MINISTÉRIO DA CIÊNCIA, TECNOLOGIA, INOVAÇÕES E COMUNICAÇÕES
INSTITUTO NACIONAL DE PESQUISAS ESPACIAIS

sid.inpe.br/mtc-m21c/2018/11.01.15.50-TDI

PROPOSTA DE UM MÉTODO DE ESTIMATIVA DE ESFORÇO PARA PROJETOS DE DESENVOLVIMENTO DE SOFTWARE CRÍTICOS

Lucas Pereira dos Santos

Dissertação de Mestrado do Curso de Pós-Graduação em Engenharia e Tecnologia Espaciais/Engenharia e Gerenciamento de Sistemas Espaciais, orientada pelo Dr. Maurício Gonçalves Vieira Ferreira, aprovada em 05 de dezembro de 2018.

URL do documento original:
<http://urlib.net/8JMKD3MGP3W34R/3S5RGU5>

INPE
São José dos Campos
2019

PUBLICADO POR:

Instituto Nacional de Pesquisas Espaciais - INPE
Gabinete do Diretor (GBDIR)
Serviço de Informação e Documentação (SESID)
CEP 12.227-010
São José dos Campos - SP - Brasil
Tel.:(012) 3208-6923/7348
E-mail: pubtc@inpe.br

MINISTÉRIO DA CIÊNCIA, TECNOLOGIA, INOVAÇÕES E COMUNICAÇÕES
INSTITUTO NACIONAL DE PESQUISAS ESPACIAIS

sid.inpe.br/mtc-m21c/2018/11.01.15.50-TDI

PROPOSTA DE UM MÉTODO DE ESTIMATIVA DE ESFORÇO PARA PROJETOS DE DESENVOLVIMENTO DE SOFTWARE CRÍTICOS

Lucas Pereira dos Santos

Dissertação de Mestrado do Curso de Pós-Graduação em Engenharia e Tecnologia Espaciais/Engenharia e Gerenciamento de Sistemas Espaciais, orientada pelo Dr. Maurício Gonçalves Vieira Ferreira, aprovada em 05 de dezembro de 2018.

URL do documento original:
<http://urlib.net/8JMKD3MGP3W34R/3S5RGU5>

INPE
São José dos Campos
2019

Dados Internacionais de Catalogação na Publicação (CIP)

Santos, Lucas Pereira dos.

Sa59p Proposta de um método de estimativa de esforço para projetos de desenvolvimento de software críticos / Lucas Pereira dos Santos. – São José dos Campos : INPE, 2019.

xxii + 185 p. ; (sid.inpe.br/mtc-m21c/2018/11.01.15.50-TDI)

Dissertação (Mestrado em Engenharia e Tecnologia Espaciais/Engenharia e Gerenciamento de Sistemas Espaciais) – Instituto Nacional de Pesquisas Espaciais, São José dos Campos, 2018.

Orientador : Dr. Maurício Gonçalves Vieira Ferreira.

1. Desenvolvimento de software. 2. Estimativa de esforço. 3. Software crítico para segurança. 4. Software embarcado. 5. COCOMO. 6. DO-178C. I.Título.

CDU 629.7:004.42

Aluno (a): *Lucas Pereira dos Santos*

Título: "PROPOSTA DE UM MÉTODO DE ESTIMATIVA DE ESFORÇO PARA PROJETOS DE DESENVOLVIMENTO DE SOFTWARE CRÍTICOS"

Aprovado (a) pela Banca Examinadora em cumprimento ao requisito exigido para obtenção do Título de *Mestre* em

Engenharia e Tecnologia Espaciais/Eng. Gerenc. de Sistemas Espaciais

Dr. Nilson Sant'Anna

Presidente / INPE / SJCampos - SP

( ) Participação por Video - Conferência

(X) Aprovado *( ) Reprovado*

Dr. Maurício Gonçalves Vieira Ferreira

Orientador(a) / INPE / SJCampos - SP

( ) Participação por Video - Conferência

( ) Aprovado *( ) Reprovado*

Dr. Edson Alves Ribeiro

Membro da Banca / INPE / São José dos Campos - SP

( ) Participação por Video - Conferência

(X) Aprovado *( ) Reprovado*

Dr. Rodrigo Rocha Silva

Convidado(a) / FATEC-MC/CISUC / Mogi das Cruzes - SP

( ) Participação por Video - Conferência

(X) Aprovado *( ) Reprovado*

Este trabalho foi aprovado por:

( ) maioria simples

(X) unanimidade

São José dos Campos, 05 de dezembro de 2018

Aos professores e orientadores do curso de Pós-Graduação do INPE.

