experiência em engenharia de software em geral e também nas tecnologias. Os resultados de processo, por exemplo, podem ser qualidade do produto (faltas por KLOC (Kilo Lines of Code) ou por (PF) Function Point), manutenibilidade (o esforço para fazer um certo tipo de modificação), produtividade (LOC (Lines of Code) ou Pontos de Função por mês de pessoa time-to-market, ou satisfação de cliente (medido através de um cliente). Esta relação depende do contexto particular (por exemplo, o tamanho da organização ou o tamanho do projeto).

Em geral, estamos mais preocupados com os resultados do processo. No entanto, para alcançar os resultados de processo que nós desejamos (por exemplo, uma melhor qualidade, maior durabilidade, uma maior satisfação dos clientes), temos de aplicar os processos adequados.

Evidentemente, não é apenas o processo que tem um impacto sobre os resultados. Outros fatores, tais como a capacidade do pessoal e as ferramentas que são utilizadas, desempenham um papel importante. Quando avaliar o impacto de um processo de mudança, por exemplo, é importante observar outras influências. Além disso, à medida em que o processo for institucionalizado (isto é, processo de fidelidade) é importante, pois isso pode explicar porque é "bom" processos não darem sempre os resultados esperados em um determinada situação.

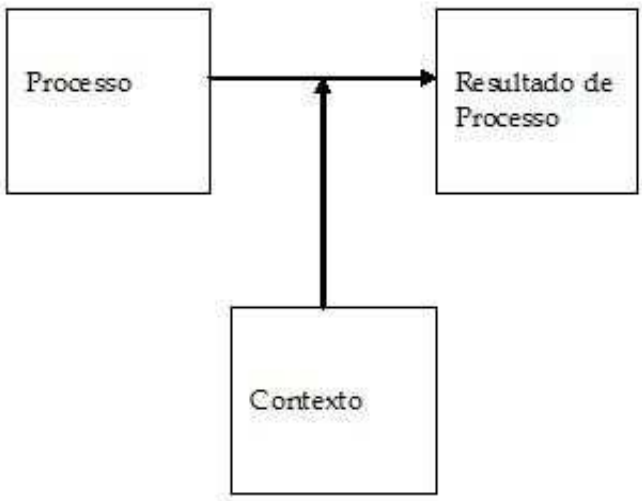


Figura 24: Relação entre processos e resultados

Medição de tamanho. O tamanho do produto de software é mais frequentemente avaliado por medidas de tamanho (por exemplo, linhas de código fonte em um módulo, páginas em um documento de especificação de requisitos de software), ou funcionalidade (por exemplo, pontos em uma função específica). Os princípios do tamanho da funcional , são fornecidos em IEEE Std 14143,1. Padrões internacionais para medir tamanho que incluem os métodos da funcional ISO / IEC 19761, 20926, 20968 e [IEEE 14143.1-00; ISO19761-03; ISO20926-03; ISO20968-02].

Medição de estrutura. Uma variada gama de medidas de estrutura de produtos de software pode ser aplicada para alto e baixo nível de design e artefatos de código para refletir no fluxo de controle (por exemplo o cyclomatic, número, nós de código), fluxo de dados (por exemplo, medidas de corte), otimização (por exemplo, a medida polinomial, a medida BAND), o controlo das estruturas (por exemplo, a medida vetorial, a medida NPATH), estrutura modular e interação (por exemplo, informações de fluxo, com base em medidas de árvore, acoplamento e coesão). [Fen98: C8; Pre04: C15].

Medição de qualidade. Como um atributo multidimensional, medição da qualidade é mais difícil de definir do que as anteriores. Além disso, algumas das dimensões da qualidade exigem a medição em termos qualitativos, e não de forma quantitativa. Uma discussão mais detalhada de mensuração do software é fornecida em Qualidade do Software KA, tema 3.4. ISO modelos de qualidade de software e medições relacionadas estão descritas na norma ISO 9126, partes 1 de 4 [ISO9126-01]. [Fen98: c9,c10; Pre04: c15; Som05: c24].

b)Qualidade dos resultados das medições

A qualidade dos resultados da medição (precisão, reprodução, repetição, conversão, erros de medição) é essencial para a medição de programas para proporcionar resultados eficazes e delimitados. As características chaves dos instrumentos de medição para garantir resultados de medição e qualidade foram definidas na norma ISO Vocabulário internacional sobre metrologia. [VIM93].

A teoria de medição estabelece a base sobre a forma que as medições podem ser feitas. Esta teoria e tipos de escala são discutidas [em Kan02]. A medição é definida

na teoria como "a atribuição de números para os objetos de uma forma sistemática de representar propriedades do objeto. "

Uma avaliação da medição de software e as implicações de cada tipo de escala em relação à sua seleção posterior de métodos de análise de dados é especialmente importante. [Abr96; Fen98: c2; Pfl01: C11] Escalas significativas são relacionada em uma classificação de escalas. A teoria da medição fornece cada vez mais formas de como realizar as medidas. Se os números atribuídos são apenas para fornecer etiquetas para classificar os objetos, eles são denominados como nominal. Se eles forem atribuídos de uma forma que classifica os objetos (por exemplo, bom, melhor), eles são chamados como ordinais. Se eles lidam com as magnitudes de propriedade em relação a uma unidade de medição definida, eles são chamados como intervalo (e os intervalos são uniformes entre os números, a menos que de outra maneira não estão especificado, e são, portanto aditivo).

As medições estão em nível de proporção se tiverem um nível absoluto ponto zero, portanto níveis de distâncias para o ponto zero são significativos.

c) Modelos de Informação sobre Software

Como os dados são reunidos e o repositório de medição é preenchido, nos ajudam a construir modelos de dados utilizando o conhecimento e os dados no repositório. Estes modelos existentes para fins de análise, classificação e previsão. Tais modelos têm de ser avaliados para assegurar que os seus níveis de precisão são suficientes e que as suas limitações são conhecidas e entendidas. O refinamento de modelos, que se realiza durante e após os projetos são concluídos, esta é outra atividade importante.

Modelo de construção. Modelo de construção inclui ajustes e avaliação do modelo. A meta abordagem é orientada para a medição do modelo de processo de construção, na medida em que modelos são construídos para responder a perguntas pertinentes e atingir objetivos na melhoria de software. Este processo também é influenciado pelas limitações implícitas de medição em escalas com relação à escolha de métodos de análise . Os modelos são ajustados (usando particularmente observações pertinentes, por exemplo, os projetos recentes utilizam a mesma tecnologia) e sua eficácia é avaliada (por exemplo, pelo seu desempenho em testes e amostras). [Fen98: C4, C6, C13; Pfl01: c3, C11, C12; Som05: C25]

Modelo de implementação. Modelo de implementação inclui interpretação e refinamento de modelos. Os modelos ajustados são aplicados no processo, os seus resultados são interpretados e avaliados no contexto do processo / projeto, e os modelos são então refinados, quando necessário. [Fen98: c6; Pfl01: c3, C11, C12; Pre04: C22; Som05: C25]

d)Técnicas de Medição de Processo

Técnicas de medição podem ser utilizadas para a análise da engenharia de processos de software e para identificar pontos fortes e fracos. Isto pode ser realizado para iniciar a implementação e mudança do processo, ou para avaliar as consequências desta implementação e mudança.

A qualidade dos resultados de medição, como a precisão, repetição e reprodutibilidade, são as questões que não são usadas medidas de processos de engenharia de software, são instrumentos baseados em medições como, por exemplo, quando os assessores destinam uma grande quantidade para um determinado processo. Uma discussão sobre métodos para atingir qualidade de medição são apresentados em [Gol99].

As técnicas de medição de processo foram classificadas em dois tipos gerais: analítica e análise comparativa. Os dois tipos de técnicas podem ser usados em conjunto, porém, são baseado em tipos diferentes de informação. (Car91)

Técnicas analíticas. As técnicas analíticas são caracterizadas como: "elementos quantitativos para determinar onde melhorias são necessárias e se uma iniciativa de melhora foi bem-sucedida". O tipo analítico é exemplificado pela Quality Improvement Paradigm (QIP) constituído por um ciclo da compreensão, da avaliação, e de embalagens [SEL96]. As técnicas apresentadas a seguir, se destinam a outros exemplos de técnicas analíticas, e refletem o que é feito na prática. [Fen98; Mus99], (Lyu96; Wei93; Zel98). Organizações específicas usam estas técnicas, pelo menos parcialmente, na sua maturidade.

- Estudos experimentais: Experimentação envolve criação de experimentos controlados na organização para avaliar processos. (McG94) Geralmente, um novo processo é comparado com o atual e determina se o antigo tem um melhor andamento.
- Outro tipo de estudo experimental é o processo de simulação. Este tipo de

estudo pode ser utilizado para a análise do processo de comportamento, explorar a melhoria de processos, predizer se o atual processo foi alterado e controlar os processos em execução. Dados iniciais sobre o desempenho do atual processo necessitam ser recolhidos, contudo, como uma base para a simulação.

- Definição de Revisão de Processos: é um meio pelo qual um processo de definição (descritivo, prescritivo ou ambos) é revisto, e as eficiências e melhorias do processo são identificadas. Os exemplos típicos disto são apresentados em (Ban95; Kel98). Uma maneira fácil para analisar um processo é compará-lo com um padrão existente (nacional, internacional, ou um organismo profissional), tais como IEEE / EIA 12207,0 [IEEE12207.0-96]. Com esta abordagem, dados quantitativos não são reunidos no processo, caso sejam, desempenham um papel de suporte. Os indivíduos realizando a análise da definição do processo usam seus conhecimentos e capacidades para decidir sobre as mudanças necessárias para o resultado do processo. Estudos de observação também podem fornecer feedback útil para a identificação de melhorias no processo. (Agr99)
- Classificação de Defeito Ortogonal: é uma técnica que pode ser utilizada para interligar faltas encontradas com causas potenciais. Baseia-se em um mapeamento entre os tipos de falhas e erros. (Chi92; Chi96) A norma IEEE sobre a classificação de falhas (ou anomalias) pode ser útil neste contexto (IEEE Standard da classificação das Anomalias Software (IEEE1044-93).
- Controle de Estatística do Processo: é um meio eficaz para identificar a estabilidade, ou a falta dela, através da utilização de gráficos e controle das interpretações. Uma boa introdução para o SPC em contexto de engenharia de software é apresentado em (Flo99).
- Processo de Software Pessoal: define uma série de melhorias para o desenvolvimento de práticas em uma determinada ordem [Hum95]. É "bottom-up" no sentido em que prevê a recolha de dados pessoais e melhorias baseadas em interpretações dos dados.

Técnicas de teste de desempenho de um sistema. O segundo tipo de teste de desempenho de um sistema, "depende da identificação de uma organização 'excelente' em um campo e documentação das suas práticas e instrumentos.”Teste

de desempenho de um sistema assume isto: se a organização proficiente adotar as práticas da organização excelente, esta também ira se tornar excelente. Teste de desempenho de um sistema implica avaliar a maturidade de uma organização ou a capacidade de seus processos. Ele é exemplificado pelo processo de avaliação de software. Uma visão geral do processo de avaliação e sua aplicação está prevista no (Zah98).

2.1.3.9 Ferramentas e Métodos da Engenharia de Software

Ferramentas de desenvolvimento de software são as que se destinam a ajudar nos processos de ciclo de vida do software. As ferramentas permitem, repetir e definir ações automatizadas, reduzindo a carga cognitiva sobre a engenharia de software, deixando-o livre para concentrar-se na criatividade e aspectos do processo. As ferramentas são muitas vezes concebidas para apoiar em particular os métodos da engenharia de software, reduzindo qualquer carga administrativa associada a métodos aplicados manualmente. Como os métodos de engenharia são destinados a tornar a engenharia de software mais sistemática, eles variam o escopo para suportar tarefas individuais englobando um ciclo de vida completo.

Os métodos da engenharia de software impõem na sua estrutura metas de fabricação e atividades sistemáticas com a finalidade de se ter uma maior probabilidade de sucesso. Os métodos normalmente fornecem notações e vocabulários, para o procedimento de tarefas identificáveis e orientações para verificar o processo e o produto. Eles variam amplamente no escopo para uma simples e completa fase do ciclo de vida. A ênfase nesta KA (Knowledge Area - Área de Conhecimento), esta nos métodos da engenharia de software englobando várias fases do ciclo de vida, desde a fase de especificação até os métodos cobertos por outras KAs.

Embora existam manuais detalhados, ferramentas específicas e numerosos trabalhos de investigações em relação a ferramentas inovadoras, técnicas genéricas, de escrita de ferramentas de software são relativamente escassas. Uma das dificuldades é a elevada taxa de mudança nas ferramentas de software em geral. Detalhes específicos são alterados regularmente.

A engenharia de software ferramentas e métodos KA cobre todos os processos de

um ciclo de vida, e portanto, é relacionada para toda KA neste guia.

Figura 25: Tópicos da KA Ferramentas e Métodos da Engenharia de Software

2.1.3.9.1 Ferramentas da Engenharia de Software

Os cinco primeiros tópicos nas subáreas das Ferramentas de Engenharia de Software, correspondem aos cinco primeiros KAs deste Guia (Requisitos de Software, Design de Software, Construção de Software, Teste de Software e Manutenção de Software). Os próximos quatro tópicos restantes correspondem as KAs (Gestão de Configuração de Software, Gestão de Engenharia de Software, Processos de Engenharia de Software e Qualidade do Software). Um outro tópico adicional dirige-se a diversas áreas, contanto que sejam semelhantes e abordadas como ferramenta de integração técnicas que são potencialmente aplicáveis a todas as classes de ferramentas.

a) Ferramentas de Requisitos de Software

As ferramentas para tratar os requisitos de softwares foram classificadas em duas categorias: ferramentas de modelagem e ferramentas de rastreabilidade.

- Ferramentas de requisitos de modelagem. Estas ferramentas são usadas para obtenção, análise, especificação e validação de requisitos de software.
- Ferramentas de requisitos de rastreabilidade. [Dor02] Estas ferramentas estão crescendo e se tornando cada vez mais importantes na medida que cresce a complexabilidade do software. Desde que sejam também relevantes em outros processos de ciclo de vida, eles são apresentados separadamente dos requisitos de ferramentas de modelagem.

b) Ferramentas de Design de Software

Este tópico refere-se às ferramentas para criação e verificação de projetos de softwares. Existe uma variedade destas ferramentas, consequentemente uma diversidade de notações e métodos de projeto de softwares. Apesar de existir toda esta variedade, não foi encontrado divisões para este tópico.

c) Ferramentas de Construção de Software

Este tópico refere-se às ferramentas de construção de software. Estas ferramentas são usadas para produção e tradução da representação de um programa (por exemplo, código fonte) que é suficientemente detalhado e desenvolvido para permitir a execução na máquina.

- Editores de programas. Estas ferramentas são usadas para a criação e modificação de programas, e possivelmente os documentos associados a eles. Eles podem ser editores de textos ou documentos de uso geral, ou podem ser editores específicos para uma determinada linguagem.
- Compiladores e Geradores de Código. Tradicionalmente os compiladores não eram tradutores interativos de código fonte, mas houve uma tendência em integrar compiladores e programas de edição, para se ter ambientes de programação integrados. Este tópico também referencia processadores, linker/loaders, e geradores de código.
- Interpretadores – estas ferramentas fornecem execução de software através de emulação. Elas podem apoiar as atividades de construção de softwares,

fornecendo um ambiente mais controlável e observável para a execução de programas.

- Depuradores. Estas ferramentas são consideradas uma categoria separada, desde que elas deem apoio aos processos de construção de software, mas elas são diferentes dos programas de edição e dos compiladores.

d) Ferramentas de Teste de Software

- Geradores de teste. Estas ferramentas auxiliam no desenvolvimento de casos de testes.
- Execução de testes por etapas. Estas ferramentas permitem a execução de casos de teste em um ambiente controlado onde o comportamento do objeto que está sendo testado é observado.
- Ferramentas de avaliação de teste. Estas ferramentas suportam a avaliação dos resultados dos testes de desempenho, ajudando a determinar se o comportamento observado está de acordo com o comportamento previsto.
- Ferramentas de Gerenciamento de Teste. Estas ferramentas fornecem suporte para todos os aspectos do processo de testes de software.
- Ferramentas de Análise e desempenho. [Rei96] Estas ferramentas são usadas para medir e analisar o desempenho do software, que é uma forma especializada de ensaios onde o objetivo é avaliar o comportamento da operação e não o comportamento funcional (correção).

e) Ferramentas de Manutenção de Softwares

Este tópico refere-se às ferramentas que são particularmente importantes na manutenção de software existentes e que necessitem de modificações. São identificadas duas categorias: ferramentas de compreensão e ferramentas de reengenharia.

- Ferramentas de compreensão. [Re196] Estas ferramentas ajudam na compreensão humana de programas. Os exemplos incluem ferramentas de visualização e partes animadas de programas.
- Ferramentas de reengenharia. Na KA manutenção de software é definido como a investigação e alteração do software, ficando-o sujeito a construção de uma nova forma, incluindo implementações subsequentes desta nova

forma. As ferramentas de reengenharia suportam essas atividades.

Ferramentas de engenharia reversa auxiliam o processo de trabalho a criar artefatos de um produto já existente, como especificação e descrições de desenhos, que podem então ser transformados para gerar um novo produto a partir de um antigo.

f) Ferramentas de Gerenciamento de Configuração de Software

As ferramentas para gerenciar configuração de Software foram divididas em três categorias: monitoramento, gerenciamento de versões e ferramentas de correção.

- Ferramentas para gerenciar defeito, aprimoramento, edição e monitoramento. Estas ferramentas são usadas na conexão com o monitoramento de problemas associados a um determinado produto de software.
- Ferramentas para gerenciar versões. Estas ferramentas estão envolvidas na gestão das várias versões de um produto.
- Ferramentas de correção. Estas ferramentas são usadas para obter uma tarefa de correção e desenvolvimento de software. A categoria inclui instalação de ferramentas que se tornaram amplamente utilizadas para configurar a instalação de produtos de software.

Informações adicionais são fornecidas em Configuração e Gerenciamento de Software KA, tópico 1.3 planejamento para SCM.

g) Ferramentas de gerenciamento de Engenharia de Software

Ferramentas de gerenciamento de engenharia de software são subdivididas em três categorias: planejamento e acompanhamento de projeto, gerenciamento de risco e medição.

- Ferramentas de planejamento e acompanhamento de projeto. Estas ferramentas de software são usadas para medir esforços e estimar custos no projeto, bem como a sincronização de projetos.
- Ferramentas de gerenciamento de riscos. Estas ferramentas são usadas na identificação, estimativa e monitoramento de riscos.
- Ferramentas de Medição. As ferramentas de medição auxiliam na performance de atividades relacionadas na mensuração dos programas.

h) Ferramentas para Métodos de Engenharia de Software

Ferramentas para métodos de engenharia de software são divididas em ferramentas de modelagem, ferramentas de gerenciamento e ambiente de desenvolvimento de software.

- Ferramentas para métodos de modelagem. [Pf101] Estas ferramentas são usadas para modelar e verificar o processo de engenharia de software.
- Ferramentas para métodos de gerenciamento. Estas ferramentas fornecem suporte para o gerenciar a engenharia de software.
- Ambiente CASE integrado. [Rei96, Som05] (ECMA55-93, ECMA69-94, IEEE1209-92, IEEE1348-95, Mul96) Engenharia de software integrada com ferramentas auxiliadas por computador ou ambientes que abrangem múltiplas fases do ciclo de vida da engenharia de software fazem parte deste sub tópico. Tais ferramentas desempenham múltiplas funções e, portanto, interagem potencialmente com o ciclo de vida a ser executado pelo processo de software.
- Ambiente centrado no processo de engenharia de software. [Rei96] (Gar96) estes ambientes incorporam explicitamente informações no processo do ciclo de vida do software e orientam o monitoramento do usuário de acordo com os processo definidos.

i) Ferramentas de Qualidade do Software

Ferramentas de qualidade são divididas em duas categorias: verificação e análise de ferramentas.

- Ferramentas de auditoria e análise. Estas ferramentas são usadas para auxiliar na análise e auditorias.
- Ferramentas de análise estática. [Cla96, Pfl01, Rei96] estas ferramentas são usadas para analisar artefatos de software, tais como análises sintática e semântica, bem como dados, controle de fluxo e análise de dependências. Essas ferramentas são destinadas para verificar artefatos de softwares de modo analisar conformidade ou para verificação das propriedades desejadas.

j) Ferramentas para Problemas Variados

Este tópico abrange assuntos aplicáveis para todas as classes de

ferramentas. Existem três categorias bem identificadas: ferramentas de integração técnica, meta ferramentas e ferramentas de evolução.

- Ferramentas de integração técnica. [Pfl01, Rei96, Som01] (Bro94) Ferramentas de integração são importantes para estruturar pessoas a ferramentas. Esta categoria potencialmente sobrepõe com a integração da categoria do ambiente CASE onde são aplicadas integrações técnicas; no entanto elas são suficientemente distintas para merecer sua própria categoria. São os modelos típicos de ferramentas de integração, plataformas, apresentação, processo, dados e controles.
- Metas ferramentas. Metas ferramentas geram outras ferramentas; um exemplo clássico é compilar um código / compiladores.
- Ferramentas de evolução [Pfl01] (IEEE1209-92, IEEE1348-95, Mos92, Val97) Por causa da contínua evolução da engenharia de software, ferramentas de evolução é um tópico essencial.

2.1.3.9.2 Métodos de Engenharia de Software

As sub áreas dos Métodos da Engenharia de Software são divididas em três tópicos: relacionamento de métodos heurísticos com abordagens informais, relacionamento de método formal com abordagens matematicamente baseadas e relacionamento de métodos de prototipagem com abordagens em engenharia de software baseada em diferentes formas de protótipo. Estes três tópicos não são separados em partes; particularmente eles representam interesses distintos. Por exemplo um objeto orientado a métodos pode incorporar técnicas formais e contar com um protótipo de verificação e validação. Tal como ferramentas de engenharia de software, são desenvolvidas tecnologias continuamente. Consequentemente a KA evita descrever na medida que possível nomear metodologias particulares.

a) Métodos Heurísticos

Este tópico contém quatro categorias: Estruturado, orientado a dados, orientado a objetos e domínio especifico. A categoria domínio específico inclui métodos especializados para desenvolvimento de sistemas o qual envolve tempo real, segurança ou aspectos seguros.

- Métodos estruturados [Dor02, Pfl01, Pre04, Som05] O sistema é embutido

para um ponto de vista funcional, iniciando com uma visão de alto nível e refinado progressivamente em um projeto mais detalhado.

- Métodos orientados a dados. [Dor02, Pre04] aqui o ponto inicial são os dados estruturados que um programa manipula melhor do que a função que executa.
- Métodos orientados a objeto. [Dor02, Pfl01, Pre04, Som05] O sistema é visualizado como uma coleção de objetos, em vez de funções.

b) Métodos Formais

Esta subseção é dividida matematicamente com métodos de engenharia de software, e esta subdividida com os vários aspectos de métodos formais.

- Especificação de linguagens e notações. [Cla96, Pfl01, Pre01] Este tópico foca a especificação de notação ou de linguagem usada. Linguagens de especificações podem ser classificadas como orientado a modelo, orientado por propriedade ou orientado por comportamento.
- Refinamento. [Pre04] Este tópico dá a forma de como o método é refinado (ou transformado) numa especificação de forma que seja mais próxima da forma final desejada de um programa executável.
- Propriedades de verificação/confirmação [Cla96, Pfl01,Som05] Este tópico trata as propriedades de verificação que são especificas as abordagens formais, incluindo tanto o modelo de teoria como o modelo de verificação.

c) Métodos de Prototipagem

Esta subseção abrange métodos que envolvem a prototipagem de software e é subdividida em estilos de prototipagem, metas e avaliações técnicas.

- Estilo de prototipagem. [Dor02, Pfl01, Pre04] (Pom96) O tópico estilo de prototipagem identifica as várias abordagens: rejeitada, evolutiva e especificação executável.
- Os objetivos de um método de prototipagem podem ser os requisitos de desenho de arquitetura, ou interface do usuário.
- Avaliações técnicas de prototipagem. Este tópico abrange as maneiras pelas quais os resultados de um treinamento de prototipagem são usados.

2.1.3.10 Qualidade de Software

O que é qualidade de software, e por que é tão importante que este assunto faça parte do Guia SWEBOK? Durante anos, autores e organizações têm definido "qualidade" de formas diferentes. Para Phil Crosby (Cro79), a melhor definição era "conformidade com os requisitos do usuário". Watts Humphrey (Hum89) refere-se ao termo como "alcançar excelentes níveis de adequação para uso", enquanto a IBM cunhou a frase "Qualidade dirigida pelo mercado" que está baseada em alcançar a satisfação total do cliente. O critério Baldrige para qualidade organizacional (NIST03) utiliza uma frase semelhante, "Qualidade dirigida pelo cliente". Mais recentemente, qualidade foi definida na (ISO9001-00) como "o grau com o qual um conjunto de características intrínsecas cumpre requisitos". Este capítulo trata das considerações da qualidade de software que transcendem os processos do ciclo de vida. Qualidade de software é uma questão onipresente na engenharia de software, e por isso é também considerada em muitas das KAs. Em resumo, o Guia SWEBOK descreve vários modos de alcançar qualidade de software.

Em particular, esta KA cobrirá técnicas estáticas, aquelas que não requerem a execução do software sendo avaliado, enquanto técnicas dinâmicas são cobertas na KA Teste de Software.

Figura 26: Tópicos para a KA Qualidade de Software

2.1.3.10.1 Fundamentos da Qualidade de Software

O entendimento dos requisitos de qualidade, bem como a clara comunicação com o engenheiro de software no que constitui qualidade, requer que vários aspectos sejam formalmente definidos e discutidos. Um engenheiro de software deve entender os significados básicos dos conceitos e características da qualidade e seus valores para o produto em desenvolvimento ou manutenção.

O conceito fundamental é o de que requisitos de software definem as características necessárias de qualidade e influenciam os métodos de medida e critérios de aceitação na avaliação de tais características.

a) Cultura e Ética da Engenharia de Software

Espera-se que os engenheiros de software compartilhem um compromisso com qualidade de software como parte de sua cultura. Uma robusta cultura de engenharia de software é descrita em [Wie96].

A ética pode desempenhar um papel significante na qualidade de software, na cultura e nas atitudes dos engenheiros de software. O IEEE Computer Society e a ACM [IEEE99] desenvolveram um código de ética e práticas profissionais baseada em oito princípios para ajudar os engenheiros de software a reforçar atitudes relacionadas à qualidade e à independência do seu trabalho.

b) Valor e Custo da Qualidade

A noção de "qualidade" não é tão simples quanto pode parecer. Para qualquer produto desenvolvido, existem várias qualidades desejadas relevantes para determinadas perspectivas do produto, que devem ser discutidas e determinadas durante a definição dos requisitos. Características de qualidade podem ser requeridas ou não, podem ser requeridas em maior ou menor grau, e ainda podem existir trade-offs entre elas. [Pfl01].

O custo da qualidade pode ser diferenciado em custo de prevenção, custo de avaliação, custo de fracasso interno e custo de fracasso externo. [Hou99]. A motivação por trás de um projeto de software é o desejo de se criar um software que tenha valor, e este valor pode ou não pode ser quantificado como um custo. O cliente terá algum custo máximo em mente, em troca do qual se espera que o propósito básico do software seja cumprido. O cliente também pode ter alguma

expectativa sobre a qualidade do software. Às vezes clientes podem não ter pensado nas questões de qualidade ou nos custos relacionados a ela. A característica é meramente decorativa ou é essencial ao software? Se a resposta se encontra em algum ponto entre as duas opções, como quase sempre é o caso, é importante que o cliente faça parte do processo de decisão, e esteja completamente a par dos custos e benefícios. O ideal é que a maioria destas decisões seja feita no processo de definição de requisitos do software (veja a KA Requisitos de Software), mas estas questões também podem surgir ao longo do ciclo de vida do software.

Não há nenhuma regra definida para como estas decisões devem ser tomadas, mas o engenheiro de software deve apresentar as alternativas de qualidade e os seus custos. Uma discussão relativa ao custo e ao valor dos requisitos de qualidade pode ser encontrada em Wei96:c11.

c) Modelos e Características da Qualidade

A terminologia para características de qualidade de software diferem entre os modelos, podendo haver diferenças entre o número de níveis de hierarquia e a quantidade total de características. Vários autores produziram modelos de características de qualidade de software ou atributos que podem ser úteis para discussão, planejamento e avaliação da qualidade dos produtos de software. [Boe78; McC77] A ISO/IEC definiu três modelos relacionados de qualidade de produtos de software (qualidade interna, qualidade externa e qualidade em uso) (ISO9126 -01) e um conjunto de partes relacionadas (ISO14598 -98).

O gerenciamento de qualidade de software e a qualidade dos processos de engenharia de software têm um grande impacto na qualidade final do produto.
Modelos e critérios que avaliam as capacidades das organizações de software são, principalmente, organização de projeto e considerações de gerenciamento, e, como tais, estão cobertos nas KAs Gerência de Engenharia de Software e Processo de Engenharia de Software.

Evidentemente, não é possível distinguir completamente a qualidade do processo da qualidade do produto.
Qualidade de Processo, discutida na KA Processo de Engenharia de Software deste Guia, influencia as características dos produtos de software percebidas pelo consumidor.

Dois importantes padrões de qualidade são o TickIT e o padrão ISO9001-00, junto com suas diretrizes para aplicação em software(ISO90003-04)

Outro padrão da industria de qualidade de software é o CMMI [SEI02], também discutido na KA Processo de Engenharia de Software. O CMMI tem o objetivo de fornecer orientações para melhorar processos. Áreas de processos específicos relacionados com gerência de qualidade são (a) garantia da qualidade do processo e do produto, (b) verificação de processo e (c) validação de processo. O CMMI classifica, revisa e audita como formas de verificação, e não como processos específicos como a norma (IEEE12207.0-96).

Inicialmente houve alguma discussão sobre qual dos dois padrões (ISO9001 ou CMMI) deveria ser usado por engenheiros de software para assegurar qualidade. Esta discussão esta amplamente publicada, e, como resultado, foi tomada a posição de que os dois são complementares e que ter a certificação ISO9001 pode contribuir consideravelmente para conseguir os níveis mais altos de maturidade do CMMI. [Dac01].

O engenheiro de software primeiramente precisa determinar qual o real propósito do software. Nesta consideração, é de extrema importância ter em mente que os requisitos do cliente vêm em primeiro lugar e que eles incluem requisitos de qualidade, não apenas requisitos funcionais. Assim, o engenheiro de software tem a responsabilidade de elicitar os requisitos de qualidade que geralmente não estão explícitos, e discutir sua importância bem como o nível de dificuldade em obtê-los. Todos os processos associados com a qualidade de software (por exemplo, construção, verificação e melhoria da qualidade) serão projetados com estes requisitos em mente, e isto leva a custos adicionais.

A norma (ISO9126-01) define, para dois de seus três modelos de qualidade, as características sub características de qualidade relacionadas, e medidas que são úteis para avaliar a qualidade de produtos de software. (Sur03).

O significado do termo "produto" é estendido para incluir qualquer artefato obtido como saída de qualquer processo usado para construir o produto final de software. Exemplos de um produto incluem, mas não são limitados a, uma especificação de requisitos de todo o sistema, uma especificação dos requisitos de software para um componente de um sistema, um módulo do projeto, código, documentação de teste ou relatórios produzidos como um resultado de tarefas de análise de qualidade.

Enquanto a maioria dos tratamentos de qualidade é descrita em termos do software final e do desempenho do sistema, boas práticas de engenharia exigem que avaliações em produtos intermediários relevantes para a qualidade sejam realizadas durante todo o processo de engenharia de software.

d) Melhoria da Qualidade

A qualidade de produtos de software pode ser aperfeiçoada através de um processo interativo de melhoria contínua que requer controle de gerenciamento, coordenação e avaliação de muitos processos concorrentes: (1) O ciclo de vida do processo de software, (2) o processo de detecção de defeito/erro, remoção e prevenção, e (3) A melhoria da qualidade do processo. (Kin92)

A teoria e os conceitos atrás da melhoria de qualidade, como qualidade em construção através da prevenção e descoberta antecipada de erros, melhoria contínua e foco no cliente, são pertinentes à engenharia de software. Estes conceitos estão baseados nos trabalhos de especialistas em qualidade que têm declarado que a qualidade de um produto está diretamente ligada à qualidade do processo utilizado para desenvolvê-lo. (Cro79, Dem86, Jur89)

Abordagens como a Gestão da Qualidade Total (TQM), e a metodologia Planejar, Executar, Verificar, Atuar (PDCA) são ferramentas para se atingir os objetivos de qualidade. O apoio gerencial provê avaliações de processos e produtos, e os resultados encontrados. Em seguida, um programa de melhoria é desenvolvido identificando ações detalhadas e melhoria nos projetos para serem abordadas em um prazo razoável. A gerência se certifica de que cada projeto de melhoria tenha o suficiente em recursos para alcançar a meta definida para o mesmo. O apoio gerencial frequentemente deve ser solicitado implementando atividades de comunicação preventivas. O envolvimento de grupos de trabalho, como também apoio de gerências intermediárias e alocação de recursos para níveis de projeto, são discutidos na KA Processo de Engenharia de Software.

2.1.3.10.2 Processos de Gestão da Qualidade de Software

A Gestão da Qualidade de Software (SQM) aplica-se a todas as perspectivas dos processos de software, produtos e recursos. Definem processos, donos e requisitos para os mesmos, medições do processo e de seus resultados, e canais de

avaliação. (Art93)

Os processos de gestão de qualidade de software consistem de muitas atividades. Algumas podem encontrar defeitos diretamente, enquanto outras indicam onde análises adicionais podem ser valiosas. Estas também são referenciadas como atividades de busca direta de defeito. Muitas atividades frequentemente se enquadram nos dois grupos.

Planejamento para qualidade de software envolve:

(1) Definir os requisitos do produto em termos de suas características de qualidade (descrito em mais detalhe na KA Gestão da Engenharia de Software).

(2) Planejar os processos para alcançar o produto requerido (descrito nas KAs Projeto de Software e Construção de Software).

Estes aspectos diferem, por exemplo, dos processos de planejamento SQM, que confrontam o planejamento de características de qualidade com a atual implementação desses planos. Os processos de gerência de qualidade de software devem abordar a forma como os produtos de software precisam satisfazer os requisitos do cliente e das partes interessadas (stakeholders), prover valor para os clientes e outras partes (stakeholders) e prover qualidade de software necessária para satisfazer os requisitos.

SQM pode ser usado para avaliar tanto os produtos intermediários quanto o produto final.

Alguns dos específicos processos SQM estão definidos no padrão (IEEE12207.0-96):

- Processo de garantia da qualidade
- Processo de verificação
- Processo de validação
- Processo de revisão
- Processo de auditoria

Estes processos incentivam a qualidade e também localizam possíveis problemas. Mas eles diferem um pouco na sua ênfase.

Processos SQM ajudam garantir melhor qualidade de software em um determinado projeto. Eles também fornecem, como um produto secundário, informações gerais para gerenciamento, inclusive uma indicação da qualidade de todo o processo de engenharia de software. As KAs Processo de Engenharia de

Software e Gestão da Engenharia de Software discutem programas de qualidade para as organizações de desenvolvimento de software. SQM pode fornecer feedback relevante para estas áreas.

Processos SQM consistem em tarefas e técnicas para indicar como os planejamentos de software (por exemplo, gerência, desenvolvimento, gestão de configuração) estão sendo implementados, e o quanto os produtos intermediários e finais estão de acordo com os requisitos especificados. Os resultados destas tarefas são reunidos em relatórios de gerência antes das ações corretivas serem tomadas. A gerência de um processo SQM é encarregada de garantir que os resultados destes relatórios sejam precisos.

Como descrito nesta KA, processos SQM estão intimamente ligados; eles podem se sobrepor e às vezes até mesmo se combinar. Eles parecem largamente reativos por natureza porque abordam os processos como praticados e os produtos como produzidos; mas eles têm um papel importante no inicio da fase de planejamento sendo preventivos em termos de processos e procedimentos necessários para atingir as características de qualidade e grau de qualidade necessários pelas partes interessadas (stakeholders) no software.

A Gerência de Risco também pode desempenhar um papel importante no cumprimento da qualidade de software. Incorporar análise disciplinada de riscos e técnicas de gestão nos processos do ciclo de vida do software pode aumentar o potencial para se desenvolver um produto de qualidade (Cha89). Recorra à KA Gestão de Engenharia de Software para material relacionado sobre gerência de risco.

a) Garantia da Qualidade de Software

Os Processos SQA garantem que os produtos de software e os processos no ciclo de vida do projeto estejam em conformidade com seus requisitos de especificação, pelo planejamento, ordenamento e execução de uma série de atividades para fornecer a confiança adequada de que a qualidade está sendo incorporada ao software. Isto significa assegurar que o problema está clara e adequadamente declarado e que os requisitos da solução estão corretamente definidos e expressados. SQA procura manter a qualidade ao longo do desenvolvimento e manutenção do produto pela execução de uma variedade de

atividades em cada fase que pode resultar na identificação precoce de problemas, uma característica quase inevitável de qualquer atividade complexa. O papel da SQA com respeito ao processo é assegurar que os processos planejados sejam apropriados e depois implementados de acordo com o planejamento, e que processos de medições pertinentes sejam fornecidos à organização apropriada.
O planejamento SQA define os meios que serão usados para assegurar que o desenvolvimento de software para um certo produto satisfaça os requisitos dos usuários e seja da melhor qualidade possível dentro das restrições do projeto. Para fazê-lo, em primeiro lugar é preciso garantir que os objetivos da qualidade estejam claramente definidos e compreendidos. É preciso considerar gerenciamento, desenvolvimento e planos de manutenção para o software. Para detalhes consulte a norma (IEEE730 -98).

As atividades e tarefas específicas de qualidade são definidas, com os seus custos e requerimentos de recursos, seus objetivos globais de gestão, e o seu cronograma em relação a esses objetivos na gestão da engenharia de software, desenvolvimento ou planos de manutenção. O planejamento SQA deve ser consistente com o plano de gestão de configuração do software (consulte a KA Gestão de Configuração de Software). O planejamento SQA identifica documentos, normas, práticas e convenções que regem o projeto e como eles serão conferidos e monitorados para garantir conformidade e adequação. O planejamento SQA também identifica métricas, técnicas estatísticas, procedimentos para reportar problemas e ações de correção, recursos como ferramentas, técnicas e metodologias, segurança para mídias físicas, treinamento, informação e documentação de SQA. Além disso, o planejamento SQA aborda as atividades de garantia da qualidade de software e qualquer outro tipo de atividade descritas no projeto de software, como obtenção de fornecedores para o projeto ou instalação do software de prateleira (commercial off-the-shelf software - COTS), e serviço após a entrega do software. Também podem existir critérios de aceitação bem como relatórios e gestão de atividades que são essenciais à qualidade do software.

b) Verificação & validação

Por questões de simplicidade, Verificação e Validação (V&V) são tratadas como um único tópico neste Guia ao invés de em tópicos separados como na norma

(IEEE12207.0-96). "Verificação e Validação de Software é uma abordagem disciplinada para avaliação dos produtos de software ao longo do ciclo de vida do produto. V&V se esforça para garantir que a qualidade seja incorporada ao software e que este satisfaça os requisitos do usuário" (IEEE1059-93).

V&V aborda diretamente a qualidade do produto de software e utiliza técnicas de teste para localizar defeitos que, em seguida, devem ser resolvidos. Eles também avaliam os produtos intermediários, e, neste caso, as etapas intermediárias dos processos do ciclo de vida.

O processo V&V determina se os produtos de uma certa atividade de desenvolvimento ou de manutenção estão em conformidade com os requisitos da atividade, e se o produto final de software atende à sua finalidade e satisfaz requisitos do usuário. Verificação é uma tentativa de garantir que o produto seja construído corretamente, no sentido que a atividade de desenvolvimento dos produtos reúne as especificações impostas em atividades anteriores. Validação é uma tentativa de se certificar que o produto certo foi construído, ou seja, que o produto específico cumpre sua função específica. Os processos de verificação e validação começam cedo no desenvolvimento ou fase de manutenção. Eles fornecem uma análise das principais funcionalidades do produto tanto em relação ao produto imediatamente antecessor quanto em relação a especificações às quais ele deve se ater.

O propósito do planejamento de V&V é garantir que cada recurso, papel, e responsabilidade sejam claramente assegurados. Os planos de V&V resultantes documentam e descrevem os vários recursos, seus papéis e atividades, além das técnicas e ferramentas a serem utilizadas. Uma compreensão dos propósitos diferentes de cada atividade de V&V ajudará no planejamento cuidadoso das técnicas e recursos necessários para cumprir seus propósitos. As normas (IEEE1012 -98:s7 e IEEE1059 -93: Apêndice A) especificam o que normalmente acontece em um plano de V&V.