RESUMO

Encontrar a causa raiz para as diversas falhas e atrasos em projetos de desenvolvimento de software tem sido objeto de estudo de diversos pesquisadores nos últimos anos. Estimar corretamente os custos e prazos do projeto antes de iniciá-lo é uma etapa importante do processo de desenvolvimento que não deve ser ignorada. Essa dissertação apresenta um modelo de estimativa de esforço para projetos de desenvolvimento de software críticos, baseado na adaptação do modelo COCOMO II de modo a considerar os aspectos de segurança definidos pela norma RTCA DO-178C. O desenvolvimento deste modelo inclui o estudo da literatura e dos conceitos de estimativa de esforço em projetos de desenvolvimento de software em geral, em especial o COCOMO II; a apresentação da norma DO-178C, seus processos, características e aplicação; o estudo de caso da aplicação do COCOMO II em um projeto real executado pela indústria aeronáutica brasileira seguindo e cumprindo com todos os objetivos da DO-178C, analisando o esforço real demandado por tal projeto em cada uma de suas fases; a aplicação do modelo proposto neste mesmo projeto, comparando os resultados com relação aos obtidos pela estimativa realizada com o COCOMO II; a aplicação do modelo proposto no projeto de missão espacial denominado FireSat. Com relação a comparação com o COCOMO II, a aplicação do modelo proposto apresentou uma melhor precisão na estimativa de esforço, com um erro relativo de 10,33% contra 31,3% obtidos pelo COCOMO II. Uma melhor performance do modelo proposto também foi observada na estimativa do prazo para desenvolvimento do projeto. No entanto, na estimativa do tamanho médio da equipe o modelo COCOMO II se mostrou mais aderente que o modelo proposto por esta dissertação. Na comparação com a estimativa realizada para o projeto FireSat, o modelo proposto apresentou resultados coerentes com o estimado originalmente. Deste modo, conclui-se que o modelo proposto é mais indicado para estimativa de esforço em projetos de desenvolvimento de software crítico que seguem os processos estabelecidos pela DO-178C, com espaço para se tornar ainda mais preciso mediante realização dos trabalhos futuros recomendados.

Palavras-chave: Desenvolvimento de Software. Estimativa de Esforço. Software Crítico para Segurança. Software Embarcado. COCOMO. DO-178C.

AN EFFORT ESTIMATION MODEL FOR SAFETY-CRITICAL SOFTWARE DEVELOPMENT PROJECTS

ABSTRACT

Finding the root cause for the various failures and delays in software development projects has been the subject of study by several researchers in recent years. Properly estimating project costs and deadlines before starting it is an important step in the development process that should not be ignored. This work presents an effort estimation model for safety-critical software development projects, based on the adaptation of the COCOMO II model in order to consider the safety aspects defined by the standard RTCA DO-178C. The development of this model includes the literature review and the study of the software effort estimation concepts, focused mainly on the COCOMO II model; the introduction of the DO-178C standard, its processes, characteristics and application; the case study of an application of COCOMO II in a real project executed by the Brazilian aeronautical industry, which has followed and met all the objectives of the DO-178C, analyzing the real effort demanded by such project and comparing it to the COCOMO II estimated values; the application of the proposed model in this same project, comparing the results with those obtained by the estimation with COCOMO II and the application of the proposed model in the space mission project called FireSat. Compared with COCOMO II, the application of the proposed model presented a better accuracy in the effort estimation, with a relative error of 10.33% versus 31.3% obtained by COCOMO II. A better performance of the proposed model was also observed in the schedule estimation of the project. However, in the team size estimation, the COCOMO II model proved to be more adherent than the model proposed by this work. In comparison with the estimation made for the FireSat project, the proposed model presented results consistent with the original estimation. Thus, it is concluded that the proposed model is more suitable for safety-critical software effort estimation development projects that follow the processes established by the DO-178C, with room to become even more precise if further research is carried out following the recommendations presented in this work.

Keywords: Software Development. Effort Estimation. Safety-Critical Software. Embedded Software. COCOMO. DO-178C.

LISTA DE FIGURAS

Pág.

LISTA DE TABELAS

Pág.

LISTA DE SIGLAS E ABREVIATURAS

ARP	Aerospace Recommended Practice
CMM	Capability Maturity Model
COCOMO	Constructive Cost Model
DO	Document
EADS	European Aeronautic Defense and Space Company
FAA	Federal Aviation Administration
FBW	Fly-By-Wire
IEC	International Electrotechnical Commission
INPE	Instituto Nacional de Pesquisas Espaciais
ISO	International Organization for Standardization
LOC	Lines of Code
MBD	Model Based Development
MRE	Erro Médio Relativo
NASA	National Aeronautics and Space Administration
PM	Pessoa-Mês
PMBOK	Project Management Body of Knowledge
PMI	Project Management Institue
RBT	Requirements-Based Test
RSC	Reusable Software Component
RTCA	Radio Technical Commission for Aeronautics
SAE	Society of Automotive Engineers
SID	Serviço de Informação e Documentação
SMAD	Space Mission Analysis and Design
SSA	System Safety Assessment
SW	Software
SWEBOK	Software Engineering Body of Knowledge
TDEV	Tempo de Desenvolvimento
WCET	Worst Case Execution Time

SUMÁRIO

Pág.