O plano também contempla a gestão, comunicação, políticas e procedimentos de V&V e suas atividades de interação, bem como a elaboração de relatórios de defeitos e documentação de requisitos.

c) Revisões e Auditorias

Por questões de simplicidade, revisões e auditorias são tratadas como um único tópico neste Guia, ao invés de em dois tópicos separados como em (IEEE12207.0-96). Os processos de revisão e auditoria estão amplamente definidos em (IEEE12207.0-96) e em mais detalhe em (IEEE1028 -97). São apresentados cinco tipos de revisões ou auditorias na Norma IEEE1028-97:

- Revisões Gerenciais
- Revisões Técnicas
- Inspeção (Verificação)
- Walk-throughs
- Auditorias

"O propósito de uma revisão gerencial é monitorar o progresso, determinar o status de planos e cronogramas, confirmar requisitos e alocação de seus sistemas, ou avaliar a eficácia das abordagens de gerência utilizadas para atingir a adequação ao objetivo" [IEEE1028-97]. Tais abordagens apoiam decisões sobre mudanças e ações corretivas que são requeridas durante um projeto de software. Revisões gerenciais determinam a adequação dos planos, cronogramas e requisitos, e monitoram seu progresso ou inconsistências. Estas revisões podem ser executadas em produtos tais como relatórios de auditoria, relatórios de progresso, relatórios de V&V e muitos tipos de planos, inclusive gestão de risco, gestão de projeto, gestão de configuração de software, segurança de software e avaliação de risco, entre outros. Consulte as KAs Gerenciamento de Engenharia de Software e Gerenciamento de Configuração de Software para material relacionado.

"O propósito de uma revisão técnica é avaliar um produto de software para determinar se ele é adequado para o uso planejado. O objetivo é identificar inconsistências a partir de normas e especificações aprovadas. Os resultados deverão fornecer evidências confirmando (ou não) que o produto cumpre as especificações e adere aos padrões necessários, e que mudanças são controladas" (IEEE1028 - 97).

Devem ser estabelecidos papéis específicos em uma revisão técnica: um tomador de decisões, um líder de revisões, um relator, e uma equipe técnica para dar assistência às atividades de revisão. Uma revisão técnica requer que entradas obrigatórias estejam no lugar e em ordem para proceder:

- Declaração de objetivos

- Um específico produto de software
- O específico plano de gerenciamento de projeto
- A lista de assuntos associados com este produto
- O procedimento de revisão técnica

A equipe acompanha os procedimentos de revisão. Uma pessoa qualificada tecnicamente apresenta uma visão geral do produto, e a análise é realizada durante uma ou mais reuniões. A revisão técnica estará completa uma vez que forem finalizadas todas as atividades listadas na análise.

"O propósito de uma inspeção é descobrir e identificar anomalias nos produto de software" (IEEE1028 -97). Duas importantes diferenças entre inspeções e revisões são as seguintes:

- Um indivíduo que ocupe uma posição de gerenciamento sobre qualquer membro da equipe de inspeção não deve participar da inspeção.
- Uma inspeção deve ser conduzida por um facilitador imparcial treinado em técnicas de inspeção.

Inspeções de software sempre envolvem o autor de um produto intermediário ou final, enquanto outras revisões não o fazem. Inspeções também incluem um líder de inspeção, um relator, um leitor e alguns (2 a 5) inspetores. Os membros de uma equipe de inspeção podem possuir diferentes especializações, como especialização no domínio, especialização em métodos de design ou especialização em linguagens. As inspeções são geralmente conduzidas em uma parte relativamente pequena do produto a cada momento. Cada membro da equipe deve examinar o produto de software e outra revisão é aplicada antes da reunião de avaliação, talvez aplicando uma técnica analítica (consulte a seção 3.3.3) numa pequena seção do produto, ou no produto inteiro com foco em apenas um aspecto, por exemplo, interface.

Qualquer anomalia encontrada é documentada e enviada ao líder de inspeção. Durante o processo, o líder de inspeção conduz a sessão e verifica se todos os participantes estão preparados para a tarefa. Um checklist, com anomalias e questões pertinentes aos assuntos de interesse, é uma ferramenta comum usada em inspeções. A lista resultante frequentemente classifica as anomalias (consulte a IEEE1044-93 para detalhes) e a sua completude e correção são garantidas por uma revisão da equipe. A decisão de saída da inspeção tem que corresponder a um dos três critérios seguintes:

- Aceitar com nenhuma ou, no máximo, com pequenas modificações
- Aceitar com verificação de refactoring
- Inspecionar

Reuniões de inspeção duram, tipicamente, algumas horas, enquanto que revisões técnicas e auditorias, em geral, possuem um escopo mais abrangente e duram mais tempo.

"O propósito de um walk-through é avaliar um produto de software. Um walk-through pode ser realizado com a finalidade de educar uma audiência relacionada com o produto de software." (IEEE1028 -97) Os principais objetivos são [IEEE1028-97]:

- Encontrar anomalias
- Melhorar o produto de software
- Considerar alternativas de implementações
- Avaliar conformidade a normas e especificações

O walk-through é semelhante a uma inspeção, mas normalmente é conduzida com menos formalidade. O walk-through é principalmente organizado pelo engenheiro de software para dar aos companheiros de equipe oportunidade de revisar seu trabalho, como uma técnica de garantia.

"O propósito de uma auditoria de software é fornecer uma avaliação independente da conformidade de produtos de software e processos para padrões, diretrizes, planos e procedimentos" [IEEE1028 - 97]. A auditoria é uma atividade formalmente organizada, com participantes que têm papéis específicos, como auditor líder, outro auditor, um relator ou um iniciador, e inclui um representante da organização examinada. A auditoria identificará exemplos de não conformidade e produzirá um relatório exigindo que a equipe entre com ação corretiva.

Embora possam existir muitos nomes formais para revisões e auditorias como identificados na Norma (IEEE1028 - 97), o ponto importante a se considerar é que eles podem acontecer em praticamente qualquer produto e qualquer fase do desenvolvimento ou processo de manutenção.

2.1.3.10.3 Considerações práticas

a) Requisitos de Qualidade de Software

Vários fatores influenciam no planejando, gerenciamento e seleção de

atividades de SQM e técnicas, incluindo:

- O domínio do sistema em que o software irá residir (segurança crítica, missão crítica, negócio crítico)
- Os requisitos de sistema e do software
- Os componentes que serão utilizados no sistema comercial (externos) ou padrões (internos)
- As normas especificas de engenharia de software aplicadas.
- Os métodos e ferramentas a serem utilizados no desenvolvimento, manutenção e na avaliação e melhoramento da qualidade
- O orçamento, a equipe, a organização do projeto, planos e cronogramas de todos os processos
- Os usuários destinados ao uso do sistema
- O nível de integridade do sistema

Informação sobre estes fatores exercem influência em como os processos de SQM são organizados e documentados, como atividades específicas de SQM são selecionadas, que recursos são necessários e quais devem impor limites sobre os esforços.

Em casos onde falhas de sistema podem ter consequências extremamente severas, dependência geral (de hardware, software e humana) é o principal requisito de qualidade, se posicionando acima das funcionalidades básicas. Dependência de software inclui características como tolerância a falha, segurança, usabilidade. Confiabilidade também é um critério que pode ser definido em termos de dependência (ISO9126).

Uma base literária para sistemas deve ser indispensável (sistemas de "alta confiança" ou "alta integridade"). A terminologia para sistemas elétricos e mecânicos tradicionais que podem não incluir software tem sido importada para discutir ameaças ou perigos, riscos, integridade de sistema e conceitos relacionados, e pode ser encontrada nas referências citadas para esta seção.

O nível de integridade é determinado baseado nas possíveis consequências de falhas no software e na probabilidade das falhas. Para softwares no qual a segurança ou garantia é importante, podem ser utilizadas técnicas como análise de perigo ou ameaça para segurança a fim de se desenvolver uma atividade de planejamento que identificaria potenciais pontos de problemas. O histórico de falha

semelhantes no software também pode ajudar na identificação de quais técnicas serão mais úteis na identificação de falhas e na avaliação de qualidade. Níveis de integridade (por exemplo, gradação de integridade) são propostos em (IEEE1012 -98).

b) Caracterização de Defeitos

Os processos de SQM encontram defeitos. A caracterização destes defeitos conduz à compreensão do produto, facilita correções do processo ou do produto e informa à gerência do projeto ou ao cliente o status do processo ou produto. Existem muitas taxonomias de falhas, e, embora tentativas tenham sido feitas para se obter uma taxonomia padronizada de erros e falhas, a literatura indica que ainda existem algumas versões diferentes em uso ([Bei90, Chi96, Gra92], IEEE1044-93). Caracterização de defeitos também é usada em auditorias e revisões, o líder de revisão frequentemente apresenta uma lista de falhas fornecidas por membros da equipe para apreciação em uma reunião de revisão.

Enquanto novos métodos de design e linguagens evoluem, juntamente com os avanços em tecnologias de software globais, novas classes de defeitos aparecem e uma grande dose de esforço é exigida para interpretar as classes definidas anteriormente. Ao localizar defeitos, o engenheiro de software está interessado não só no número de defeitos, mas também nos tipos. A informação isolada, sem nenhuma classificação, não é realmente de muita utilidade na identificação das causas subjacentes dos defeitos, uma vez que alguns tipos específicos de problemas precisam ser agrupados juntos em ordem para determinar o que será feito a respeito sobre eles. O ponto é estabelecer uma taxonomia de defeito que seja significante à organização e para os engenheiros de software.

SQM encontra informações em todas as fases de desenvolvimento e manutenção de software. Na versão original em inglês deste guia, as palavras "defect" e "fault" são utilizadas como referência ao termo "defeito". Culturas ou padrões diferentes podem utilizar significados diferentes para estes termos, o que levou a tentativas de defini-los. Definições parciais tomadas, a partir da norma (IEEE610.12-90) são:

- Erro (error): "A diferença entre um resultado computado e o resultado correto"
- Defeito (fault): "Um passo, processo ou definição de dados incorreto"

- Falha (failure): "O resultado incorreto de um defeito"
- Engano (mistake): "Uma ação humana que produz um resultado incorreto"

Modelos de confiança são construídos a partir de dados obtidos em falhas durante o teste ou funcionamento do software, e assim podem ser utilizados para predizer futuros fracassos e auxiliar em decisões sobre quando os testes devem terminar. [Mus89]

Uma provável ação resultante do SQM é encontrar e remover os defeitos do produto sujeitos a análise. Outras ações permitem a obtenção do resultado perfeito a partir das atividades do SQM. Estas ações incluem analisar e resumir os resultados, e utilizar técnicas de medição para melhorar o produto e o processo bem como localizar e remover os defeitos. Melhoria de processos é discutida principalmente na KA Processo de Engenharia de Software, sendo o processo SQM uma fonte de informação.

Os dados sobre falhas e defeitos encontrados durante a implementação de técnicas de SQM podem ser perdidos a menos que sejam registrados. Para algumas técnicas (por exemplo, revisões técnicas, auditorias, inspeções), relatores estão presentes para estabelecer informações, junto com questões e decisões. Quando ferramentas automatizadas forem usadas, a saída da ferramenta pode fornecer a informação de defeito. Os dados sobre os defeitos podem ser coletados e registrados em um SCR (pedido de mudança de software) e pode ser entrado subsequentemente em algum tipo de banco de dados, manualmente ou automaticamente, de uma ferramenta de análise. Relatórios sobre defeitos são fornecidos à administração da organização.

c) Técnicas de Gerenciamento de Qualidade de Software

Técnicas de SQM podem ser categorizadas em muitas formas: estático, pessoas intensivas, analítico, dinâmico.

Técnicas estáticas envolvem análise da documentação de projeto e software e outras informações sobre os produtos de software, sem os executar. Estas técnicas podem incluir atividades pessoas intensivas (como definido em 3.3.2) ou atividades analíticas (como definido em 3.3.3) conduzidos por indivíduos, com ou sem a ajuda de ferramentas automatizadas.

A colocação para técnicas de pessoas intensivas, incluindo revisões e

auditorias, pode variar de uma reunião formal a um ajuntamento informal ou uma situação de desk-check, mas (normalmente, pelo menos) duas ou mais pessoas estão envolvidas. Uma preparação antes do tempo pode ser necessária. Outros itens de recursos de análise podem incluir checklists e resultados de técnicas analíticas e testes. Estas atividades são discutidas em (IEEE1028 -97) nas revisões e auditorias. [Fre98, Hor03] e [Jon96, Rak97]

Um engenheiro de software geralmente aplica técnicas analíticas. Às vezes vários engenheiros de software utilizam a mesma técnica, mas cada um a aplica em partes diferentes do produto. Algumas técnicas são dirigidas por ferramentas; outras são manuais. Algumas podem encontrar defeitos diretamente, mas elas tipicamente são usadas para apoiar outras técnicas. Algumas também incluem várias avaliações como parte da análise de qualidade global. Exemplos de tais técnicas incluem análise de complexidade, análise de controle de fluxo e análise algorítmica.

Cada tipo de análise tem um propósito específico e nem todos os tipos são aplicados em todos os projetos. Um exemplo de uma técnica de apoio é a análise de complexidade, que é útil para determinar se o projeto ou implementação é ou não é muito complexo para desenvolver corretamente, testar, ou manter. O resultado de uma análise de complexidade pode também ser utilizado para desenvolver casos de testes.

Técnicas de encontrar defeito, tais como análise de controle de fluxo, podem também ser utilizadas para apoiar outras atividades. Para softwares com muitos algoritmos, análise algorítmica é importante, especialmente quando um algoritmo incorreto poderia causar um resultado catastrófico. Há muitas técnicas de análise para listar todas aqui. A lista e referências fornecidas podem oferecer discernimento na seleção de uma técnica, como também sugestões para uma futura leitura.

Outros tipos de técnicas analíticas mais formais são conhecidos como métodos formais. Eles são usados para verificar requisitos de software e projeto. Prova de correção aplica a partes críticas de software. Elas foram principalmente utilizadas na verificação de partes cruciais de sistemas críticos, como segurança específica e requisitos de segurança. (Nas97)

Diferentes tipos de técnicas dinâmicas são executados ao longo do desenvolvimento e manutenção do software. Geralmente, são técnicas de teste, mas técnicas como simulação, modelo de verificação e execução simbólica também

podem ser consideradas dinâmicas. Leitura de código é considerada uma técnica estática, mas engenheiros de software experientes são capazes de executar o código em paralelo enquanto eles o leem Neste caso, leitura de código também pode ser qualificada como uma técnica dinâmica. Estas diferenças na tarefa de categorização indicam que pessoas com papéis diferentes na organização podem considerar e aplicar tais técnicas de forma diferente.

Alguns testes podem ser realizados em processos de desenvolvimento, processos SQA ou em processos V&V, mais uma vez, dependendo da organização do projeto. Pelo fato dos planejamentos SQM lidarem com testes, esta seção inclui alguns comentários sobre os mesmos. A KA Teste de Software fornece discussões e referências teóricas e técnicas para teste e automação.

Os processos de garantia descritos em SQA e V&V examinam cada saída relacionada com a especificação de requisitos de software com a finalidade de se garantir rastreamento, consistência, completude, corretude e performance. Esta confirmação também inclui os artefatos de saída dos processos de desenvolvimento e manutenção, coletando, analisando e medindo os resultados. A SQA garante que tipos apropriados de teste sejam adequadamente planejados, desenvolvidos e implementados e a V&V desenvolve planos de teste, estratégias, casos e procedimentos.

Testes são discutidos em detalhes na KA Teste de Software. Dois tipos de testes podem cair sob a pauta de SQA e V&V, devido à sua responsabilidade pela qualidade dos materiais usados no projeto:

- Avaliação e teste das ferramentas utilizadas no projeto (IEEE1462-98)
- Teste de conformidade (ou revisão de teste de conformidade) de componentes e produtos COTS utilizados no produto; existe agora um padrão para pacotes de software (IEEE1465-98)

Em alguns casos uma organização de V&V independente pode ser convidada para acompanhar o processo de teste e, às vezes, testemunhar a execução para garantir que ele seja conduzido conforme procedimentos especificados. Novamente, V&V pode ser chamada para avaliar o teste propriamente dito: adequação de planos e procedimento, e adequação e precisão de resultados.

Outro tipo de teste que pode cair sob a pauta da organização de V&V é o teste de terceiros. O terceiro não é o desenvolvedor, nem é, de qualquer forma,

associado com o desenvolvimento do produto. O terceiro é um mecanismo independente, normalmente credenciado por algum órgão da autoridade. O seu objetivo é testar um produto para conformidade de um conjunto específico de requisitos.

d) Métricas da Qualidade de Software

Os modelos de qualidade de produto de software frequentemente incluem métricas para determinar o grau de cada característica de qualidade atingida pelo produto.

Se selecionadas corretamente, métricas podem apoiar a qualidade de software (entre outros aspectos de ciclo de vida dos processos) em múltiplas formas. Elas podem ajudar no gerenciamento do processo e em tomadas de decisão. Elas podem encontrar áreas problemáticas e gargalos no processo de software; e elas podem ajudar os engenheiros de software a avaliar a qualidade do seu trabalho para propósitos SQA e para a melhoria de qualidade de processo a longo prazo.

Com a sofisticação crescente de softwares, as questões de qualidade vão além de simples questões como se o software funciona ou não, para algo como quão bem ele atinge metas de qualidade mensuráveis.

Há alguns tópicos onde métrica apoia diretamente SQM. Estes incluem ajuda para decidir quando interromper os testes. Para isto, modelos de confiança e benchmarks, ambos usando dados de falhas e fracasso, são úteis.

O custo dos processos SQM é um assunto que quase sempre é levantado durante a decisão de como um projeto deve ser organizado. frequentemente são utilizados modelos genéricos de custo, que são baseados em quando um defeito é encontrado e quanto esforço é necessário para corrigir o defeito em relação ao custo de se encontrar o defeito mais cedo no processo de desenvolvimento. Dados de projeto podem dar uma melhor estimativa de custo. Uma discussão sobre este tópico pode ser encontrada em [Rak97: pp. 39-50]. Informações relacionadas podem ser encontradas nas KAs Processo de Engenharia de Software e Gestão de Engenharia de Software.

Finalmente, os próprios relatórios de SQM fornecem informações valiosas, não só sobre estes processos, mas também sobre como todos os processos de ciclo vida do software podem ser melhorados. Discussões sobre estes tópicos são

encontradas em [McC04] e (IEEE1012 -98).

Enquanto as métricas para características de qualidade e funcionalidades do produto puderem ser úteis em si (por exemplo, o número de requisitos defeituosos ou a proporção de requisitos defeituosos), técnicas matemáticas e gráficas poderão ser aplicadas para ajudar na sua interpretação. Estas se encaixam nas categorias seguintes e são discutidos em [Fen97, Jon96, Kan02, Lyu96, Mus99].

- Baseado em estatística (por exemplo, análise de Pareto, gráficos de execução , scatter plots, distribuição normal)
- Testes estatísticos (por exemplo, teste binomial, teste do qui-quadrado (chi-squared test)
- Análise de tendências
- Predição (por exemplo, modelos de confiança)

As técnicas e testes baseados em estatística frequentemente fornecem um snapshot das áreas mais problemáticas do produto de software sob análise. As tabelas e gráficos resultantes são visualizações que os tomadores de decisões podem usar para concentrar os recursos onde eles parecem mais necessários.

Resultados de análise de tendência podem indicar que um cronograma não tem sido respeitado, tais como nos testes, ou que certas classes de falhas tornam-se mais intensas a menos que seja tomada alguma ação corretiva em desenvolvimento. As técnicas de predição ajudam no planejamento de testes e na previsão de tempo de falhas. Mais discussões sobre métricas, em geral, aparecem nas KAs Processo de Engenharia de Software e Gerenciamento de Engenharia de Software. Informações mais específicas sobre métricas de teste são apresentadas na KA Teste de Software.

Referências [Fen97, Jon96, Kan02, Pfl01] apresentam discussões sobre análise de defeito, que consiste em medir a ocorrências de defeitos e, em seguida, aplicar métodos estatísticos para entender os tipos de defeitos que acontecem mais frequentemente, ou seja, respondendo a perguntas, a fim de avaliar suas densidades. Eles também ajudam a entender as tendências, bem como técnicas de detecção que estão trabalhando, e bem como os processos de desenvolvimento e manutenção estão progredindo. Medidas de cobertura de teste ajudam estimar quanto esforço de teste ainda precisa ser feito, e a prever possíveis defeitos restantes. A partir destes métodos de medida, podem ser desenvolvidos perfis de

defeito para um domínio específico de aplicação. Então, para o próximo sistema de software dentro daquela organização, os perfis podem ser usados para guiar os processos de SQM, ou seja, para gastar o esforço onde os problemas são prováveis de acontecer. Semelhantemente, benchmarks (pontos de referência) ou típica contagem de defeitos desse domínio, podem servir como um auxílio para determinar quando o produto estará pronto para entrega.

A discussão sobre o uso dados de SQM para melhorar o processo de desenvolvimento e de manutenção aparece nas KAs Gestão de Engenharia de Software e Processo de Engenharia de Software.

	[Boe78]	[Dac01]	[Hou99]	[IEEE99]	[ISO9001-00]	[ISO900003-04]	[Jon96]	[Kia95]	[Lap91]	[Lew92]	[Llo03]	[McC77]	[Mus99]	[NIST03]	[Pfl01]	[Pre04]	[Rak97]	[Sei02]	[Wal96]	[Wei93]	[Wei96]
1. Fundamentos da qualidade de software																					
1.1 Cultura e Ética da Engenharia de Software				*																	*
1.2 Valor e Custo da Qualidade	*		*				*							*	*	*				*	*
1.3 Modelos e Características de Qualidade	*	*			*	*		*	*	*	*	*	*	*		*	*	*	*		
1.4 Melhoria da Qualidade de Software														*		*					*

Figura 27: Tópicos x referências (7)

	[Ack02]	[Ebe94]	[Fre98]	[Gil93]	[Gra92]	[Hor03]	[Lew92]	[Pfl01]	[Pre04]	[Rad02]	[Rak97]	[Sch99]	[Som05]	[Voa99]	[Wal89]	[Wal96]
2. Processos de gerencia de Qualidade de Software																
2.1 Garantia/ da Qualidade de Software	*	*	*		*	*		*	*		*	*	*	*	*	*
2.2 Verificação e Validação			*			*		*	*				*		*	*
2.3 Revisões e Auditorias	*		*	*		*	*	*	*	*	*		*	*	*	*

Figura 28: Tópicos x referências (8)

	Bas84	Bei90	Con86	Chi96	Fen97	Fre98	Fri95	Gra92	Hor03	Jon96	Kan02	Lev95	Lew92	Lyu96	McC04	Mus89	Mus99	Pen93	Pfl01	Rak97	Rub94	Sch99	Wak99	Wal89	Wal96	Wei93	Zel98
3. Considerações Práticas Qualidade de Software																											
3.1 Requisitos de Qualidade de Software									*				*							*		*		*	*		
3.2 Caracterização de Defeitos		*		*			*	*	*				*			*					*		*	*			
3.3 Técnicas de SQM	*	*	*	*	*	*	*		*	*		*				*		*		*		*	*			*	*
3.4 Métricas de Qualidade de Software					*			*		*	*			*	*		*		*	*							

Figura 29: Tópicos x referências (9)

2.1.3.11 Disciplinas Relacionadas Com Engenharia de Software

A fim de circunscrever a engenharia de software, é necessário identificar as disciplinas de engenharia de software com o qual partilha uma fronteira comum. Este capítulo identifica, em ordem alfabética, estas disciplinas relacionadas. Evidentemente, as Disciplinas Relacionadas também partilham muitas coisas em comum entre si.

Usando um consenso baseado em fonte reconhecida, este capítulo identifica, para cada disciplina relacionada:

- Uma definição informativa (quando possível)
- Uma lista de áreas do conhecimento

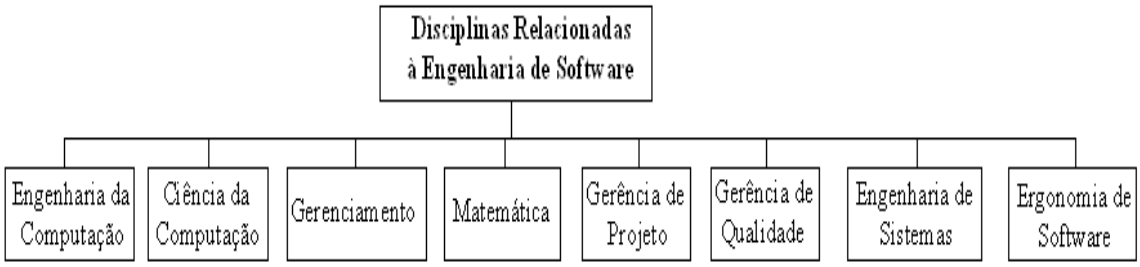


Figura 30: Disciplinas relacionadas à Engenharia de Software

2.1.3.11.1 Engenharia da Computação

O relatório de desenho do volume para computador que cria o projeto Currículos de Computação 2001 (CC2001) afirma que "engenharia da computação encarna a ciência e tecnologia de concepção, construção, implementação e

manutenção de software e hardware componentes dos modernos sistemas de computação e equipamento controlado por computador”

Este relatório identifica as seguintes Áreas de Conhecimento (conhecidas como áreas no relatório) para a engenharia da computação:

- Algoritmos e Complexidade
- Organização e Arquitetura de Computadores
- Engenharia de Sistemas de Computação
- Circuitos e Sistemas
- Lógica Digital
- Estruturas Discretas
- Processamento de Sinal Digital
- Sistemas Distribuídos
- Eletrônica
- Sistemas Integrados
- Interação Humano Computador
- Gestão de Informação
- Sistemas Inteligentes
- Redes de Computadores
- Sistemas Operacionais
- Programação Fundamental
- Probabilidade e Estatística
- Questão Social e Profissional
- Engenharia de Software
- Teste e Verificação
- VLSI / ASIC Design

2.1.3.11.2 Ciência da Computação

O relatório final do volume de Ciência da Computação do projeto Currículo de Computação 2001 (CC2001)[2] identifica as seguintes listas de áreas do conhecimento (identificado como áreas do relatório) para computação científica:

- Estruturas Discretas
- Programação Fundamental

- Algoritmos e Complexidade
- Organização e Arquitetura
- Sistemas Operacionais
- Net-Centric Computing
- Linguagem de Programação
- Interação Humano Computador
- Computação Gráfica e Visual
- Sistemas Inteligentes
- Gestão de Informação
- Questão Social e Profissional
- Engenharia de Software
- Computação Científica e Métodos Numéricos

2.1.3.11.3 Gerenciamento

As diretrizes europeias do MBA definidas pela associação europeia de corpos nacionais da abonação (EQUAL³) indicam que o mestre do grau da administração de negócio deve incluir a cobertura e a instrução em:

- Finanças
- Marketing e Vendas
- Gerenciamento de Operações
- Sistemas de Gestão da Informação
- Lei/Regra
- Gestão de Recursos Humanos
- Economia
- Análise Quantitativa
- Política e Estratégia de Negócio

2.1.3.11.4 Matemática

Duas fontes são selecionadas para identificar a lista de áreas do conhecimento para a matemática. O relatório intitulou de “critérios de abonação e procedimentos”, a Câmara Canadense de abonação da Engenharia que identificam

os elementos apropriados das seguintes áreas e devem estar atuais em um currículo da engenharia de graduação:

- Álgebra linear
- Cálculo diferencial e integral
- Equações diferenciais
- Probabilidade
- Estatísticas
- Análise numérica
- Matemática discreta

Uma lista mais focalizada de tópicos matemáticos (chamados unidades e tópicos no relatório) que sustente a tecnologia de programação pode ser encontrada no relatório de esboço do volume na tecnologia de programação do projeto de computação dos currículos 2001 (CC2001).

2.1.3.11.5 Gestão de Projetos

A gestão de projeto é definida na edição 2000 como um de guia ao corpo da gestão do projeto de conhecimento (PMBOK® Guide 6) publicado pelo instituto da gestão do projeto e adotado como IEEE STD 1490-2003, como “a aplicação do conhecimento, das habilidades, das ferramentas, e das técnicas para projetar atividades cumprir exigências do projeto”.

As áreas do conhecimento identificadas no guia PMBOK para a gestão do projeto são:

- Gerência da integração do projeto
- Gerência do espaço do projeto
- Gerência de tempo do projeto
- Gerência do custo do projeto
- Gerência de qualidade do projeto
- Gerência de recursos humanos do projeto
- Gerência de comunicação do projeto
- Gestão de riscos do projeto
- Gerência da obtenção do projeto

2.1.3.11.6 Gerência de Qualidade

A gerência de qualidade é definida na ISO 9000-2000 como “Coordenando Atividades para Dirigir e Controlar uma Organização no que diz Respeito à Qualidade.” As três referências selecionadas na gerência de qualidade são:

- Sistemas de gestão da qualidade do 9000:2000 do ISO - Fundamentos e Vocabulário
- Sistemas de gestão da qualidade do 9001:2000 do ISO – Exigências
- Sistemas de gestão da qualidade do 9004:2000 do ISO - Diretrizes para melhorias do Desempenho

A sociedade americana para a qualidade identifica as seguintes áreas do conhecimento (tópicos da avaria do primeiro nível em seu esboço) em seu corpo para a certificação de um coordenador da qualidade:

- Desenvolvimento, execução e verificação de sistemas da qualidade
- Planejando, controlando, e assegurando a qualidade do produto e do processo
- Confiabilidade e gestão de riscos
- Resolução de problema e melhoria de qualidade
- Métodos quantitativos

2.1.3.11.7 Ergonomia de software

O campo de ergonomia é definido pelo Comitê Técnico da Ergonomia no ISO 159 como segue: A “ergonomia ou (fatores humanos) é a disciplina científica relacionada com a compreensão das interações entre o ser humano e os outros elementos de um sistema, e a profissão que aplica a teoria, os princípios, os dados e os métodos ao projeto a fim de aperfeiçoar o desempenho do bem estar humano e do sistema total”.

Uma lista de áreas do conhecimento para a ergonomia como se aplica ao software é proposto abaixo:

a) Cognição

- Cognitivo Al I: raciocinando
- Aprendizagem de máquina e indução da gramática
- Métodos formais na ciência cognitiva: Linguagem

- Métodos formais na ciência cognitiva: Raciocínio
- Métodos formais na ciência cognitiva:

b) Arquitetura cognitiva

- Cognitivo AI II: aprendendo
- Fundações da ciência cognitiva
- Extração de informação do discurso e do texto
- Processamento lexical
- Aquisição de língua computacional
- A natureza de IHC
- (META-) Modelos de IHC

c) Uso e contexto dos computadores

- Organização social e trabalho humano
- Áreas de aplicação

d) Ajuste e Interação Humano – Máquina

- Características humanas
- Processamento de informação humano
- Linguagem, comunicação, interação
- Ergonomia

e) Sistema informático e arquitetura da relação

- Dispositivos de entrada e de saída
- Técnicas do diálogo
- Gênero do diálogo
- Gráficos de computador
- Arquitetura do diálogo

f) Processo de desenvolvimento

- Aproximações do projeto
- Técnicas da execução

- Técnicas da avaliação
- Sistemas do exemplo e estudos de caso

Uma lista mais focalizada de tópicos no projeto de relação homem-máquina (chamado unidades e tópicos no relatório) para finalidades do currículo da tecnologia de programação pode ser encontrada no relatório de esboço do volume da tecnologia de programação do Projeto de computação dos currículos 2001 (CC2001).

2.1.3.11.8 Engenharia de sistemas

O Conselho Internacional da Engenharia de Sistemas (INCOSE) indica que “a engenharia de sistemas é uma aproximação interdisciplinar e significa permitir a realização de sistemas bem sucedidos. Centra-se sobre a definição de necessidades do cliente e da funcionalidade exigida cedo no ciclo de desenvolvimento, documentando exigências, então continuação com síntese de projeto e validação do sistema quando considera o problema completo: operações desempenho, teste, fabricação, custo e programação, treinamento e sustentação e eliminação.”

A engenharia de sistemas integra todas as disciplinas e grupos da especialidade em um esforço de equipe que dá forma a um processo de desenvolvimento estruturado e que prossiga do conceito de produção à operação. A engenharia de sistemas considera o negócio e as necessidades técnicas de todos os clientes com o objetivo de fornecer um produto de qualidade que encontre as necessidades do usuário.

O Conselho internacional na engenharia de sistemas (INCOSE, www.incose.org) está trabalhando em um guia ao corpo de engenharia de sistemas de conhecimento. As versões preliminares incluem as seguintes áreas da competência dos primeiros níveis:

- Processos de negócio e avaliação operacional (BPOA)
- Sistema/arquitetura da solução/teste (SSTA)
- Ciclo da vida e do custo; Análise de benefício de custo (LCC & CBA)
- Serviço-habilidade/logística (S/L)
- Modelagem, simulação e Análise (MS& A)
- Gerência: Risco, configuração, linha de base (Mgt)

2.1.4 Apêndices

2.1.4.1 Apêndice A

2.1.4.1.1 Especificações da Descrição de Área Para o SWEBOK

Este documento apresenta a versão 1.9 das especificações fornecidas pela Equipe Editorial ao Especialista de Áreas do Conhecimento quanto às Descrições de Áreas do Conhecimento do Guia do Corpo de Conhecimento de Engenharia de Software (Versão de Ironman).

Este documento começa apresentando especificações dos conteúdos da Descrição da Áreas do Conhecimento. Os critérios e as exigências são propostos para divisões de tópicos, para a base lógica dessas divisões, para a descrição sucinta dos tópicos, para selecionar materiais de referência, e para identificar Áreas do Conhecimento relevantes de Disciplinas Relacionadas. Os documentos de entrada importantes também são identificados bem como é explicado seus papeis dentro do projeto. Questões não relacionados ao conteúdo, como formato de submissão e diretrizes de estilos, também são discutidas.

2.1.4.1.2 Critérios e Exigências para Propor as Divisões de Tópicos

As seguintes exigências e critérios devem ser usados quando se propuser uma separação de tópicos dentro de uma dada Área do Conhecimento:

a) Espera-se que editores associados proponham uma ou possivelmente duas divisões complementares que sejam específicas à sua Área do Conhecimento. Os tópicos encontrados em todos as divisões dentro de uma dada Área do Conhecimento devem ser idênticos.

b) Espera-se que essas divisões de tópicos sejam "razoáveis", não "perfeitas". O Guia do Corpo de Conhecimento de Engenharia de Software é visto definitivamente como um esforço multifásico, e muitas iterações dentro de cada fase bem como múltiplas fases serão necessárias para melhorar continuamente essas divisões.

c) A divisão proposta de tópicos dentro de uma Área do Conhecimento deve

decompor o subconjunto do Corpo de Conhecimento de Engenharia de Software que é “geralmente aceito.” Veja abaixo uma discussão mais detalhada disto.

d) A divisão proposta de tópicos dentro de uma Área do Conhecimento não deve presumir domínios de aplicação específicos, necessidades negociais, tamanhos das organizações, estruturas organizacionais, filosofias de gerência, modelos de ciclo de vida de software, tecnologias de software, ou métodos de desenvolvimento de software.

e) A divisão proposta de tópicos, tanto quanto possível, deve ser compatível com várias escolas do pensamento dentro da engenharia de software.

f) A divisão proposta de tópicos dentro das Áreas do Conhecimento deve ser compatível com a divisão da engenharia de software geralmente encontrado na indústria e na literatura sobre engenharia de software e padrões.

g) Espera-se que a divisão proposta de tópicos seja tão inclusiva quanto possível. Julga-se que é melhor sugerir tópicos demasiadamente e descartá-los do que fazer ao contrário.

h) Espera-se que os Editores Associados das Áreas do Conhecimento adotem a posição que embora "os temas" seguintes sejam comuns através de todas as Áreas do Conhecimento, eles também sejam uma parte integrante de todas as Áreas do Conhecimento e por isso devem ser incorporados na divisão proposta de tópicos de cada Área do Conhecimento. Esses temas comuns são qualidade (em geral) e medida. Por favor observe que o problema de como tratar propriamente essa "corrida cruzada" ou “tópicos ortogonais” e se eles devem ser tratados de uma maneira diferente ainda não foi completamente resolvido.

i) As divisões propostas devem ter no máximo dois ou três níveis de profundidade. Mesmo embora nenhum limite superior ou inferior seja imposto ao número de tópicos dentro de cada Área do Conhecimento, espera-se que os Editores Associados das Áreas do Conhecimento proponham um número razoável e

gerenciável de tópicos por Área do Conhecimento. Também deve-se dar ênfase na seleção dos próprios tópicos e não na sua organização em uma hierarquia apropriada. Os nomes dos tópico propostos devem ser expressivos o bastante para terem significado mesmo quando citados fora do Guia do Corpo de Conhecimento de Engenharia de Software .

j) A descrição de uma Área do Conhecimento incluirá um diagrama (na forma de árvore) descrevendo a divisão do conhecimento.

2.1.4.1.3 Critérios e exigências para descrever Tópicos

A descrição dos tópicos deve ser somente a necessária, assim o leitor pode selecionar o material de referência apropriado segundo suas necessidades.

2.1.4.1.4 Critérios e exigências para Selecionar Material de Referência

a) Materiais de referência específicos devem ser identificados para cada tópico. Cada material de referência naturalmente pode cobrir múltiplos tópicos.

b) Os materiais de referência propostos podem ser capítulos de livros, artigos referenciados de jornais, artigos referenciados de conferências, relatórios técnicos ou industriais, ou qualquer outro tipo de artefato reconhecido, como documentos web. Eles devem ser disponíveis de maneira geral e não devem ser de natureza confidencial. A referência deve ser tão exata quanto possível, identificando quais capítulos ou seções específicas são relevantes.

c) Os materiais de referência propostos devem ser em inglês.

d) Um montante razoável de materiais de referência deve ser selecionados para cada Área do Conhecimento. As seguintes diretrizes devem ser usadas na determinação do quanto é razoável:

- Se o material de referência foi escrito de uma maneira coerente seguindo a divisão proposta de tópicos e em um estilo uniforme (por exemplo, em um novo livro baseado na descrição da Áreas do

Conhecimento proposta), uma média razoável de número de páginas seria 500. Contudo, este meta pode não ser atingível selecionando o material de referência existente devido a diferenças em estilo, sobreposição e redundância com outros materiais de referência selecionados.

- O montante de material de referência seria razoável se ele se consistisse do material de estudo da Área do Conhecimento para o exame de licenciamento de engenharia de software que um graduado passaria depois de concluir quatro anos de experiência de trabalho.
- O Guia do Corpo de Conhecimento de Engenharia de Software é destinado por definição para ser seletivo na sua escolha de tópicos e materiais de referência associados. A lista dos materiais de referência de cada Área do Conhecimento deve ser examinada e será apresentada como “uma seleção informada e razoável” e não como uma lista definitiva.
- Materiais de referência adicionais podem ser incluídos na lista “Leituras Complementares”. Essas leituras complementares devem ainda estar relacionadas a divisão de tópicos. Elas também devem discutir o conhecimento geralmente aceito. Não deve haver uma matriz entre o material de referência enumerado em Leituras Complementares e os tópicos individuais.

e) Se considerado factível e rentável pela Sociedade de Computação IEEE, o material de referência selecionado será publicado no Web site do Guia do Corpo de Conhecimento de Engenharia de Software. Para facilitar esta tarefa, deve ser dada preferência ao referir materiais quais os direitos autorais já pertencem à Sociedade de Computação IEEE. Contudo, isto não deve ser visto como um constrangimento ou uma obrigação.

f) Uma matriz do material de referência versus tópicos deve ser fornecida.

2.1.4.1.5 Critérios e exigências para Identificar KAs das Disciplinas relacionadas

Espera-se que os Editores Associados das Áreas do Conhecimento identifiquem em uma seção em separado que Áreas do Conhecimento das Disciplinas Relacionadas são suficientemente relevantes para as Áreas do Conhecimento de Engenharia de Software que foram atribuídas a terem o conhecimento esperado por um graduado com quatro anos da experiência.

Esta informação será especialmente útil e empenhará muito diálogo entre a inciativa do Guia do Corpo de Conhecimento de Engenharia de Software e nossas iniciativas irmãs responsáveis por definir um currículo comum de engenharia de software e normas de realização padrão para engenheiros de software.

A lista de Áreas do Conhecimento de Disciplinas Relacionadas pode ser encontrada na Lista Base Proposta de Disciplinas Relacionadas. Se considerado necessário e se acompanhado de uma justificativa, os Especialistas das Áreas do Conhecimento também podem propor Disciplinas Relacionadas adicionais não incluídas ou identificadas na Lista Base Proposta de Disciplinas Relacionadas. (Por favor observe que uma classificação dos tópicos das Disciplinas Relacionadas foi produzida mas será publicada no Web site em uma data posterior em um documento de trabalho separado. Por favor contate a equipe editorial para mais informação).