1 INTRODUÇÃO

A indústria de software é sem dúvida uma das que vêm introduzindo as maiores inovações do mundo moderno, conforme relatado pela World Intellectual Property Organization (2016). Desde a criação do conceito dos *smartphones,* um leque de oportunidades se abriu para os empreendedores e desenvolvedores de software em geral (CAPALDO et al., 2003). A forma de se pedir um lanche, chamar um taxi, pegar carona no carro de um estranho ou até mesmo de se conseguir um encontro mudou completamente nos últimos anos. Um dos grandes responsáveis por essa grande evolução é a indústria de desenvolvimento de software (CHESBROUGH; VANHAVERBEKE; WEST, 2006). Contudo, tamanha inovação também requer alguns cuidados adicionais com a responsabilidade que o produto de software passa a ter em nosso dia a dia. O exemplo de se chamar um transporte via aplicativo de celular, em breve, será envolvido por uma série de cuidados com a segurança, uma vez que o carro chegará até sua casa sem que haja ninguém o conduzindo, algo proporcionado pela tecnologia de navegação autônoma (SCARAMUZZA et al., 2014). Tais cuidados com a segurança já são realidade na indústria de desenvolvimento de software para aplicações conhecidas como *safety critical*, isto é, software críticos para segurança humana ou que envolvam alto valor de investimento (PRIES; QUIGLEY, 2011).

Na aviação, por exemplo, o controle de voo de aviões com tecnologia *Fly-By-Wire* é de responsabilidade total do software embarcado que gerencia todos os atuadores e comandos do piloto. Antigamente, o maior desafio no desenvolvimento de um avião era em torno de questões aerodinâmicas, atualmente o software é o item de maior preocupação e, em alguns casos, responsável por grandes atrasos no lançamento de um novo produto (KUZNECOVA, 2017).

O envio de um satélite para o espaço, seu controle de órbita, a aquisição e o envio dos dados para a estação terrena também são tarefas de responsabilidade do software embarcado (NGUYEN et al., 2008).

Seja no exemplo da aviação ou no do envio de satélite para o espaço, a segurança do software em questão é motivo de muita pesquisa e horas de engenharia gastas em desenvolvimento e aperfeiçoamento de tais produtos. Para se atingir tais requisitos de segurança e confiabilidade, dentro do custo e prazo estipulado, é preciso que o projeto tenha um bom gerenciamento e uma estimativa de esforço precisa (PMI, 2008; THE STANDISH GROUP, 2012). A característica e desafios do desenvolvimento de software crítico torna sua estimativa de esforço algo ainda mais desafiador (JØRGENSEN, 2014a).

Apesar da constante melhoria das práticas de gerenciamento de projetos, o relatório produzido pela consultoria *The Standish Group* (1995) e divulgado em 1995 com o título de *The Chaos Report* relatou que apenas 16,2% dos projetos de desenvolvimento de software são finalizados dentro do prazo e custo estipulados. Em empresas grandes, esse número é ainda pior, apenas 9% dos projetos terminam dentro do custo e prazo definidos. Analisando apenas o prazo, a média divulgada pelo relatório é de que os projetos consomem 222% do tempo original estipulado (THE STANDISH GROUP, 1995).

Nos dias de hoje, qualquer leitor pode questionar tais números, uma vez que em 1995 ainda estávamos entrando no que se conhece hoje como a era do conhecimento. A internet ainda não tinha o alcance de hoje e as práticas de gerenciamento de projetos e novas tecnologias estavam apenas começando. Realmente, na versão mais recente do relatório da mesma consultoria, intitulado como The CHAOS Manifesto (2012), é reportada uma evolução com relação aos dados do passado, sendo 37% de todos os projetos pesquisados terminaram dentro do prazo e custo estipulados, 42% sofreram algum tipo de desafio para finalizarem e apenas 21% foram cancelados ou considerados fracassados.

A Tabela 1.1 exibe uma comparação dos principais números entre 1995 e 2012. Os projetos estudados foram classificados em três grupos: (1) projetos de “Sucesso”, que foram finalizados dentro do prazo e orçamento estimados e com todo escopo especificado; (2) projetos “Desafiadores”, que foram finalizados, porém acima do orçamento, entregues em atraso e com menos

funções das que foram especificadas; e (3) projetos “Fracassados”, que foram cancelados ao longo do desenvolvimento.

Tabela 1.1 – CHAOS Report de 1995 para CHAOS Manifesto de 2012

	CHAOS Report 1995	CHAOS Manifesto 2012
Sucesso	16,2%	37%
Desafiadores	52,7%	42%
Fracassados	31,1%	21%

Fonte: The Standish Group (1995, 2012).

Os principais fatores para a evolução apresentada pelo relatório de 2012 envolvem:

- Suporte do nível executivo
- Envolvimento do usuário
- Objetivos claros de negócio
- Maturidade Emocional
- Otimizações
- Métodos Ágeis
- Experiência em Gerenciamento de Projetos
- Novas ferramentas e infraestrutura

Ainda assim, apesar da evolução apresentada nos quase 20 anos entre um relatório e outro, os números são bem aquém daqueles que seriam desejáveis pela indústria em geral.

1.1 Problema

Os números apresentados pela consultoria *Standish Group* ainda podem ser questionáveis, em função da consultoria não divulgar os dados dos projetos pesquisados, tampouco se esta segue ou não métodos científicos.

Deste modo, Moløkken e Jørgensen (2003) em sua pesquisa comparam seus resultados obtidos contra os números apresentados pelo CHAOS Report (1995) e, mesmo não mostrando um cenário tão ruim, também concluem que em média os projetos estouram o orçamento em cerca de 30%-40%, contra

89% reportados pelo *CHAOS Report* (1995). Números melhores, porém ainda desanimadores.