2.1.4.1.6 Índice dos Conteúdos Comuns

As descrições de Áreas do Conhecimento devem usar o seguinte índice de conteúdo:

- Introdução
- Divisão de tópicos de Áreas do Conhecimento (para objetivos de esclarecimento, acreditamos que esta seção deve ser colocada na frente e não em um apêndice no fim do documento. Também, deve ser acompanhado por uma figura que descreve a divisão)
- Matriz de tópicos versus material de Referência
- Referências recomendadas da Áreas do Conhecimento descrita (por favor não os misture com referências usadas para escrever a descrição da Áreas do Conhecimento)
- Lista de Leituras Complementares

2.1.4.1.7 O que Queremos Dizer com “Conhecimento Geralmente Aceito”?

O corpo de conhecimento de engenharia de software é um termo tudo em um que descreve a soma do conhecimento dentro da profissão de engenharia de software. Contudo, o Guia do Corpo de Conhecimento da Engenharia de Software procura identificar e descrever qual subconjunto do corpo do conhecimento é geralmente aceito ou, em outras palavras, o corpo principal do conhecimento. Para ilustrar melhor o que “o conhecimento geralmente aceito” é quanto a outros tipos do conhecimento, a Figura 31 propõe um esquema de três categorias preliminares para classificar o conhecimento.

O Instituto de Gerenciamento de Projetos no seu Guia do Corpo de Conhecimento de Gerenciamento de Projetos1 conhecimento “geralmente aceito” para gerenciamento de projetos da seguinte maneira: “Meios ‘geralmente aceitos’ que o conhecimento e as práticas descritas são aplicáveis à maior parte dos projetos na maior parte do tempo, tendo consenso comum sobre o seu valor e utilidade. ‘Geralmente aceito’ não significa que o conhecimento e as práticas descritas são ou devem ser aplicados uniformemente em todos os projetos; a equipe de gerenciamento de projetos é sempre responsável por determinar o que é apropriado para qualquer projeto dado.”

O Guia do Corpo de Conhecimento de Gerenciamento de Projetos é agora um Padrão IEEE. No encontro inicial Mont Tremblant em 1998, o Conselho Consultivo Industrial melhor definiu “geralmente aceito” como conhecimento a ser incluído no material de estudo de um exame de engenharia de software que um graduado passaria depois de concluir quatro anos de experiência de trabalho. Essas duas definições devem ser vistas como complementares.

Também se espera que Editores Associados das Áreas do Conhecimento estejam pensando a frente nas suas interpretações, levando em consideração não só o que é “geralmente aceito” hoje, mas o que eles esperam que seja “geralmente aceito” em um período de 3 a 5 anos.

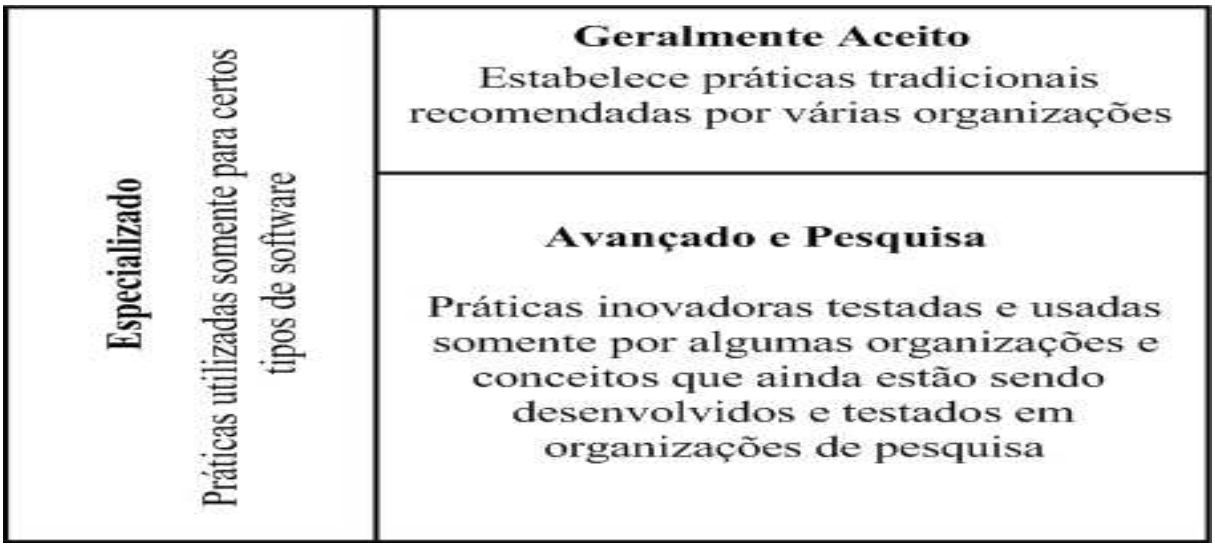


Figura 31: Categorias de Conhecimento

2.1.4.1.8 Comprimento de Descrição de Áreas do Conhecimento

Espera-se que as Descrições das Áreas do Conhecimento estejam atualmente em torno de 10 páginas usando o formato da Conferência Internacional de Engenharia de Software, definido abaixo. Isto inclui texto, referências, apêndices, tabelas, etc. Isto, naturalmente, não inclui os próprios materiais de referência. Este limite não deve, contudo, ser visto como um constrangimento ou uma obrigação.

2.1.4.1.9 Papel da Equipe Editorial

Alain Abran e James W. Moore são os Editores Executivos e são responsáveis por manter boas relações com a Sociedade de Computação IEEE, o Conselho Consultivo Industrial, o Conselho de Controle de Modificação Executivo, e o Painel de Peritos bem como pela estratégia geral, aproximação, organização, e consolidação do projeto.

Pierre Bourque e Robert Dupuis são os Editores e são responsáveis pela coordenação, operação e logística deste projeto. Mais especificamente, os Editores são responsáveis por desenvolver o plano de projeto e a especificação da descrição das Áreas do Conhecimento, coordenando Editores Associados das Áreas do Conhecimento e suas contribuições, recrutando revisores e chefes de revisão, bem como coordenando vários ciclos de revisão.

Por isso os Editores são responsáveis pela coerência do Guia inteiro e por identificar e estabelecer conexões entre as Áreas do Conhecimento. Os Editores e os Editores Associados das Áreas do Conhecimento negociarão a resolução de

lacunas e sobreposições entre Áreas do Conhecimento.

2.1.4.1.10 Documentos Relacionados Importantes

a) P. Bourque, R. Dupuis, A. Abran, J. W. Moore, L. Tripp, e D. Frailey, “Uma Lista Básica de Áreas do Conhecimento da Versão Stone Man do Guia do Corpo de Conhecimento de Engenharia de Software,” Université du Québec à Montréal, Montréal, Fevereiro de 1999.

Baseado na versão Straw Man, nas discussões mantidas e expectativas afirmadas na reunião de início do Conselho Consultivo Industrial, em outras propostas de corpo de conhecimento, e em critérios definidos neste documento, este documento propõe uma lista básica de dez Áreas do Conhecimento da Versão Teste do Guia do Corpo de Conhecimento de Engenharia de Software. Esta base pode desenvolver-se naturalmente com o progresso de trabalhos e identificação de problemas durante o curso do projeto.

b) P. Bourque, R. Dupuis, A. Abran, J. W. Moore, e L. Tripp, “Uma Lista Básica Proposta de Disciplinas Relacionadas da Versão Stone Man do Guia do Corpo de Conhecimento de Engenharia de Software,” Université du Québec à Montréal, Montréal, Fevereiro de 1999.

Baseado na versão Straw Man, nas discussões mantidas e expectativas afirmadas na reunião de início do Conselho Consultivo Industrial, e no trabalho subsequente, este documento propõe uma lista básica de Disciplinas Relacionadas e Áreas do Conhecimento dentro dessas Disciplinas Relacionadas. Este documento foi submetido e discutido com o Conselho Consultivo Industrial, e uma lista reconhecida de Áreas do Conhecimento ainda tem de ser identificada para certas Disciplinas Relacionadas. Os editores associados serão informados sobre a evolução deste documento.

c) P. Bourque, R. Dupuis, A. Abran, J. W. Moore, L. Tripp, K. Shyne, B. Pflug, M. Maya, e G. Tremblay, Guia do Corpo de Conhecimento de Engenharia de Software - Versão Straw Man, relatório técnico, Université du Québec à Montréal, Montréal, Setembro de 1998.

Este relatório é a base do projeto inteiro. Ele define a estratégia geral do projeto, base lógica, e princípios subjacentes, propondo uma lista inicial de Áreas do Conhecimento e Disciplinas Relacionadas.

d) 4. J. W. Moore, Padrões de Engenharia de Software, O Mapa de Caminho do Usuário, IEEE Computer Society Press, 1998.

Este livro descreve o alcance, papéis, usos e tendências de desenvolvimento dos padrões de engenharia de software mais amplamente utilizados. É concentrado em importantes atividades de engenharia de software — qualidade e gerenciamento de projetos, engenharia de sistemas, confiança, e segurança. A análise e o reagrupamento das coleções de padrões expõem as relações chave entre padrões. Mesmo embora o Guia do Corpo de Conhecimento de Engenharia de Software não seja um projeto de desenvolvimento de padrões de engenharia de software por si só, um cuidado especial será tomado em todas as partes do projeto quanto à compatibilidade do Guia com o atual IEEE e com a Coleção de Padrões de Engenharia de Software ISO.

e) Glossário Padrão IEEE de Terminologia de Engenharia de Software, std 610.12-1990, IEEE, 1990.

A hierarquia de referências de terminologia é o Dicionário Acadêmico de Merriam Webster (10 ed.), IEEE std 610.12, e novas definições propostas se necessário.

f) Tecnologia da Informação – Processos de Ciclo de Vida de Software, Internacional std ISO/IEC 12207:1995 (E), 1995.

Este padrão é considerado o padrão chave quanto à definição do processo de ciclo de vida e foi adotado por dois corpos de padronização principais da engenharia de software: ISO/IEC JTC1 SC7 e o Comitê de Padrões de Engenharia de Software da Sociedade de Computação IEEE. Também foi indicado como o padrão fundamental em torno o qual o Comitê de Padrões de Engenharia de Software (SESC) está harmonizando atualmente a sua coleção inteira de padrões. Este padrão foi uma entrada chave à versão Straw Man.

Mesmo embora não tenhamos intenção que o Guia do Corpo de

Conhecimento de Engenharia de Software seja totalmente 12207-compatível, este padrão permanece uma entrada chave à versão Stone Man e cuidados especiais serão tomados em todas as partes do projeto quanto à compatibilidade do Guia com o padrão 12207.

g) Merriam Webster's Collegiate Dictionary (10 ed.).
Veja nota para o padrão IEEE 610.12.

2.1.4.1.11 Estilo e Diretrizes Técnicas

As Descrições de Área do Conhecimento devem enquadrar-se com a Conferência Internacional de Procedimentos de Engenharia de Software (os padrões são disponíveis em http://sunset.usc.edu/icse99/cfp/technical_papers.html).
Espera-se que as Descrições das Áreas do Conhecimento sigam o Guia de Estilo da Sociedade de Computação IEEE. Ver http://www. computer.org/author/style/cs-style.htm.

O formato preferencial de submissão é o Microsoft Word. Por favor contate a Equipe Editorial se isto não for factível para você.

2.1.4.1.12 Outras Diretrizes Detalhadas

Referindo o guia, recomendamos que você use o título inteiro "Guia do SWEBOK" em vez de só "SWEBOK".
Por objetivo de simplicidade, recomendamos que os Editores Associados das Áreas do Conhecimento evitem notas. Em vez disso, eles devem tentar incluir o seu conteúdo no texto principal.

Recomendamos usar no texto referências explícitas para padrões, ao contrário de inserir simplesmente números que referem itens na bibliografia. Acreditamos que isto permite que o leitor seja melhor exposto à fonte e ao alcance do padrão.

O texto que acompanha figuras e tabelas deve ser evidente ou conter texto relacionado suficiente. Isto poderá assegurar que o leitor saiba o que as figuras e as tabelas significam. Assegure-se que você usa a informação atual de referências (versões, títulos, etc.).

Para assegurar-se que alguma informação contida no Guia do SWEBOK não

fique rapidamente obsoleta, por favor evite denominar diretamente ferramentas e produtos. Em vez disso, tente denominar suas funções. A lista de ferramentas e produtos pode sempre ser adicionada a um apêndice.

Espera-se que você explique todos os acrônimos usados nos mínimos detalhes e use todos os direitos autorais apropriados, marcas de serviço, etc.
As Descrições de Áreas do Conhecimento devem sempre ser escritas na terceira pessoa.

2.1.4.1.13 Edição

A Equipe Editorial e os editores profissionais editarão as Descrições das Áreas do Conhecimento. A Edição inclui a edição de cópia (gramática, pontuação, e capitalização), edição do estilo (conformidade ao estilo das revistas da casa, Sociedade de Computação), e edição do conteúdo (fluxo, significado, clareza, correção, e organização). A edição final será um processo colaborativo no qual a Equipe Editorial e os autores colaboraram para realizar uma Descrição das Áreas do Conhecimento concisa, bem expressa, e útil.

2.1.4.1.14 Liberação de Direitos Autorais

Todas as propriedades intelectuais associadas com o Guia do Corpo de Conhecimento de Engenharia de Software permanecerão com a Sociedade de Computação IEEE. Foi pedido aos Editores Associados das Áreas do Conhecimento que assinassem um formulário de liberação de direitos autorias.

Também é entendido que o Guia do Corpo de Conhecimento de Engenharia de Software será posto em domínio público pela Sociedade de Computação IEEE, gratuita por meio de tecnologia web ou por outros meios.

2.1.4.2 Apêndice B

2.1.4.2.1 Evolução do SWEBOK

Apesar de guia para o Corpo de conhecimento de engenharia de software ser uma referência em atingir uma grande concordância no conteúdo da disciplina de engenharia de software, não é o fim de um processo. O guia de 2004 é simplesmente a edição corrente do guia que continuará evoluindo para corresponder às necessidades da comunidade de engenharia de software. O planejamento da evolução ainda não está completo, mas um esboço da tentativa do processo é oferecido nessa seção. Como foi descrito, este processo está sendo aprovado pelo "Board" do Conselho de projetos industriais e resumido pelo "board" de governadores da Sociedade de Computação IEEE, mas ainda não está financiada ou implementada.

2.1.4.2.2 Stakeholders

A Adoção me larga escala do Guia SWEBOK tem produzido uma comunidade substancial de partes envolvidas em adição à própria Sociedade de Computação. Há um número de projetos - dentro e fora da Sociedade de Computação- que estão coordenando o seu conteúdo com o conteúdo do Guia SWEBOK.(Mais sobre isso em um momento). Algumas corporações, incluindo alguns dos membros do "Board" do Conselho de projetos industriais, adotaram o guia para uso em seus programas internos para educação e treinamento. Em um amplo sentido, a comunidade de praticantes de engenharia de software,profissionais da comunidade desenvolvedora e comunidade da educação prestam atenção ao guia do SWEBOK para ajudar a a definir o escopo de seus esforços. Um notável grupo de parte envolvida são os titulares de certificação da Sociedade de computação - Profissional de de Desenvolvimento de Software Certificado - porque o escopo do exame CSDP é altamente alinhado com o escopo do guia SWEBOK.A sociedade de Computação IEEE e outras organizações estão agora conduzindo um número de projetos que dependem da evolução do guia SWEBOK:

- O exame CSDP, inicialmente desenvolvido em paralelo com o guia SWEBOK, irá evoluir para algo mais próximo do guia, ambos em escopo e material de referência.
- O currículo da sociedade de aprendizagem a distância de computação para

engenheiros de software terá o mesmo escopo que o guia SWEBOK. Uma visão geral inicial do curso já está disponível.

- Apesar dos objetivos da educação para estudantes não graduados difiram um pouco daqueles de desenvolvimento profissional,o projeto conjunto ACMM/IEEE-CS para desenvolver um currículo de engenharia de software para não graduados é altamente reconciliados com o escopo do guia SWEBOK.
- O comitê de padrões para Engenharia de software (SESC) tem organizado essa coleção pelas áreas de conhecimento do guia SWEBOK e a associação de padrões IEEE já publicou uma edição em CD-ROM dos padrões de engenharia de software que refletem sua organização.
- ISO/IEC JTC1/SC7, os padrões internacionais de organização para software e engenharia de software, está adotando o guia SWEBOK como relatório técnico ISO/IEC 19759 e harmonizando sua coleção com a do IEEE.
- A editora da Sociedade de Computação IEEE, em cooperação com o SESC, está desenvolvendo uma série de livros baseados nos padrões de engenharia de software e no guia SWEBOK.
- O portal da sociedade de computação de engenharia de software ("SE online"), atualmente em planejamento, será organizado por áreas de conhecimento do guia SWEBOK
- O uso da versão trial do SWEBOK foi traduzido para o Japonês. Prevê-se que em 2004 a versão será também traduzida em Japonês, chinês e possivelmente outras línguas

O CSDP adicionou uma área de conhecimento, Práticas de negócio e engenharia econômica,para as dez áreas de conhecimento cobertas pelo guia SWEBOK.

2.1.4.2.3 O Processo de Evolução

Obviamente, um produto com tanta aceitação precisa evoluir de uma forma aberta,consultiva, deliberada e transparente para que outros projetos possam coordenar esforços com êxito. A estratégia atualmente planejada é evoluir o guia SWEBOK usando uma abordagem "time-boxed". A abordagem time-box é selecionada porque permite que o guia SWEBOK coordene e realize revisões de

projeto antecipadamente a uma data fixada para convergência. O time box inicial está planejado atualmente para quatro anos em duração.

No início de cada time box, em consulta com a coordenação de projetos,um plano global para revisão de quatro anos precisa ser determinado. Durante o primeiro ano, mudanças estruturais para o guia SWEBOK (isto é, mudanças em número ou escopo de áreas de conhecimento) precisam ser determinadas. Durante o segundo e o terceiro anos,a seleção e o tratamento de tópicos de dentro das áreas de conhecimento precisa ser revisado. Durante o quarto ano, o texto das descrições das áreas de conhecimento precisa ser revisado e referências atualizadas precisam ser selecionadas.

O projeto global precisa ser gerenciado por um comitê de voluntários da Sociedade de computação e representantes projetos de coordenação. O comitê precisa ser responsável por definir planos globais, coordenado com as partes envolvidas, e recomendar a aprovação de uma revisão final. O comitê deve ser ajudado por um comitê conselheiro do SWEBOK(SWAC) composto de utilizadores do guia SWEBOK. O SWAC precisa também focar para obter suporte financeiro corporativo para a evolução do guia SWEBOK. Suporte financeiro corporativo passado permitiu tornar o guia SWEBOK disponível gratuitamente em um Web Site. Suporte futuro irá permitir continuar a prática para edições futuras.
De forma fictícia, cada dos quatros anos precisa incluir um ciclo de oficina, elaboração,votação e resolução por voto. Um ciclo anual precisa envolver as seguintes atividades:

- Uma oficina, organizada como parte de uma conferência maior, precisa especificar problemas para tratamento durante o próximo ano, priorizar os problemas, recomendar abordagens para lidar com eles, e nomear redatores para implementar as abordagens.
- Cada redator deve escrever ou modificar a descrição da área de conhecimento usando a abordagem recomendada pela oficina e referências disponíveis. No ano final do ciclo, os redatores devem recomendar referências específicas atualizadas para citação no guai SWEBOK. Redatores também devem ser responsáveis para modificar suas redações em resposta a comentários de votantes.
- Cada ciclo anual deve incluir votações nas revisões para as descrições das

áreas de conhecimento. Votantes devem revisar as redações das descrições das áreas de conhecimento e referências recomendadas, oferecer comentários, e votar a aprovação das revisões. Votação deve ser aberta a membros da sociedade de computação e outros participantes qualificados. (Não membros devem pagar honorários para custear o despesa de votação). titulares do CSDP devem ser particularmente bem-vindos como membros do grupo de votação ou como voluntários em outros papéis. Uma resolução do comitê de votação deve ser selecionada por um comitê de gerenciamento para servir como intermediário entre os redatores e os votantes. Seu trabalho é determinar o consenso para mudanças requisitadas pelo grupo de votação e para garantir que os redatores irão implementar as mudanças necessárias. Em alguns casos, o comitê de resolução de votos precisa formular questões para guiar a revisão da redação. Cada objetivo anual é alcançar consenso entre o grupo de votação nas novas e revisadas áreas de conhecimento e ganhar o voto de aprovação dos votantes. Apesar do guia SWEBOK não mudar até o fim do time box, o material aprovado de cada ciclo anual estará disponível gratuitamente na conclusão do time box, o produto completo, guia SWEBOK 2008, será revisado e aprovado por um "board" da Sociedade de Computação e governadores para publicação. Se o suporte financeiro contínuo puder ser obtido, o produto estará disponível gratuitamente em um Web Site.

2.1.4.2.4 Mudanças Antecipadas

É importante notar que o guia SWEBOK é inerentemente um documento conservador por várias razões. Primeiro, limita-se ao conhecimento das características de engenharia de software; então informações de disciplinas relacionadas - mesmo disciplinas aplicadas por engenheiros de software - é omitida. Segundo, é desenvolvida e aprovada por um processo consensual, para que possa apenas gravar informação para que a concordância em grande escala possa ser obtida.

2.1.4.3 Apêndice C

2.1.4.3.1 Distribuição dos Padrões de EG da IEEE/ISO para as KAs do SWEBOK

O apêndice C do SWEBOK fornece uma lista com os padrões de engenharia de software produzidos pelo IEEE e ISO / IEC JTC1/SC7, bem como alguns padrões selecionados a partir de outras fontes. Para cada padrão, o seu número e o título é fornecido. Além disso, a coluna "Descrição" fornece o material extraído do resumo ou outro material da norma introdutória. Cada uma das normas é mapeado para as áreas de conhecimento do Guia SWEBOK.

Em alguns casos, como o vocabulário de normas-padrão do mesmo se aplica a todos os KAS, em tais casos, aparece um X em cada coluna. Na maioria dos casos, cada norma tem uma alocação forçada a uma única e principal área de conhecimento; esta atribuição está marcada com um "P". Em muitos casos, existem distribuições secundárias de KAs, estas são marcadas com um "S". A lista é ordenada por número de norma, independentemente da sua categoria (IEEE, ISO, etc.).

Número padrão	Nome padrão	Descrição	Requisitos de software	Design de software	Construção de software	Testes de software	Manutenção de Software	Gerência da Configuração de Software	Gerência de Engenharia de Software	Processo de Engenharia de Software	Ferramentas e Métodos da Engenharia de Software	Qualidade de Software
IEEE Std 610.12-1990 (R2002)	Padrão IEEE Vocabulário de terminologias de Engenharia de Software	Este padrão é o vocabulário de terminologias de engenharia de software	X	X	X	X	X	X	X	X	X	X
IEEE Std 730-2002	Padrão IEEE para qualidade e garantia de planejamento de software	Este padrão especifica o formato e conteúdo dos planos de garantia e qualidade de software						S	S			P
IEEE Std 828-1998	Padrão IEEE para planejamento da gerência de configuração de software	Este padrão especifica o conteúdo do plano de Gerência de configuração de software juntamente com requisitos para atividades específicas						P	S			
IEEE Std 829-1998	Padrão IEEE para documentação de teste de software	Este padrão descreve a forma e conteúdo de um conjunto básico de documentação para planejamento, execução e relatório de teste de software			S	P						S
IEEE Std 830-1998	IEEE Prática recomendada para especificação de requisitos de software	Este documento recomenda o conteúdo e características da especificação de requisitos de software. São fornecidos esboços de exemplo.	P									
IEEE Std 982.1-1988	Padrão IEEE Dicionário de medidas para produção de software confiável	Este padrão fornece um conjunto de medidas para avaliar a confiabilidade de um produto de software e para a obtenção de previsões iniciais da confiabilidade de um produto em desenvolvimento		S	S	S			S			P
IEEE Std 1008-1987 (R 2003)	Padrão IEEE para unidade de teste de software	Este padrão descreve uma abordagem sólida de unidade de teste de software, e os conceitos e premissas sobre as quais se baseia. Ele também fornece informação, orientação e recursos.			S	P				S		S

Número padrão	Nome padrão	Descrição	Requisitos de software	Design de software	Construção de software	Testes de software	Manutenção de Software	Gerência da Configuração de Software	Gerência de Engenharia de Software	Processo de Engenharia de Software	Ferramentas e Métodos da Engenharia de Software	Qualidade de Software
IEEE Std 1012-1998 e 1012a-1998	Padrão IEEE para verificação e validação de software	Este padrão descreve o processo de verificação e validação de software que são usados para determinar se os produtos de software satisfazem os requisitos da atividade e também se os software satisfazem as necessidades do usuário para o fim pretendido. O escopo inclui análise, avaliação, revisão, inspeção, avaliação e testes de produtos e processos.										P
IEEE Std 1016-1998	IEEE Prática recomendada para descrição de design de software	Este documento recomenda o conteúdo e organização das descrições de design de software		P								
IEEE Std 1028-1997 (R 2002)	Padrão IEEE para análise de software	Este padrão define cinco tipos de análises de software e procedimentos para sua execução. Os tipos incluem análise de gerenciamento, análise técnica, inspeções, simulações e auditorias.	S	S	S			S	S			P
IEEE Std 1044-1993 (R2002)	Padrão IEEE Classificação para anomalias de software	Este padrão fornece uma abordagem uniforme para a classificação das anomalias detectadas nos softwares e sua documentação. Inclui listas úteis de classificações de anomalias de dados relacionados				S			S	S		P
IEEE Std 1045-1992 (R 2002)	Padrão IEEE para métricas de produtividade de software	Este padrão fornece uma terminologia consistente para medição da produtividade de software e define uma forma coerente para medir os elementos que estão inseridos nos mecanismos de produtividade de software.							P	S		
IEEE Std	Padrão IEEE para planejamento e	Este padrão descreve o formato e conteúdo do							P			

Número padrão	Nome padrão	Descrição	Requisitos de software	Design de software	Construção de software	Testes de software	Manutenção de Software	Gerência da Configuração de Software	Gerência de Engenharia de Software	Processo de Engenharia de Software	Ferramentas e Métodos da Engenharia de Software	Qualidade de Software
1058-1998	gerenciamento de projeto de software	planejamento e gerenciamento de projeto de software										
IEEE Std 1061-1998	Padrão IEEE para Metodologia e métrica da qualidade de software	Este padrão descreve uma metodologia – abrangendo todo o ciclo de vida – para o estabelecimento dos requisitos de qualidade e identificação, implementação e validação de medidas correspondentes			S		S		S	S		P
IEEE Std 1062, 1998 Edition	IEEE Prática recomendada para aquisição de software	Este documento recomenda um conjunto práticas úteis que podem ser selecionadas e aplicadas durante a aquisição de software. É principalmente indicado para aquisições que incluem desenvolvimento ou modificação, em vez de compra de software de prateleira.	S						P			
IEEE Std 1063-2001	Padrão IEEE para documentação de software de usuário	Este padrão fornece os requisitos mínimos para a estrutura, conteúdo e formato da documentação de usuário – impressos e eletrônicos.			P							S
IEEE Std 1074-1997	Padrão IEEE para Processos de Ciclo de Vida de Desenvolvimento de Software	Este padrão descreve uma aproximação para a definição dos processos de ciclo de vida do software								P		
IEEE Std 1175.1-2002	Guia IEEE para ferramentas CASE de Interconexão, Classificação e Descrição	Este padrão é o primeiro de uma série planejada de padrões sobre a integração de ferramentas CASE dentro de um ambiente produtivo de engenharia de software. Esta parte descreve conceitos fundamentais e introduz as partes restantes (planejadas)									P	

Número padrão	Nome padrão	Descrição	Requisitos de software	Design de software	Construção de software	Testes de software	Manutenção de Software	Gerência da Configuração de Software	Gerência de Engenharia de Software	Processo de Engenharia de Software	Ferramentas e Métodos da Engenharia de Software	Qualidade de Software
IEEE Std 1219-1998	Padrão IEEE para Manutenção de Software	Este padrão descreve um processo para manutenção de software e seu gerenciamento					P			S		
IEEE Std 1220-1998	Padrão IEEE para Aplicação e Gerenciamento dos Processos de Engenharia de Sistemas	Este padrão descreve as atividades dos sistemas de engenharia e processos necessários ao longo de um ciclo de vida do sistema para desenvolver sistemas que unam as necessidades dos clientes e suas e limitações.								P		
IEEE Std 1228-1994	Padrão IEEE para Planejamento de Segurança de Software	Este padrão descreve o conteúdo mínimo de um plano para aspectos de software de desenvolvimento, aquisição, manutenção e retirada de um sistema de segurança crítica	S			S			S			P
IEEE Std 1233, 1998 Edition	Guia IEEE Para Especificação de Requisições de Desenvolvimento de Sistemas	Este documento fornece orientações sobre o desenvolvimento de um Sistema de Especificação de Requisitos, abrangendo a identificação, organização, apresentação e modificação dos requisitos. Ele também proporciona orientação sobre as características e qualidades dos requisitos.	P									
EEE Std 1320.1-1998	Padrão IEEE para Linguagem de Modelagem Funcional – Sintaxe e Semântica para IDEF0	Este padrão define a modelagem IDEF0 como linguagem utilizada para representar as decisões, ações, e atividades de uma organização ou sistema.	S	S						S	P	

Número padrão	Nome padrão	Descrição	Requisitos de software	Design de software	Construção de software	Testes de software	Manutenção de Software	Gerência da Configuração de Software	Gerência de Engenharia de Software	Processo de Engenharia de Software	Ferramentas e Métodos da Engenharia de Software	Qualidade de Software
		IDEF0 pode ser usada para definir as exigências em termos de funções a serem desempenhadas por um determinado sistema.										
EEE Std 1320.2-1998	Padrão IEEE para Modelagem Conceitual – Sintaxe da Linguagem e Semântica para IDEF1X 97 (IDEFobject)	Este padrão define duas linguagens conceituais de modelagem, chamados coletivamente IDEF1X97 (IDEFObject). A linguagem suporta a implementação de bancos de dados relacionais, objeto de bancos de dados e modelos de objeto.	S	S							P	
IEEE Std 1362-1998	Guia IEEE de Tecnologia da Informação - Sistema de Definição - Conceito de Operações (CONOPS) Documento	Este documento fornece orientação sobre o formato e o conteúdo de um Conceito de Operações (CONOPS) documento, descrevendo as características de um sistema de propostas do ponto de vista dos usuários.	P									
IEEE Std 1420.1-1995, 1420.1a-1996, and 1420.1b-1999 (R2002)	Padrão IEEE para Tecnologia da Informação – Reutilização de Software – Modelo de Dados para Interoperabilidade da Biblioteca de Reutilização	Este padrão e seus suplementos descrevem informações das bibliotecas de reutilização de software que devem ser capazes de mudar a ordem de alternância das propriedades.									P	
IEEE Std	Padrão IEEE – Adoção do padrão	Este padrão fornece direcionamentos para a									P	

Número padrão	Nome padrão	Descrição	Requisitos de software	Design de software	Construção de software	Testes de software	Manutenção de Software	Gerência da Configuração de Software	Gerência de Engenharia de Software	Processo de Engenharia de Software	Ferramentas e Métodos da Engenharia de Software	Qualidade de Software
1462-1998 //ISO/IEC 14102:1995	internacional isso/IEC 14102:1995 – Tecnologia da Informação - Direcionamento para a Avaliação e Seleção de Ferramentas CASE	avaliação e seleção de ferramentas CASE.										
IEEE Std 1465-1998 //ISO/IEC 12119	Padrão IEEE, Adoção do Padrão Internacional ISO/IEC 12119:1994(E), Tecnologia da Informação – Pacotes de Software – Qualidades de Requisitos e Testes	Este padrão descreve requisitos de qualidade especificamente adequados para pacotes de software e orientações sobre testes de pacotes contra tais requisitos.	S									P
IEEE Std 1471-2000	IEEE Práticas Recomendadas para Descrição de Arquitetura de Software em Sistemas Concentrados	Este documento recomenda uma estrutura conceitual e conteúdo para a descrição arquitetural de sistemas de software concentrados.	S	S							P	
IEEE Std 1490-1998	IEEE Guia – Adoção do Padrão PMI – Um Guia para o Projeto de Gestão Corpo de Conhecimento	Este documento é a adoção da IEEE do Projeto de Gestão do Corpo de Conhecimento definido pelo Instituto de Gerenciamento de Projetos (PMI). Ele identifica e descreve os conhecimentos geralmente aceitos sobre gestão de projetos.							P			
IEEE Std 1517-1999	Padrão IEEE para Tecnologia da Informação - Processo do Ciclo de Vida do Software – Reutilização de	Este padrão fornece processos de ciclo de vida para reutilização sistemática do software. Os processos são adequados para uso com IEEE/EIA 12207			S					P		

Número padrão	Nome padrão	Descrição	Requisitos de software	Design de software	Construção de software	Testes de software	Manutenção de Software	Gerência da Configuração de Software	Gerência de Engenharia de Software	Processo de Engenharia de Software	Ferramentas e Métodos da Engenharia de Software	Qualidade de Software
	Processos											
IEEE Std 1540-2001 //ISO/IEC 16085:2003	Padrão IEEE para Processos de Ciclo de Vida do Software – Gestão de Risco	Este padrão fornece um processo de ciclo de vida para gestão de risco. O processo é adequado para uso com IEEE/EIA 12207							S	P		
IEEE Std 2001-2002	IEEE Práticas Recomendadas para a Internet – Engenharia de Sítios Web , Gestão de Sítios Web e Ciclo de Vida de Sítios Web	Este documento recomenda práticas para engenharia de páginas www para uso em ambientes de Intranet e Extranet.										P
ISO 9001:2000	Sistemas de Gestão da Qualidade - Requisitos	Este padrão especifica os requisitos para um sistema de gestão de qualidade organizacional, objetivando fornecer requisitos conhecidos de produtos e aumentar a satisfação do cliente.								S		P
ISO/IEC 9126-1:2001	Engenharia de Software – Qualidade de Produto – Parte 1: Modelo de Qualidade	Este padrão fornece um modelo para a qualidade do produto de software em relação à qualidade interna, qualidade externa e qualidade em uso. O modelo é sob a forma de uma taxonomia de características definidas que o software pode apresentar.	P	S	S	S						
IEEE/EIA 12207.0-1996	Implementação Industrial do Padrão Internacional ISO/IEC 12207:1995, Padrão para	Este padrão fornece uma estrutura de processos usados pelo ciclo de vida completo de software	X	X	X	X	X	X	X	P	X	X

Número padrão	Nome padrão	Descrição	Requisitos de software	Design de software	Construção de software	Testes de software	Manutenção de Software	Gerência da Configuração de Software	Gerência de Engenharia de Software	Processo de Engenharia de Software	Ferramentas e Métodos da Engenharia de Software	Qualidade de Software
//ISO/IEC 12207:1995	Tecnologia de Informação – Processos de Ciclo de Vida de Software											
IEEE/EIA 12207.1-1996	Implementação Industrial do Padrão Internacional ISO/IEC 12207:1995, Padrão para Tecnologia da Informação – Processos de Ciclo de Vida de Software – Ciclo de Vida de Dados	Este documento fornece orientações sobre gravação de dados resultantes do ciclo de vida dos processos IEEE/EIA 12207.0.	X	X	X	X	X	X	X	P	X	X
IEEE/EIA 12207.2-1997	Implementação Industrial do Padrão Internacional ISO/IEC 12207:1995, Padrão de Tecnologia de Informação – Processos de Ciclo de Vida de Software – Considerações de Implementação	Este documento fornece orientações adicionais para a implementação do ciclo de vida dos processos IEEE/EIA 12207.0.	X	X	X	X	X	X	X	P	X	X
IEEE Std 14143.1-200// ISO/IEC 14143-1:1998	IEEE Adoção do ISO/IEC 14143-1:1998 – Tecnologia da Informação – Medida de Software – Medida de Tamanho Funcional – Parte 1: Definição de Conceitos	Este padrão descreve os conceitos fundamentais de uma classe de medidas conhecidas coletivamente como tamanho funcional.	P				S		S			S
ISO/IEC TR 14471:199	Tecnologia da Informação – Engenharia de Software – Direcionamentos para a Adoção de	Este documento fornece orientação no estabelecimento de processos e atividades que podem ser aplicados na adoção da tecnologia CASE.									P	

Número padrão	Nome padrão	Descrição	Requisitos de software	Design de software	Construção de software	Testes de software	Manutenção de Software	Gerência da Configuração de Software	Gerência de Engenharia de Software	Processo de Engenharia de Software	Ferramentas e Métodos da Engenharia de Software	Qualidade de Software
9	Ferramentas CASE											
ISO/IEC 14598 (Six parts	Tecnologia da Informação – Avaliação de Produto de Software	A série ISO / IEC 14598 dá uma visão geral dos processos de avaliação de produtos de software e fornece orientações e requisitos para avaliação em nível organizacional ou de projeto. Os padrões fornecem métodos de medida, estimativa e avaliação.										P
ISO/IEC 14764:1999	Tecnologia da Informação – Manutenção de Software	Este padrão desenvolve o processo de manutenção na ISO / IEC 12207. Ele fornece orientação na aplicação dos requisitos de processo.					P					
ISO/IEC 15026:1998	Tecnologia da Informação – Níveis de Integridade de Software e Sistemas	Este padrão internacional introduz o conceito de nível de integridade de software e requisitos de integridade de software. Ele define o conceito associado com níveis de integridade e requisitos de integridade de software e as exigências locais de cada processo.	S	S								P
ISO/IEC TR 15271:1998	Tecnologia da Informação – Guia para ISO/IEC 12207 (Processos de Ciclo de Vida do Software)	Este documento é um guia para uso da isso/IEC 12207.								P		
ISO/IEC 15288:2002	Engenharia de Sistemas – processos de Ciclo de Vida de Sistema	Este padrão fornece uma estrutura de processos usados em todo o ciclo de vida de sistemas feitos pelo homem.								P		
ISO/IEC	Engenharia de Software –	Este relatório técnico (agora a ser revisto como um								P		

Número padrão	Nome padrão	Descrição	Requisitos de software	Design de software	Construção de software	Testes de software	Manutenção de Software	Gerência da Configuração de Software	Gerência de Engenharia de Software	Processo de Engenharia de Software	Ferramentas e Métodos da Engenharia de Software	Qualidade de Software
Software TR 15504 (9 parts) Process and Draft IS 15504 (5 parts)	Avaliação de Processos	padrão) fornece os requisitos de métodos de avaliação de processo de execução, como base para a melhoria do processo ou da determinação capacidade										
ISO/IEC 15939:2002	Engenharia de Software – Processo de Medição de Software	Este padrão fornece um processo de ciclo de vida para medição de software. O processo é adequado ao uso com IEEE/EIA 12207.							S	P		S
ISO/IEC 19761:2003	Engenharia de Software – COSMIC–FFP - Um Método de Medição de Tamanho Funcional	Este padrão descreve o COSMIC-FFP Método de Medição de Tamanho Funcional, um método de medição de tamanho funcional de acordo com os requisitos do padrão ISO/IEC 14143-1.	P				S		S			S
ISO/IEC 20926:2003	Engenharia de Software – IFPUG 4.1 Método de Medição de Tamanho Funcional Desajustado - Manual de Práticas de Contagem	Este padrão descreve o IFPUG 4.1 contagem de ponto de função desajustado, um método de medição de tamanho funcional de acordo com os requisitos do padrão ISO/IEC 14143-1.	P				S		S			S
ISO/IEC 20968:2002	Engenharia de Software – Mk II Análise de Ponto de Função – Manual de Práticas de Contagem	Este padrão descreve o Mk II Analisador de Pontos de Função, um método de medição de tamanho funcional de acordo com os requisitos do padrão ISO/IEC 14143-1.	P				S		S			S

Número padrão	Nome padrão	Descrição	Requisitos de software	Design de software	Construção de software	Testes de software	Manutenção de Software	Gerência da Configuração de Software	Gerência de Engenharia de Software	Processo de Engenharia de Software	Ferramentas e Métodos da Engenharia de Software	Qualidade de Software
ISO/IEC 90003	Software e Engenharia de Sistemas – Orientações para a Aplicação da ISO 9001:2000 para Software de computador	Este padrão fornece orientações para organizações na aplicação do ISO 9001:2000 para a aquisição, suprimento, desenvolvimento, operação e manutenção de software de computador.								S		P

2.1.4.4 Apêndice D

2.1.4.4.1 Classificação dos Temas Segundo a Taxonomia de Bloom

Taxonomia de Bloom é uma classificação bem conhecida e amplamente utilizada para objetivos educacionais cognitivos. Foi adotado com o objetivo de ajudar o público que deseja utilizar o guia como uma ferramenta na definição de material de curso, currículos universitários, critérios universitários para credenciamento de programas, descrições de trabalho, descrições de papéis dentro de uma definição do processo de engenharia de software, caminhos de desenvolvimento profissional e programas de formação profissional entre outras necessidades. Os níveis da taxonomia de Bloom para os tópicos do guia SWEBOK são sugeridos neste apêndice para uma graduação de engenharia de software com quatro anos de experiência. A graduação de engenharia de software com quatro anos de experiência é, em essência o "alvo" do Guia SWEBOK conforme definido pelo que se entende por conhecimento geral aceito.