Neste contexto é que a estimativa de esforço em projetos de desenvolvimento de software se apresenta como alternativa de redução de tamanhos atrasos e prejuízos, por oferecer métodos para se prever o custo, prazo e esforço necessários para se atingir os resultados desejados.

Estimativa de esforço em projetos de desenvolvimento de software é um importante tópico da academia e da indústria e passou a ser mais estudado a partir dos anos 70, quando Putnam (1978) publicou seu estudo que posteriormente ficou conhecido como o Modelo Putnam. Desde então, vários outros métodos e modelos foram desenvolvidos (MOLOKKEN; JØRGENSEN, 2003; JØRGENSEN; SHEPPERD, 2007; KHATIBI; JAWAWI, 2011), com o objetivo de aprimorar as estimativas tornando cada vez mais preciso se avaliar o esforço necessário para um projeto de desenvolvimento de software. Boehm et al. (2000) apresentam um bom resumo sobre os principais métodos conhecidos atualmente.

Estimativa de esforço em projetos de desenvolvimento de software também é uma disciplina fundamental dentro da engenharia de software (SOMMERVILLE, 2007; PRESSMAN, 2009). No caso específico de software críticos, a complexidade e importância de uma boa estimativa de esforço do projeto de desenvolvimento se faz ainda mais necessária. Estimativas super otimistas podem comprometer o desenvolvimento e processo de testes, colocando a equipe com a sensação de atraso e sob pressão. Por outro lado, estimativas muito pessimistas podem resultar em custos não competitivos, inviabilizando o projeto (DE BARCELOS TRONTO; DA SILVA; SANT'ANNA, 2008). Por serem software caros, com maior envolvimento de mão de obra especializada, eventuais atrasos no prazo ou aumento de custo pode gerar milhões de dólares em prejuízo. Além disso, por se tratar de produtos de software que podem expor a segurança de pessoas, o processo de desenvolvimento geralmente é regulamentado por normas conceituadas e respeitadas pela indústria. Nesse cenário é que se apresentam as normas DO-

178C (RTCA INC, 2011a), um regulamento amplamente utilizado pelo setor aeronáutico (YAN, 2009), a NASA-STD-8719.13C, documento normativo utilizado para projetos de software da National Aeronautics and Space Administration (NASA, 2013), e a ISO/IEC62304 da British Standards Institution (BSI, 2006) utilizada para o desenvolvimento de software em dispositivos médicos.

Na literatura, as referências a software embarcado são vastas, bem como referências ao modelo COCOMO ou outros métodos de estimativa de custo (JØRGENSEN; SHEPPERD, 2007). Porém, poucas direcionam seus estudos ou atividades de forma a aplicar tais métodos de estimativa para software crítico. Por essas razões, faz-se necessário o emprego de uma abordagem de estimativa de esforço em desenvolvimento de software que leve em consideração as características e especificidades do software crítico.

Neste trabalho apresenta-se uma revisão bibliográfica específica sobre os métodos de estimativa de esforço em geral, o conceito de software crítico, o processo de desenvolvimento de software crítico regulamento pela DO-178, o estudo de caso de um projeto utilizando os conceitos do modelo COCOMO II (BOEHM et al., 2000) e da norma DO-178C (RTCA INC, 2011a) e, por fim, uma proposta de um modelo de estimativa de esforço para projetos de desenvolvimento de software críticos.

1.2 Motivação

A indústria aeroespacial opera hoje equipamentos com alto grau de investimento, seja do setor privado ou governamental, além de demandarem grande quantidade de recursos também podem expor a vida humana em perigo. Deste modo, regulamentações, certificações e padrões de qualidade para software embarcados em tais equipamentos é uma preocupação crescente da indústria e agências regulamentadoras.

Dentre os diversos modelos e métodos de estimativa de esforço em projetos de desenvolvimento de software pesquisados por este autor, bem como artigos relacionados, os trabalhos apontam para esforços na tentativa de aumentar a

precisão dos métodos atuais ou mesmo no uso em conjunto de dois os mais métodos, visando ainda o aumento da performance.

Diante desse contexto, torna-se importante o estudo de técnicas de estimativa de esforço em projetos de desenvolvimento de software críticos, que seguem o mais alto padrão de qualidade e normas regulamentadoras, de modo a contribuir com o sucesso de tais projetos, bem como do cumprimento do prazo e custo estabelecidos.

1.3 Objetivo

O objetivo desta dissertação é propor um método de estimativa de esforço para projetos de desenvolvimento de software críticos, baseado na adaptação do modelo COCOMO II (BOEHM et al., 2000) de modo a considerar os aspectos de segurança definidos pela norma DO-178C (RTCA INC, 2011a).