Uma vez que este apêndice apenas se refere ao que pode ser considerado como conhecimento "geralmente aceito", é muito importante lembrar que um engenheiro de software deve saber muito mais do que isso. Além do conhecimento "geralmente aceito", uma graduação de engenharia de software com quatro anos deve possuir alguns elementos de disciplinas relacionadas, bem como alguns elementos de conhecimento especializado, conhecimento avançado e, possivelmente, até mesmo conhecimento de pesquisa. As seguintes suposições foram feitas quando os níveis da taxonomia proposta:

- As avaliações são propostas para um engenheiro de software "generalista" e não um engenheiro de software que trabalham em um grupo especializado, como uma equipe de gerenciamento de configuração de software, por exemplo. Obviamente, como engenheiro de software, seria necessário atingir níveis taxonomia muito maiores na área de especialidade do seu grupo;
- Um engenheiro de software com quatro anos de experiência ainda está no início de sua carreira e lhe seria atribuídas relativamente poucas funções de gestão, ou pelo menos não para grandes empreendimentos. "Tópicos relacionados ao gerenciamento" não são, portanto, uma prioridade nas avaliações propostas. Pela mesma razão, os níveis de taxonomia tendem a ser menores para "temas iniciais do ciclo de vida", tais como os relacionados

com os requisitos de software do que para os tópicos mais tecnicamente orientados, tais como aqueles dentro do projeto de software, software de construção ou testes de software.

- Assim, as avaliações podem ser adaptadas para mais engenheiros de software sênior ou engenheiros de software especializados em determinadas áreas do conhecimento, nenhum tópico é dado um nível superior de Análise de taxonomia. Isto é consistente com a abordagem da Software Engineering Education Body of Knowledge (SEEK) onde a nenhum assunto é atribuído um nível superior a taxonomia Aplicação. O objetivo do SEEK é definir um programa de educação do corpo de conhecimento de engenharia de software adequado para orientar o desenvolvimento dos currículos de graduação em engenharia de software. Embora distintos, nomeadamente em termos de alcance, SEEK e SWEBOK estão intimamente relacionados.

A taxonomia de Bloom de domínio cognitivo proposto em 1956 contém seis níveis. A tabela 14 do apêndice D do SWEBOK apresenta esses níveis e palavras-chave, muitas vezes associadas a cada nível. Maiores informações sobre a Classificação de Bloom, estão inseridas no Apêndice D do SWEBOK.

Nível da Taxonomia de Bloom	Palavras-chave Associadas
Conhecimento: Retorno de dados	Define, descreve, identifica, conhece, rótulos, listas, jogos, nomes contornos, retorna, reconhece, reproduz, seleciona, afirma.
Compreensão: Entender o significado, a tradução, a interpolação, e interpretação de instruções e problemas. Estado um problema em suas próprias palavras.	Compreende, converte, defende, distingue, estima, explica, se estende, generaliza, dá exemplos, infere, interpreta, paráfrases, prevê, reescreve, resume, traduz.
Aplicação: Usa um conceito em uma situação nova ou usa espontaneamente de uma abstração. Aplica-se o que foi aprendido em sala de aula em situações	Aplica-se, muda, calcula, constrói, demonstra, descobre, manipula, modifica, opera, prevê, prepara, produz, se relaciona, shows, resolve, usa.

Nível da Taxonomia de Bloom	Palavras-chave Associadas
novas no trabalho.	
Análise: Separa o material ou conceitos em componentes de forma que sua estrutura organizacional pode ser entendida. Distingue entre fatos e inferências.	Analisa, divide, compara, contrasta, diagramas, desconstrói, diferencia, discrimina, diferencia, identifica, ilustra, infere, define, relaciona, seleciona, separa.
Síntese: Cria uma estrutura ou padrão a partir de elementos diversos. Coloca as peças para formar um todo, com ênfase na criação de um novo significado ou estrutura.	Categoriza, combina, compila, compõe, cria, inventa, projeta, explica, gera, modifica, organiza, planeja, se recicla, reconstrói, se refere, se reorganiza, revisa, reescreve, resume, diz, escreve.
Avaliação: Faz julgamentos sobre o valor das ideias ou materiais.	Avalia, compara, conclui, os contrastes, critica, critica, defende, descreve, discrimina, avalia, explica, interpreta, justifica, diz, resume, suporta.

A repartição dos assuntos nos quadros não corresponde perfeitamente repartição contida nas áreas do conhecimento. A avaliação para esse anexo foi preparado, enquanto alguns comentários ainda estavam próximos do término.

Finalmente, lembre-se que as avaliações do presente apêndice definitivamente devem ser vistos apenas como uma proposta para ser desenvolvida e validada.

Requisitos de Software

Repartição dos Tópicos	Nível de Taxonomia
1. Fundamentos de requisitos de software	
Definição de requisitos de software	C
Requisitos de produto e processo	C
Requisitos funcionais e não funcionais	C
Propriedades emergentes	C
Requisitos quantificáveis	C
Requisitos de sistema e requisitos de software	C
2. Processo de requisitos	
Modelo de processo	C
Atores do processo	C
Suporte do processo e gerenciamento	C
Qualidade do processo e aproveitamento	C
3. Requisitos de elicitação	
Origens dos requisitos	C
Técnicas de elicitação	AP
4. Análise de requisitos	
Classificação de requisitos	AP
Modelagem conceitual	AN
Design arquitetural e alocação de requisitos	AN
Negociação de requisitos	AP
5. Especificação de requisitos	
Documento de definição do sistema	C
Especificação dos requisitos do sistema	C
Especificação dos requisitos de software	AP
6. Validação de requisitos	
Análise de requisitos	AP
Prototipação	AP
Validação de modelo	C
Testes de aceitação	AP
7. Considerações práticas	
Processo de requisitos de natureza iterativa	C
Gerenciamento de mudança	AP
Atributos de requisitos	C
Rastreamento de requisitos	AP
Requisitos de medição	AP

Design de Software

Repartição dos Tópicos	Nível de Taxonomia
1. Fundamentos de Design de software	
Conceitos gerais de design	C
Contexto de design de software	C
Processo de design de software	C
Técnicas de ativação	AN
2. Questões chave no design de software	
Concorrência	AP
Controlando e carregando eventos	AP
Distribuição de componentes	AP
Erro e manejo de exceção e tolerância a falhas	AP
Apresentação e interação	AP
Persistência de dados	AP
3. Estrutura e arquitetura de software	
Estruturas arquiteturais e pontos de vista	AP
Estilos arquiteturais	AN
Padrões de design	AN
Famílias de programas e frameworks	C
4. Análise da qualidade do design de software e avaliação	
Atributos de qualidade	C
Análises que qualidade e técnicas de avaliação	AN
Medições	C
5. Notações de design de software	
Descrições Estruturais	AP
Descrições de comportamento	AP
6. Estratégias e métodos para o design de software	
Estratégias gerais	AN
Design orientado a função	AP
Design orientado a objetos	AN
Design centrado na estrutura de dados	C
Design baseado em componentes	C
Outros métodos	C

Construção de Software

Repartição dos Tópicos	Nível de Taxonomia
1. Fundamentos de Construção de software	
Minimizar complexidade	AN
Antecipação de mudança	AN
Construção para verificação	AN
Normas de construção	AP
2. Gestão de construção	
Modelos de construção	C
Planejamento de construção	AP
Medição de construção	AP
3. Considerações práticas	
Design de construção	AN
Linguagens de construção	AP
Codificando	AN
Testando a construção	AP
Qualidade da construção	AN
Integração	AP

Testes de Software

Repartição dos Tópicos	Nível de Taxonomia
1. Fundamentos de Teste de software	
Terminologia relacionada a testes	C
Assuntos chave	AP
Relações de testes com outras atividades	C
2. Níveis de teste	
Alvo de teste	AP
Objetivos de teste	AP
3. Técnicas de teste	
Baseada na intuição e na experiência do engenheiro	AP
Baseada na Especificação	AP
Código de barras	AP
Falhas baseadas	AP
Uso baseadas	AP
Baseadas na natureza da aplicação	AP
Selecionando e combinando técnicas	AP
4. Medidas Relacionadas ao Teste	
Avaliação do programa sob o teste	AN
Avaliação dos testes executados	AN
5. Processo de Teste	
Considerações práticas	C
Atividades de Teste	AP

Manutenção de Software

Repartição dos Tópicos	Nível de Taxonomia
1. Fundamentos da Manutenção de software	
Definições e tecnologia	C
Natureza da manutenção	C
Necessidade de manutenção	C
Maioria dos custos de manutenção	C
Evolução de software	C
Categorias de manutenção	AP
2. Questões chave na manutenção de software	
Questões técnicas	C
Questões gerenciais	AP
Estimativa de custos de manutenção	AN
Medição de manutenção de software	AN
3. Processo de manutenção	
Processos de manutenção	C
Atividades de manutenção	AP
4. Técnicas para manutenção	
Compreensão d programa	AN
Engenharia reversa	C

Gerência de Configuração de Software (GCS)

Repartição dos Tópicos	Nível de Taxonomia
1. Gerenciamento dos processos de GCS	
Contexto organizacional para GCS	C
Restrições e orientações para GCS	C
Planejamento para GCS	
GCS organização e responsabilidades	AP
GCS fontes e programação	AP

Repartição dos Tópicos	Nível de Taxonomia
Ferramenta de seleção e implementação	AP
Controle de Vendas/contratos	C
Controle de interface	C
Plano de gerenciamento de configuração de software	C
Inspeção de gerenciamento de configuração de software	
GCS medidas e dimensões	AP
Entrada de processos de auditorias para GCS	C
2. Identificação de configuração de software	
Identificando itens a serem controlados	
Configuração de software	AP
Itens de configuração de software	AP
Itens de relacionamento de configuração de software	AP
Versões de software	AP
Ponto de partida	AP
Itens de aquisição de configuração de software	AP
Biblioteca de software	C
3. Controle de Configuração de software	
Requisição , avaliação e aprovação de mudança de software	
Tabela de controle de configuração de software	AP
Processo de requisição de mudança de software	AP
Implementação de mudanças de softwares	AP
Desvios e concessões	C
4. Status da contabilidade de configuração de software	
Status de informações de configuração de software	C
Status de relatórios de configuração de software	AP
5. Auditoria de configuração de software	
Auditoria de configuração de software funcional	C
Auditoria de configuração de software físico	C
Auditorias em processo de uma base de software	C
6. Entrega e gerenciamento de liberação de software	
Construção de software	AP
Gerenciamento de liberação de software	C

Gerência de Engenharia de software

Repartição dos Tópicos	Nível de Taxonomia
1. Iniciação e definição de escopo	
Determinação de negociação de requisitos	AP
Análise de possibilidade	AP
Processos para requisitos de análise/revisão	C
2. Planejamento de projeto de software	
Planejamento de processos	C
Determinar resultados	AP
Estimativa de esforço, programação e custo	AP
Alocação de recursos	AP
Gerenciamento de risco	AP
Gerenciamento da qualidade	AP
Gerenciamento de projeto	C
3. Lei do projeto de software	
Implementação de projeto	AP
Gerenciamento de contratos com fornecedores	C
Implementação de processos de medição	AP
Processo de monitoramento	AN
Controle de processos	AP
Relatório	AP
4. Análise e avaliação	
Determinando a satisfação dos requisitos	AP

Repartição dos Tópicos	Nível de Taxonomia
Análise e avaliação de performance	AP
5. Fechamento	
Determinando fechamento	AP
Atividades de fechamento	AP
6. Medições de engenharia de software	
Estabelecer e manter compromisso de medição	C
Planejar o processo de medição	C
Executar o processo de medição	C
Avaliar medições c	C

Processos de Engenharia de software

Repartição dos Tópicos	Nível de Taxonomia
1. Processo de Implementação e mudança	
Processos de infraestrutura	
Grupo de processo de engenharia de software	C
Fábrica de experiência	C
Atividades	AP
Modelos para implementação de processos e mudança	K
Considerações práticas	C
2. Definição de processos	
Modelos de ciclo de vida	AP
Processos de ciclo de vida do software	C
Notações para definições de processos	C
Adaptação de processos	C
Automação	C
3. Avaliação de Processo	
Modelos de avaliação de processo	C
Métodos de avaliação de processo	C
4. Medição de produto e processo	
Medição de processo de software	AP
Medição de produto de software	AP
Medição de tamanho	AP
Medição da estrutura	AP
Medição da qualidade	AP
Resultados da medição da qualidade	AN

Repartição dos Tópicos	Nível de Taxonomia
Modelos de informação de software	
Modelo de construção	AP
Modelo de implementação	AP
Técnicas de medição	
Técnicas analíticas	AP
Técnicas de benchmarking	C

Ferramentas e métodos de Engenharia de software

Repartição dos Tópicos	Nível de Taxonomia
1. Ferramentas de software	
Ferramentas de requisitos de software	AP
Ferramentas de design de software	AP
Ferramentas de construção de software	AP
Ferramentas de testes de software	AP
Ferramentas de manutenção de software	AP
Ferramentas de processos de engenharia de software	AP
Ferramentas de qualidade de software	AP
Ferramentas de gerenciamento de configuração de software	AP
Ferramentas de gerenciamento de engenharia de software	AP
Ferramenta para assuntos diversos	AP
2. Métodos de Engenharia de software	
Métodos heurísticos	AP
Métodos formais e notações	C
Métodos de prototipação	AP
Métodos para assuntos diversos	C

Qualidade de Software

Repartição dos Tópicos	Nível de Taxonomia
1. Fundamentos de qualidade de software	
Cultura e ética em engenharia de software	AN
Valor e custo da qualidade	AN

Repartição dos Tópicos	Nível de Taxonomia
Modelos de qualidade e características	
Qualidade do processo de software	AN
Qualidade do produto de software	AN
Melhoria da qualidade	AP
2. Processos de gerenciamento da qualidade de software	
Garantia da qualidade de software	AP
Verificação e validação	AP
Análise e auditoria	
Inspeções	AP
Revisões com parceria	AP
Revisão conjunta	AP
Testes	AP
Auditorias	C
3. Considerações práticas	
Requisitos de qualidade da aplicação	
Criticidade dos sistemas	C
Fidelidade	C
Nível de integridade de software	C
Caracterização de defeitos	AP
Técnicas de gerenciamento de qualidade de software	
Técnicas estáticas	AP
Técnicas com pessoas de alto nível de conhecimento	AP
Técnicas analíticas	AP
Técnicas dinâmicas	AP
Medição de qualidade de software	AP

3 Conclusão

3.1 Apresentação dos resultados

O presente trabalho procurou fazer uma exposição da importância do Guia para o Corpo de Conhecimento de Engenharia de Software – SWEBOK. Teve como foco sua tradução e o conhecimento de temas não abordados por alguns cursos, porém tidos como importantes para a formação profissional.

As áreas de conhecimento de modo geral estão fortemente concisas. A KA de Requisitos de software no seu item de elicitação de requisitos apresenta um conjunto de regras para se obter os dados necessários ao desenvolvimento de software. A KA Design de Software mostra as várias terminologias encontradas para o design, como D-Design, PF-Design e I-Design, todas com diferentes conceitos e atuações para diferentes objetivos. Na KA Construção de Software, o princípio de reutilização de software mostra que o conceito é bem mais que criar bibliotecas de recursos, busca-se também a integração dos processos de reutilização com atividades no ciclo de vida do software. A área de conhecimento Teste de Software trouxe uma nova abordagem de testes, onde não se pensa em testes após a conclusão do produto, mas sim em todo o processo de desenvolvimento e manutenção. A KA Manutenção de Software oferece a possibilidade de se gerir um sistema e todas as realidades inerentes a ele, como aspectos de hardware, software e firmware entre outros. Temos na Ka Gerência de Engenharia de Software os mecanismos necessários para a administração de todos os pontos relativos à engenharia de software e suas especificidades. Na KA Processo de Engenharia de Software o foco está onde mudanças de processos ou de tecnologia são inicialmente introduzidas através da melhoria de processos ou produtos. A área de conhecimento Ferramentas e Métodos da Engenharia de Software fornece a descrição das ferramentas utilizadas por cada área de conhecimento assim como ferramentas para problemas variados. A área de conhecimento Qualidade de Software se torna uma das mais importantes áreas dentro do corpo de conhecimento da engenharia de software. Nela encontramos as definições para qualidade e em particular as técnicas estáticas, ou seja, os valores, conceitos e metodologias que envolvem a concepção do software.

As disciplinas relacionadas com engenharia de software cederam para esta

um acervo de conhecimento que foi fundamental para a sua existência. Através delas foram traçadas as diretrizes e o conteúdo básico (e distinto) para esta nova disciplina, que, também contribuíram, através de seus autores, com o seu rico repositório bibliográfico. Neste repositório, encontramos material de referência que é de suma importância para pesquisas científicas e estão organizadas por área de conhecimento, facilitando assim seu manuseio.

3.2 Principais contribuições

Entre as contribuições advindas deste trabalho, ressalta-se, em primeiro lugar, a tradução do SWEBOK para o português.

Também servirá de fundamentação para outros trabalhos acadêmicos e pós-graduação.

Para profissionais da área de desenvolvimento, de design, de manutenção, de construção e outras áreas afins, não só em se tratando de software, mas de um modo geral, o conhecimento dos tópicos aqui contidos representa um aumento substancial da bagagem necessária para que o profissional seja mai completo. Isto contribui ainda para que este profissional tenha uma visão mais ampla de todos estes processos e possa ser inserido em outros projetos da empresa.

Além disso, este trabalho torna-se extremamente útil como base para informar qualquer profissional interessado em conhecer a definição de cada componente do corpo de conhecimento da engenharia de software.

3.3 Aspectos positivos e Negativos

Um importante aspecto a ser expresso é que contribuir para o maior conhecimento deste guia e seus componentes foi uma atividade concluída com êxito. Destaca-se também o aprofundamento de temas de engenharia de software, o conhecimento de novos conceitos, a interligação destes conceitos com áreas de outras ciências. Isto sem dúvida foi um ponto positivo deste trabalho.

A dificuldade em se traduzir alguns termos para a nossa língua culta, talvez tenha sido um ponto a ser considerado como dificultoso. Na língua inglesa, usam-se recursos como a união de palavras cujo sentido se torna totalmente divergente das

palavras originais. Não raras são as vezes onde o melhor termo a ser empregado para tradução normalmente não tem nada haver com as palavras usadas na língua inglesa.

3.4 Futuro do Guia

Analisando o por quê de sua criação e observando os rumos que as tecnologias têm tomado no sentido da padronização, fica fácil prever que tal ferramenta, se torna imprescindível para o crescimento e estabilização dessa relativamente nova disciplina de Engenharia de Software.

Guias como PMBoK, BABOK e outros vem se tornando referência para consulta e instrumento ao qual profissionais das áreas correlatas encontram respostas. O SWEBOK, análogo em significado ao PMBoK, é tido como o PMI da Engenharia de Software e deve ser assimilado por todas as pessoas afetas a esta área, pois o SWEBOK é uma ferramenta bastante completa e está muito a frente de outras iniciativas desta amplitude.

Finalizando, o Guia para para o Corpo de Conhecimento da Engenharia de Software não deve ser negligenciado pelas instituições de ensino até mesmo por se tratar de uma ferramenta de iniciativa da IEEE que tem como uma de suas finalidades servir de referência global em assuntos considerados, de forma generalizada pela comunidade, como pertinentes a área de Engenharia de Software.