1.4 Metodologia

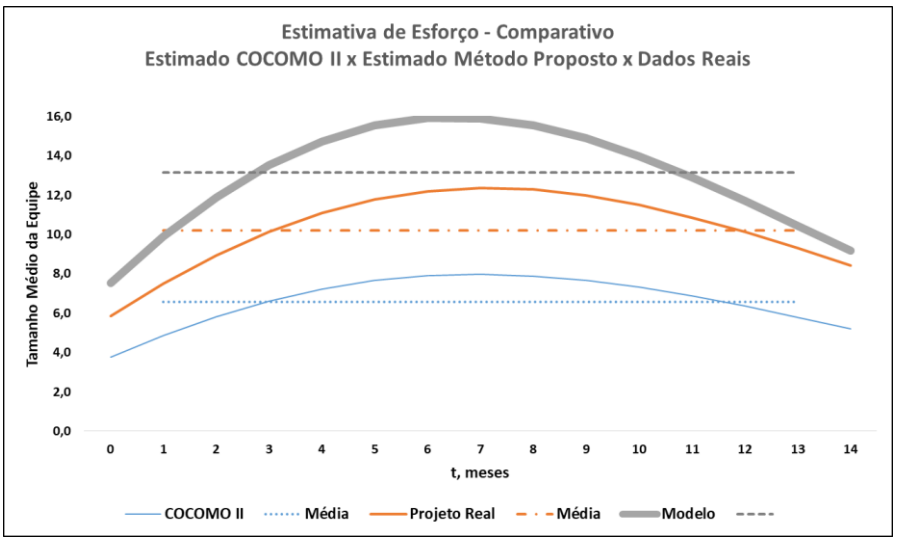
O desenvolvimento deste trabalho envolve:

- O estudo da literatura e dos conceitos de estimativa de esforço em projetos de desenvolvimento de software em geral, em especial o COCOMO II;
- A apresentação da norma DO-178C, seus processos, características e aplicação;
- O estudo de caso de um projeto real executado pela indústria aeronáutica brasileira seguindo e cumprindo com todos os objetivos da DO-178C, analisando o esforço real demandado por tal projeto em cada uma de suas fases;
- A aplicação do modelo COCOMO II, em sua versão original, no projeto do estudo de caso;
- A proposta de um novo modelo baseado no COCOMO II com as considerações dos objetivos e métodos de cumprimento da DO-178C;
- A aplicação do novo modelo proposto neste trabalho em dois diferentes projetos, um do setor aeronáutico e outro espacial; e finalmente

- Uma análise dos resultados obtidos comparando o resultado do novo método contra o obtido pelo COCOMO II em sua versão original, comparando ainda contra os dados reais do projeto estudado por meio da medição da magnitude do erro relativo.

Ao final da aplicação da metodologia proposta, espera-se então a obtenção de três curvas de esforço para cada um dos projetos estudados, (i) a curva do valor estimado pelo COCOMO II; (ii) a curva do valor estimado pelo método proposto e (iii) a curva dos valores reais obtidos pelo estudo de caso, conforme exemplificado na Figura 1.1 a seguir, com dados de demonstração.

Figura 1.1 – Gráfico comparativo: COCOMO II *x* Método Proposto *x* Real



Fonte: Produção do autor.

Como medida para avaliar a precisão do modelo e a comparação entre eles adota-se a magnitude do erro relativo *(MRE - Magnitude of Relative Error)*, onde:

$$MRE = \frac{|esforço\ real - esforço\ estimado|}{esforço\ real}$$

Este método é adotado por ser o mais comum e amplamente utilizado (GRIMSTAD, 2006).

1.5 Organização do Documento

Este documento está estruturado da seguinte forma:

a) O Capítulo 2 apresenta uma revisão bibliográfica sobre os métodos mais utilizados de estimativa de esforço em projetos de desenvolvimento de software;

b) O Capítulo 3 apresenta de forma introdutória os conceitos sobre software crítico, uma introdução sobre o regulamento aeronáutico produzido pela RTCA, a DO-178C, além de uma breve evolução desde sua primeira versão;

c) O Capítulo 4 apresenta os trabalhos correlatos, bem como uma análise comparativa entre eles.

d) O Capítulo 5 apresenta o estudo de caso de aplicação do modelo COCOMO II em um projeto de desenvolvimento de software crítico na indústria aeronáutica, seguindo e cumprindo com os objetivos da norma DO-178C.

e) O Capítulo 6 apresenta uma proposta de adaptação ao modelo COCOMO II para considerar os objetivos da norma RTCA/DO-178C na estimativa do esforço em projetos de desenvolvimento de software crítico;

f) O Capítulo 7 apresenta os passos para aplicação do modelo de estimativa de custo proposto por essa dissertação;

g) O Capítulo 8 apresenta a aplicação do modelo proposto em dois diferentes projetos de desenvolvimento de software. O primeiro deles é o mesmo projeto objeto do estudo de caso do capítulo 5. O segundo projeto é referente ao veículo espacial FireSat.

h) O Capítulo 9 apresenta a conclusão, publicações efetuadas e trabalhos futuros.

O estudo dos fundamentos conceituais apresentados a seguir é imprescindível para identificar e definir os conceitos aqui utilizados.

2 ESTIMATIVA DE ESFORÇO PROJETOS DE SOFTWARE

A dificuldade em estimar o esforço em projetos de desenvolvimento de software é algo conhecido pela indústria e academia, as pesquisas cientificas sobre este tema (MOLOKKEN; JØRGENSEN, 2003; JØRGENSEN; SHEPPERD, 2007; BASTEN; MELLIS, 2011; BRITTO et al., 2014) e os relatórios de consultorias privadas (THE STANDISH GROUP, 1995, 2012) tornam o problema mais evidente ao olhar do leitor e mostram que tal dificuldade não é característica única de um país ou organização. Obviamente o problema de estimar o custo também não é exclusivo para projetos de desenvolvimento de software.

O gerenciamento de qualquer projeto passa por um bom planejamento e estimativa de custo, de modo a poder se avaliar inclusive a viabilidade de tal projeto. O PMBOK, *Project Management Body of Knowledge* (PMI, 2008), mapeia nove áreas do conhecimento em seu guia de gerenciamento de projetos, sendo uma delas dedicada ao gerenciamento dos custos de projeto. No contexto de Engenharia de Software, o SWEBOK (IEEE COMPUTER SOCIETY, 2004), um guia análogo ao PMBOK, porém dedicado à Engenharia de Software, em seu capítulo de Software Engineering Management também dedica uma seção exclusiva a Effort, Schedule, and Cost Estimation. Além disso, renomados autores da Engenharia de Software também dedicam capítulos inteiros sobre este tema. Pressman (2009) dá ênfase a métodos empíricos e aborda o tema dentro da fase de planejamento do projeto. Sommerville (2007), por sua vez, também dedica um capítulo inteiro na seção de gerenciamento. O relatório CHAOS Manifesto (THE STANDISH GROUP, 2012), também reforça que a estimativa de custo é um dos fatores de sucesso nos projetos de desenvolvimento de software.

Apesar de tamanho interesse da indústria, academia e dos pesquisadores, o tema ainda é mal interpretado ou interpretado de diferentes maneiras por diversos autores. Grimstad et al. (2006) destacam que a falta de padronização da terminologia é um importante obstáculo no aprimoramento dos métodos e aumento de precisão. O estudo realizado por Grimstad et al. (2006) sugere que

a razão para tal falta de precisão do termo “estimativa” dentro das organizações se dá pela própria falta de precisão nos livros e artigos relacionados ao tema. Mesmo comparando renomados autores, como Pressman, Sommerville e Pfleeger, os autores Grimstad et al. (2006) concluem que nenhum deles fornecem uma definição precisa sobre o que cada um deles entendem por “estimativa de esforço”, bem como se trata-se de uma “técnica” de estimativa, um “método” ou um “modelo”.

Sendo assim, neste trabalho adota-se a definição de estimativa de esforço utilizada por Basten e Mellis (2011): "o esforço que é mais provável ser necessário para implementar a tarefa que é estimada (excluindo qualquer buffer de risco)." Para medir o esforço, adota-se a unidade PM (Pessoa-Mês), na qual 1PM = 152 horas (BOEHM et al., 2000).

Essa definição está alinhada também com a recomendação feita por Jørgensen (2014b), na qual ele sugere que o significado de “estimativa de esforço” seja entendido e comunicado como uma terminologia baseada em probabilidade.

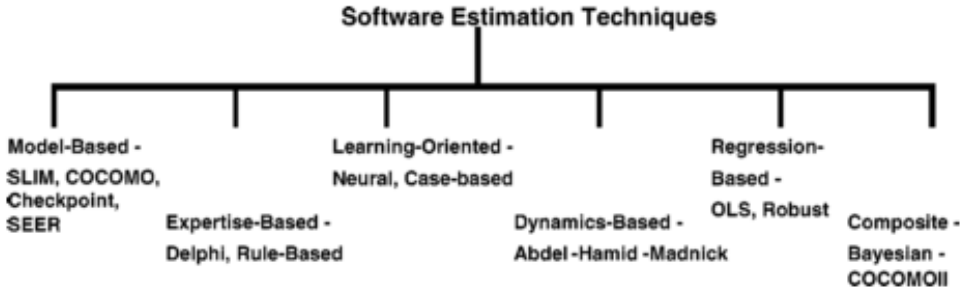
Os termos “método”, “modelo” ou “técnica” serão utilizados de forma intercambiável e considerados sinônimos neste trabalho. Os principais métodos, ou técnicas, de estimativa de esforço em projetos de desenvolvimento de software são apresentados a seguir.

2.1 Técnicas de estimativa de esforço em projetos de software

Existem diversas técnicas de estimativa de esforço em projetos de desenvolvimento de software. Diferentes autores as classificam de várias maneiras. Khatibi e Jawawi (2011) por exemplo, classificam todas as técnicas apenas em dois grupos: métodos algorítmicos e não-algorítmicos. A mesma classificação é adotada por Abbas et al. (2012) em sua pesquisa sobre o histórico das conhecidas técnicas de estimativa de custos. Os autores Jørgensen e Shepperd (2007), por sua vez, em sua revisão de 304 artigos sobre estimativa de esforço em desenvolvimento de software, oferecem uma classificação mais detalhada, com as seguintes categorias: regressão, analogia, opinião especializada, *breakdown* de atividades, pontos de função, CART (*classification and regression-trees),* simulação, redes neurais, teóricos,

bayesian e, por fim, a combinação de qualquer dois métodos. Essa classificação mais detalhada vai encontro do modo como Boehm et al. (2000) classificam as técnicas de estimativa de esforço em sua pesquisa sobre as diferentes abordagens mais utilizadas, agrupando os métodos destacados por eles em *Model-Based*, *Expertise-Based*, *Learning-Oriented*, *Dynamic-Based*, *Regression-Based* e métodos compostos, conforme mostra a Figura 2.1 a seguir. Nesta figura, os métodos SLIM, COCOMO, Checkpoint e SEER são classificados como sendo Model-Based.