Bibliografia

- [Abr96] A. Abran and P.N. Robillard, “Function Points Analysis: An Empirical Study of its Measurement Processes,” IEEE Transactions on Software Engineering, vol. 22, 1996, pp. 895-909.
- [Bas92] V. Basili et al., “The Software Engineering Laboratory — An Operational Software Experience Factory,” presented at the International Conference on Software Engineering, 1992.
- [Bec02] K. Beck, Test-Driven Development by Example,Addison-Wesley, 2002.
- [Bec99] K. Beck, Extreme Programming Explained: Embrace Change, Addison-Wesley, 1999.
- [Bei90] B. Beizer, Software Testing Techniques,International Thomson Press, 1990, Chap. 1-3, 5, 7s4, 10s3, 11, 13.
- [Boe03] B. Boehm and R. Turner, “Using Risk to Balance Agile and Plan-Driven Methods,” Computer, June 2003, pp. 57-66.
- [Com97] E. Comer, “Alternative Software Life Cycle Models,” presented at International Conference on Software Engineering, 1997.
- [Dav93] A.M.Davis, Software Requirements: Objects, Functions and States: Prentice-Hall,1993. [Gog93] J.Goguen and C.Linde, “Techniques for Reuirements Elicitation,” presented at International Symposium on Requirements Engineering, San Diego, California, 1993.
- [Dor02] M. Dorfman and R.H. Thayer, eds., Software Engineering, IEEE Computer Society Press, 2002, Vol. 1, Chap. 6, 8, Vol. 2, Chap. 3, 4, 5, 7, 8.
- [ElE99] K. El-Emam and N. Madhavji, Elements of Software Process Assessment and Improvement, IEEE Computer Society Press, 1999.
- [Fen98] N.E. Fenton and S.L. Pfleeger, Software Metrics: A Rigorous & Practical Approach, second ed., International Thomson Computer Press, 1998, Chap. 1-14.
- [Fen98] N.E. Fenton and S.L. Pfleeger, Software Metrics: A Rigorous & Practical Approach, second ed., International Thomson Computer Press, 1998.
- [Fin94] A. Finkelstein, J. Kramer, and B. Nuseibeh, “Software Process Modeling and Technology,” Research Studies Press Ltd., 1994.
- [Fow90] P. Fowler and S. Rifkin, Software EngineeringProcess Group Guide, Software Engineering Institute, Technical Report CMU/SEI-90-TR-24, 1990
- [Gol99] D. Goldenson et al., “Empirical Studies of Software Process Assessment Methods”
- [IEEE1074-97] IEEE Std 1074-1997, IEEE Standard forDeveloping Software Life Cycle Processes, IEEE, 1997.
- [IEEE12207.0-96] IEEE/EIA 12207.0-1996//ISO/ IEC12207:1995, Industry Implementation of Int. Std ISO/IEC 12207:95
- [IEEE830-98] IEEE Std 830-1998, IEEE Recommended Practice for Software Requirements Specifications: IEEE,1998
- [ISO15504-98] ISO/IEC TR 15504:1998, Information Technology - Software Process Assessment (parts 1-9), ISO and IEC, 1998.
- [ISO15939-02] ISO/IEC 15939:2002, Software Engineering — Software Measurement Process, ISO and IEC, 2002.
- [ISO9126-01] ISO/IEC 9126-1:2001, Software Engineering - Product Quality-Part 1: Quality Model, ISO and IEC, 2001.
- [Joh99] D. Johnson and J. Brodman, “Tailoring the CMM for Small Businesses, Small Organizations, and Small Projects”
- [Jor02] P. C. Jorgensen, Software Testing: A Craftsman's Approach, second edition, CRC Press, 2004, Chap. 2, 5-10, 12-15, 17, 20.
- [Kan01] C. Kaner, J. Bach, and B. Pettichord, Lessons Learned in Software Testing, Wiley Computer Publishing,

2001.

- [Kan99] C. Kaner, J. Falk, and H.Q. Nguyen, Testing Computer Software, second ed., John Wiley & Sons, 1999, Chaps. 1, 2, 5-8, 11-13, 15.
- [Kot00] G. Kotonya and I. Sommerville, Requirements Engineering: Processes and Techniques: John Wiley and Sons, 2000.
- [Lou95] P. Loucopulos and V. Karakostas, Systems Requirements Engineering: McGraw-Hill, 1995. [Pfl01] S. L. Pfleeger
- [Lyu96] M.R. Lyu, Handbook of Software Reliability Engineering, Mc-Graw-Hill/IEEE, 1996, Chap. 2s2.2, 5-7.
- [McF96] B. McFeeley, IDEAL: A User's Guide for Software Process Improvement, Software Engineering
- [Moi98] D. Moitra, "Managing Change for Software Process Improvement Initiatives: A Practical Experience- Based Approach"
- [Mus99] J. Musa, Software Reliability Engineering: More Reliable Software, Faster Development and Testing, McGraw-Hill, 1999.
- [OMG02] Object Management Group, "Software Process Engineering Metamodel Specification," 2002, em http://www.omg.org/docs/formal/02-11-14.pdf.
- [Per95] W. Perry, Effective Methods for Software Testing,John Wiley & Sons, 1995, Chap. 1-4, 9, 10-12, 17, 19-21.
- [Pfl01] S.L. Pfleeger, Software Engineering: Theory and Practice, second ed., Prentice Hall, 2001.
- [Pre04] R.S. Pressman, Software Engineering: A Practitioner's Approach, sixth ed., McGraw-Hill, 2004.
- [Rei02] D.J. Reifer, ed., Software Management, IEEE Computer Society Press, 2002, Chap. 1-6, 7-12, 13.
- [Rob99] S. Robertson and J. Robertson, Mastering the Requirements Process: Addison-Wesley, 1999.
- [San98] M. Sanders, "The SPIRE Handbook: Better, Faster, Cheaper Software Development in Small Organisations," European Commission, 1998.
- [SEI01] Software Engineering Institute, "Capability Maturity Model Integration, v1.1," 2001, em
- [SEL96] Software Engineering Laboratory, Software Process Improvement Guidebook, NASA/GSFC,
- [Som05] I. Sommerville, "Software Engineering," Seventh ed: Addison-Wesley, 2005.
- [Som97] I. Sommerville and P. Sawyer, "Requirements engineering: A Good Practice Guide," John Wiley and Sons, 1997, Chap. 1-2.
- [Sti99] H. Stienen, "Software Process Assessment and Improvement: 5 Years of Experiences with Bootstrap.
- [Tha97] R.H. Thayer, ed., Software Engineering Project Management, IEEE Computer Society Press, 1997
- [VIM93] ISO VIM, International Vocabulary of Basic and General Terms in Metrology, ISO, 1993.
- [You01] R. R. You, Effective Requirements Practices: Addison-Wesley, 2001.
- [Zhu97] H. Zhu, P.A.V. Hall and J.H.R. May, "Software Unit Test Coverage and Adequacy"
- (Har98) D. Harel and M. Politi, Modeling Reactive Systems with Statecharts: The Statemate Approach, McGraw-Hill, 1998.
- (Hen99) S.M. Henry and K.T. Stevens, "Using Belbin's Leadership Role to Improve Team Effectiveness: An Empirical Investigation," Journal of Systems and Software, vol. 44, 1999, pp. 241-250.
- (Her98) J. Herbsleb, "Hard Problems and Hard Science: On the Practical Limits of Experimentation," IEEE TCSE
- (Hoh99) L. Hohmann, "Coaching the Rookie Manager," IEEE Software, January/February 1999, pp. 16-19.
- (Hsi96) P. Hsia, "Making Software Development Visible," IEEE Software, March 1996, pp. 23-26.
- (Hum95) W. Humphrey, A Discipline for Software Engineering, Addison-Wesley, 1995.
- (Hum97) W.S. Humphrey, Managing Technical People: Innovation, Teamwork, and the Software Process: Addison-Wesley, 1997.
- (Hum99) W. Humphrey, An Introduction to the Team Software Process, Addison-Wesley, 1999.
- (Hut94) D. Hutton, The Change Agent's Handbook: A Survival Guide for Quality Improvement Champions, Irwin, 1994.
- (IEEE1044-93) IEEE Std 1044-1993 (R2002), IEEE Standard for the Classification of Software Anomalies, IEEE, 1993.
- (IEEE1061-98) IEEE Std 1061-1998, IEEE Standard for a Software Quality Metrics Methodology, IEEE, 1998.
- (IEEE1074-97) IEEE Std 1074-1997, IEEE Standard for Developing Software Life Cycle Processes, IEEE, 1997.

- (IEEE1219-98) IEEE Std 1219-1998, IEEE Standard for Software Maintenance, IEEE, 1998.
- (IEEE1220-98) IEEE Std 1220-1998, IEEE Standard for the Application and Management of the Systems Engineering Process, IEEE, 1998.
- (IEEE12207.0-96) IEEE/EIA 12207.0-1996//ISO/IEC12207:1995, Industry Implementation of Int. Std. ISO/IEC 12207:95, Standard for Information Technology-Software Life Cycle Processes, IEEE, 1996.
- (IEEE12207.1-96) IEEE/EIA 12207.1-1996, Industry Implementation of Int. Std ISO/IEC 12207:95, Standard for Information Technology-Software Life Cycle Processes -Life Cycle Data, IEEE, 1996.
- (IEEE12207.2-97) IEEE/EIA 12207.2-1997, Industry Implementation of Int. Std ISO/IEC 12207:95, Standard for Information Technology-Software Life Cycle Processes -Implementation Considerations, IEEE, 1997.
- (IEEE14143.1-00) IEEE Std 14143.1-2000//ISO/IEC14143-1:1998, Information Technology-Software Measurement-Functional Size Measurement-Part 1: Definitions of Concepts, IEEE, 2000.
- (IEEE1517-99) IEEE Std 1517-1999, IEEE Standard for Information Technology-Software Life Cycle Processes-Reuse Processes, IEEE, 1999.
- (IEEE1540-01) IEEE Std 1540-2001//ISO/IEC16085:2003, IEEE Standard for Software Life Cycle Processes-Risk Management, IEEE, 2001.
- (IEEE610.12-90) IEEE Std 610.12-1990 (R2002), IEEE Standard Glossary of Software Engineering Terminology, IEEE, 1990.
- (ISO14674-99) ISO/IEC 14674:1999, Information Technology - Software Maintenance, ISO and IEC, 1999.
- (ISO15288-02) ISO/IEC 15288:2002, Systems Engineering-System Life Cycle Process, ISO and IEC, 2002.
- (ISO15504-98) ISO/IEC TR 15504:1998, Information Technology - Software Process Assessment (parts 1-9), ISO and IEC, 1998.
- (ISO15939-02) ISO/IEC 15939:2002, Software Engineering-Software Measurement Process, ISO and IEC, 2002.
- (ISO19761-03) ISO/IEC 19761:2003, Software Engineering-Cosmic FPP-A Functional Size Measurement Method, ISO and IEC, 2003.
- (ISO20926-03) ISO/IEC 20926:2003, Software Engineering-IFPUG 4.1 Unadjusted Functional Size Measurement Method-Counting Practices Manual, ISO and IEC, 2003.
- (ISO20968-02) ISO/IEC 20968:2002, Software Engineering-MK II Function Point Analysis – Counting Practices Manual, ISO and IEC, 2002.
- (ISO90003-04) ISO/IEC 90003:2004, Software and Systems Engineering - Guidelines for the Application of ISO9001:2000 to Computer Software, ISO and IEC, 2004.
- (ISO9001-00) ISO 9001:2000, Quality Management Systems-Requirements, ISO, 1994.
- (ISO9126-01) ISO/IEC 9126-1:2001, Software Engineering-Product Quality-Part 1: Quality Model, ISO and IEC, 2001.
- (Jac98) M. Jackman, “Homeopathic Remedies for Team Toxicity,” IEEE Software, July/August 1998, pp. 43-45.
- (Kan02) S.H. Kan, Metrics and Models in Software Quality Engineering, second ed., Addison-Wesley, 2002.
- (Kan97) K. Kansala, “Integrating Risk Assessment with Cost Estimation,” IEEE Software, May/June 1997, pp. 61-67.
- (Kar96) D.W. Karolak, Software Engineering Risk Management, IEEE Computer Society Press, 1996.
- (Kar97) J. Karlsson and K. Ryan, “A Cost-Value Aproach for Prioritizing Requirements,” IEEE Software, September/October 1997, pp. 87-74.
- (Kau99) K. Kautz, “Making Sense of Measurement for Small Organizations,” IEEE Software, March/April 1999, pp. 14-20.
- (Kei98) M. Keil et al., “A Framework for Identifying Software Project Risks,” Communications of the ACM, vol. 41, iss. 11, 1998, pp. 76-83.
- (Kel98) M. Kellner et al., “Process Guides: Effective Guidance for Process Participants,” presented at the 5th International Conference on the Software Process, 1998.
- (Ker99) B. Kernighan and R. Pike, “Finding Performance Improvements,” IEEE Software, March/April 1999, pp. 61-65.
- (Kit97) B. Kitchenham and S. Linkman, “Estimates, Uncertainty, and Risk,” IEEE Software, May/June 1997, pp. 69-74.
- (Kit98) B. Kitchenham, “Selecting Projects for Technology Evaluation,” IEEE TCSE Software Process Newsletter,
- (Kra99) H. Krasner, “The Payoff for Software Process Improvement: What It Is and How to Get It,” presented at

Elements of Software Process Assessment and Improvement, 1999.

- (Kri99) M.S. Krishnan and M. Kellner, “Measuring Process Consistency: Implications for Reducing Software Defects,” IEEE Transactions on Software Engineering, vol. 25, iss. 6, November/December 1999, pp. 800-815.
- (Lat98) F. v. Latum et al., “Adopting GQM-Based Measurement in an Industrial Environment,” IEEE Software, January-February 1998, pp. 78-86.
- (Leu96) H.K.N. Leung, “A Risk Index for Software Producers,” Software Maintenance: Research and Practice, vol. 8, 1996, pp. 281-294.
- (Lis97) T. Lister, “Risk Management Is Project Management for Adults,” IEEE Software, May/June 1997, pp. 20-22.
- (Lyu96) M.R. Lyu, Handbook of Software Reliability Engineering, Mc-Graw-Hill/IEEE, 1996.
- (Mac96) K. Mackey, “Why Bad Things Happen to Good Projects,” IEEE Software, May 1996, pp. 27-32.
- (Mac98) K. Mackey, “Beyond Dilbert: Creating Cultures that Work,” IEEE Software, January/February 1998, pp. 48-49.
- (Mad94) N. Madhavji et al., “Elicit: A Method for Eliciting Process Models,” presented at Proceedings of the Third International Conference on the Software Process, 1994.
- (Mad97) R.J. Madachy, “Heuristic Risk Assessment Using Cost Factors,” IEEE Software, May/June 1997, pp. 51-59.
- (Mas95) S. Masters and C. Bothwell, “CMM Appraisal Framework - Version 1.0,” Software Engineering Institute
- (McC96) S.C. McConnell, Rapid Development: Taming Wild Software Schedules, Microsoft Press, 1996.
- (McC97) S.C. McConnell, Software Project Survival Guide, Microsoft Press, 1997.
- (McC99) S.C. McConnell, “Software Engineering Principles,” IEEE Software, March/April 1999, pp. 6-8. (Moy97) T. Moynihan, “How Experienced Project Managers Assess Risk,” IEEE Software, May/June 1997, pp. 35-41.
- (McG01) J. McGarry et al., Practical Software Measurement: Objective Information for Decision Makers, Addison-Wesley, 2001.
- (Mcg93) C. McGowan and S. Bohner, “Model Based Process Assessments,” presented at International Conference on Software Engineering, 1993.
- (McG94) F. McGarry et al., “Software Process Improvement in the NASA Software Engineering Laboratory,” Software Engineering Institute CMU/SEI-94- TR-22, 1994, em http://www.sei.cmu.edu/pub/documents/94.reports/pdf/tr22.94.pdf.
- (Nak91) T. Nakajo and H. Kume, “A Case History Analysis of Software Error Cause-Effect Relationship,” IEEE Transactions on Software Engineering, vol. 17, iss. 8, 1991.
- (Ncs98) P. Ncsi, “Managing OO Projects Better,” IEEE Software, July/August 1998, pp. 50-60.
- (Nol99) A.J. Nolan, “Learning From Success,” IEEE Software, January/February 1999, pp. 97-105.
- (Off97) R.J. Offen and R. Jeffery, “Establishing Software Measurement Programs,” IEEE Software, March/April 1997, pp. 45-53.
- (Par96) K.V.C. Parris, “Implementing Accountability,” IEEE Software, July/August 1996, pp. 83-93.
- (Pau94) M. Paulk and M. Konrad, “Measuring Process Capability Versus Organizational Process Maturity,” presented at 4th International Conference on Software Quality, 1994.
- (Pfl01) S.L. Pfleeger, Software Engineering: Theory and Practice, second ed., Prentice Hall, 2001.
- (Pfl97) S.L. Pfleeger, “Assessing Measurement (Guest Editor’s Introduction),” IEEE Software, March/April 1997, pp. 25-26.
- (Pfl97a) S.L. Pfleeger et al., “Status Report on Software Measurement,” IEEE Software, March/April 1997, pp. 33-43.
- (Pfl99) S.L. Pfleeger, “Understanding and Improving Technology Transfer in Software Engineering,” Journal of Systems and Software, vol. 47, 1999, pp. 111-124.
- (PMI00) Project Management Institute Standards Committee, A Guide to the Project Management Body of Knowledge (PMBOK), Project Management Institute, 2000.
- (Put97) L.H. Putman and W. Myers, Industrial Strength Software — Effective Management Using Measurement, IEEE Computer Society Press, 1997.
- (Rad85) R. Radice et al., “A Programming Process Architecture,” IBM Systems Journal, vol. 24, iss. 2, 1985, pp. 79-90. (Rag89) S. Raghavan and D. Chand, “Diffusing Software-Engineering Methods,” IEEE Software, July 1989, pp. 81-90. (Rog83) E. Rogers, Diffusion of Innovations, Free Press, 1983.

- (Rob99) P.N. Robillard, “The Role of Knowledge in Software Development,” Communications of the ACM, vol. 42, iss. 1, 1999, pp. 87-92.
- (Rod97) A.G. Rodrigues and T.M. Williams, “System Dynamics in Software Project Management: Towards the Development of a Formal Integrated Framework,” European Journal of Information Systems, vol. 6, 1997, pp. 51-66.
- (Rop97) J. Ropponen and K. Lyytinen, “Can Software Risk Management Improve System Development: An Exploratory Study,” European Journal of Information Systems, vol. 6, 1997, pp. 41-50.
- (Sch99) E. Schein, Process Consultation Revisited: Building the Helping Relationship, Addison-Wesley, 1999.
- (Sco92) R.L. v. Scoy, “Software Development Risk: Opportunity, Not Problem,” Software Engineering Institute, Carnegie Mellon University CMU/SEI-92-TR-30, 1992.
- (SEI95) Software Engineering Institute, The Capability Maturity Model: Guidelines for Improving the Software Process, Addison-Wesley, 1995.
- (SEL96) Software Engineering Laboratory, Software Process Improvement Guidebook, Software Engineering Laboratory, NASA/GSFC, Technical Report SEL-95-102, April 1996, em http://sel.gsfc.nasa.gov/website/
- (Sla98) S.A. Slaughter, D.E. Harter, and M.S. Krishnan, “Evaluating the Cost of Software Quality,” Communications of the ACM, vol. 41, iss. 8, 1998, pp. 67-73.
- (Sol98) R. v. Solingen, R. Berghout, and F. v. Latum, “Interrupts: Just a Minute Never Is,” IEEE Software, September/October 1998, pp. 97-103.
- (Som97) I. Sommerville and P. Sawyer, Requirements Engineering: A Good Practice Guide, John Wiley & Sons, 1997.
- (SPC92) Software Productivity Consortium, Process Definition and Modeling Guidebook, Software Productivity Consortium, SPC-92041-CMC, 1992.
- (VIM93) ISO VIM, International Vocabulary of Basic and General Terms in Metrology, ISO, 1993.
- (Vot93) L. Votta, “Does Every Inspection Need a Meeting?” ACM Software Engineering Notes, vol. 18, iss. 5, 1993, pp. 107-114.
- (Wei93) G.M. Weinberg, “Quality Software Management,” First-Order Measurement (Ch. 8, Measuring Cost and Value), vol. 2, 1993.
- (Whi95) N. Whitten, Managing Software Development Projects: Formulas for Success, Wiley, 1995.
- (Wil99) B. Wiley, Essential System Requirements: A Practical Guide to Event-Driven Methods, Addison- Wesley, 1999.
- (Yu94) E. Yu and J. Mylopolous, “Understanding ‘Why’ in Software Process Modeling, Analysis, and Design,” presented at 16th International Conference on Software Engineering, 1994
- (Zah98) S. Zahran, Software Process Improvement: Practical Guidelines for Business Success, Addison-Wesley, 1998.
- (Zel98) M. V. Zelkowitz and D. R. Wallace, “Experimental Models for Validating Technology,” Computer, vol. 31, iss. 5, 1998, pp. 23-31.
- CMU/SEI-TR-95-001, 1995, em http://www.sei.cmu.edu/pub/documents/95.reports/pdf/tr001.95.pdf.
- http:/ /emec.mec.gov.br/ em 7 de janeiro de 2011
- Http:/ /pt.wikipedia.org/wiki/Friedrich_Ludwig_Bauer em 11/01/2011
- Http:/ /wapedia.mobi/pt/engenharia_de_software em 11/01/11 colocar referência
- Institute CMU/SEI-96-HB-001, 1996, em http://www.sei.cmu.edu/pub/documents/96.reports/pdf/hb001.96.pdf.
- Method Description,” Software Engineering Institute CMU/SEI-96-TR-007, 1996, em http://www.sei.org/eoc/G47/index.html.
- Software Process Newsletter, vol. 11, 1998, pp. 18-21, em http://www.seg.iit.nrc.ca/SPN.
- Technical Report SEL-95-102, April 1996, em http://sel.gsfc.nasa.gov/website/documents/online-doc/95-102.pdf.