Figura 2.1 – Técnicas de estimativa de esforço



Fonte: Boehm et al. (2000).

Sommerville (2007), por sua vez, também apresenta uma classificação baseada em métodos algorítmicos e não-algorítmicos, porém com um detalhamento maior, dividindo os métodos em:

- Métodos algorítmicos
- Opinião especializada
- Estimativa baseada em analogia
- Lei de Parkinson
- Estimativa para vencer (*Pricing to win*)

Obviamente, na classificação feita por Sommerville há uma clara divisão entre algorítmicos e não-algorítmicos, mesmo o autor não utilizando o termo “não-algorítmicos”.

Na revisão apresentada por Rijwani et al. (2014), apesar de também utilizarem uma classificação nos mesmos dois grupos citadores anteriormente (algorítmicos e não-algorítmicos), utilizam ainda a classificação dos métodos em *Top-Down* e *Bottom-Up,* baseando-se na quebra das atividades do projeto

de cima para baixo ou de baixo para cima. Este mesmo conceito de categorização também é compartilhado por Sharma et al. (2012) em sua pesquisa.

Claramente, a literatura apresenta uma grande variedade de classificações, que pode levar o leitor a acreditar que existem mais categorias do que métodos, porém obviamente há uma intersecção entre todas essas classificações e um mesmo método por ser classificado de duas ou mais maneiras diferentes, como por exemplo Rijwani et al. (2014) que classificam o modelo COCOMO como sendo uma técnica *Bottom-Up*. Os autores Jørgensen e Shepperd (2007), por sua vez, o classificam como sendo uma técnica de regressão. Finalmente, Boehm et al. (2000) o classifica simplesmente como sendo uma técnica *Model-Based.* A Tabela 2.1 a seguir detalha a intersecção de classificação apresentada na literatura.

Tabela 2.1 – Diferentes classificações do mesmo método na literatura

Método	Modelo de Estimativa	Autor (es)
COCOMO	Algorítmico	(ABBAS et al., 2012)
		(JØRGENSEN, 2004)
		(KHATIBI; JAWAWI, 2011)
		(SHARMA; BAJPAI; LITORIYA, 2012)
	Model-Based	(BOEHM; ABTS; CHULANI, 2000)
		(MOLOKKEN; JØRGENSEN, 2003)
	Regressão	(JØRGENSEN; SHEPPERD, 2007)
	Bottom-up	(RIJWANI; JAIN; SANTANI, 2014)

Fonte: Produção do autor.

É possível observar que o mesmo autor classifica o mesmo método de maneira diferente em publicações distintas.

Deste modo, neste trabalho adota-se então a forma mais simplificada de classificação dos métodos utilizada por Khatibi e Jawawi (2011), dividindo os principais métodos (MOLOKKEN; JØRGENSEN, 2003; GRIMSTAD, 2006; JØRGENSEN; SHEPPERD, 2007) nos dois grupos, algorítmicos e não-algorítmicos, que são detalhados a seguir.

2.2 Métodos Não-algorítmicos

Nos últimos anos estimativa de esforço em desenvolvimento de software é um dos tópicos mais pesquisado pelos profissionais da área. Jørgensen e Shepperd (2007) mostram em sua revisão que entre os anos 2000 e 2004, 58% dos artigos publicados nos principais periódicos e jornais científicos da área foram sobre métodos de estimativa. Destes, cerca de 56% tratam de métodos não-algorítmicos. Neste grupo estão métodos que se baseiam em comparações, inferências, analogias ou opinião de pessoas especializadas e experientes para se estimar o esforço necessário em um determinado projeto. Basicamente, não implementam nenhum tipo de equação matemática para se obter o resultado. Dada sua simplicidade e flexibilidade, estes são os métodos mais utilizados nos projetos atualmente (BASTEN; MELLIS, 2011), além do fato de haver uma ausência de evidência de que métodos algorítmicos são mais precisos do que os não-algorítmicos (JØRGENSEN, 2004).

Nesta seção se elabora com mais detalhes as técnicas básicas dos dois principais métodos não-algorítmicos pesquisados (MOLOKKEN; JØRGENSEN, 2003; JØRGENSEN; SHEPPERD, 2007; BASTEN; MELLIS, 2011): opinião especializada (*Expert Judgment*) e estimativa baseada em analogia (*Analogy Based*).

2.2.1 Estimativa Baseada em Analogia (Analogy Based)

Analogia é qualquer relação de semelhança entre coisas ou fatos distintos, segundo o dicionário. Essa definição facilita o entendimento de métodos baseados em analogia, pois são aqueles que se baseiam em dados históricos de projetos anteriores que sejam semelhantes e que, provavelmente, estão submetidos à custos semelhantes ao projeto a ser estimado. Boehm et al. (2000) classifica esse método como sendo uma técnica orientada ao aprendizado (*Learning-oriented technique*) e a batiza como um método de Estudo de Caso. Shepperd e Schofield (1997), em seu estudo sobre estimativa de software baseado em analogia, batizam tal método como *Case Based Reasoning* (CBR), ou raciocínio baseado em casos.

Na literatura pode-se encontrar os seguintes pontos de vantagens e desvantagens deste método:

Vantagens:

- A estimativa é baseada em dados reais de projetos anteriores (SHARMA; BAJPAI; LITORIYA, 2012; RIJWANI; JAIN; SANTANI, 2014).
- As diferenças do projeto anterior com o projeto proposto podem ser identificadas e seus impactos estimados (SHARMA; BAJPAI; LITORIYA, 2012; RIJWANI; JAIN; SANTANI, 2014).
- O conhecimento e experiência do responsável pela estimativa é considerada (SHARMA; BAJPAI; LITORIYA, 2012).
- Por se basear em dados reais de projetos passados, esse método dispensa a necessidade da organização possuir pessoas com alto nível de experiência (KHATIBI; JAWAWI, 2011). Essa vantagem é muito útil em caso de empresas que estejam perdendo seu capital intelectual.

Por outro lado, os autores Rijwani et al. (2014) também destacam as seguintes desvantagens:

Desvantagens:

- É preciso determinar como descrever um projeto da melhor maneira possível. A escolha das variáveis do projeto deve ser restrita à informação disponível no momento em que a estimativa se faz necessária. As possibilidades de analogia são muitas, como o domínio da aplicação, número de entradas, o número de entidades distintas referenciadas, o número de telas e assim por diante.

Para Sharma et al. (2012), este método possui ainda a seguinte desvantagem:

- Mesmo após uma boa descrição do projeto em questão, é preciso determinar ainda o nível de similaridade e o quão confiante estamos para fazer as analogias. Poucas analogias podem levar a escolha de projetos ruins para comparação. Muitas, por outro lado, pode levar a diluição do efeito de analogias próximas.

Uma desvantagem apontada por Jorgensen (2013) em seu estudo empírico sobre estimativas baseada em analogia é o efeito “otimismo em excesso” (*over-optimistic*) em função de comparações tendenciosas. Ele comparou o resultado de quatro diferentes estudos e observou a tendência de considerarmos as tarefas mais similares do que realmente são quando comparadas umas com as outras.

Para Jorgensen (2013), “o quê e o como você compara” faz toda diferença no sucesso da estimativa. Sucesso este que é comprovado pela literatura, pois este método se apresenta como um dos mais utilizados e pesquisados. Cerca de 15% dos artigos pesquisados na revisão sistemática conduzida por Jørgensen e Shepperd (2007) são sobre este método. Já Jørgensen (2004) identificou que o estudo de práticas de estimativa de desenvolvimento de software no Laboratório de Propulsão a Jato descobriu que 83% dos estimadores usavam a analogia como suas técnicas de estimativa primária (Hinn e Habib-Agahi, 1991 apud JØRGENSEN, 2004). Os autores Basten e Mellis (2011), por sua vez, identificaram que em 44% dos projetos analisados o método utilizado para estimativa de esforço foi o de analogia. Na indústria de software da Noruega, por exemplo, este método vem sendo amplamente utilizado desde a década de 80, em mais de 60% dos projetos pesquisados (MOLOKKEN-OSTVOLD et al., 2004).

Uma característica deste método é que ele pode ser usado em conjunto com outro método. Conforme destacado por (KASHYAP; MISRA, 2013), o método *Analogy Based* pode ser usado para calibração ou identificação dos fatores de custo do modelo COCOMO II (BOEHM et al., 2000), por exemplo, que será detalhado nos próximos capítulos.

Pode parecer então que este é “o” melhor método a ser utilizado, porém algumas pesquisas mostram que a adoção de um determinado método de estimativa de esforço se dá em função do sucesso em experiências passadas com a utilização de tal método (MOLOKKEN-OSTVOLD et al., 2004; JØRGENSEN, 2010; BASTEN; MELLIS, 2011), ou seja, ser o melhor para um não significa que é o melhor para todos.

A pesquisa realizada com 275 projetos de projetos da indústria (SHEPPERD; SCHOFIELD, 1997) mostra que esta é uma técnica muito valiosa e que, no mínimo, pode ajudar os gerentes de projetos para complementar o resultado de outras técnicas já utilizadas. Nesta mesma pesquisa concluiu-se ainda que este método apresentou uma melhor performance que todos os outros métodos comparados. Obviamente, isso não significa que a estimativa por analogia é um método livre de qualquer ponto fraco, conforme concluem os próprios autores em (SHEPPERD; SCHOFIELD, 1997) e apontam as desvantagens já descritas anteriormente.

Nas próximas seções serão apresentados outros métodos que também possuem performance encorajadora e que, a escolha de um deles em particular torna-se um desafio e objeto de pesquisas (JØRGENSEN, 2010). Para tornar a seleção ainda mais difícil, algumas vezes a combinação de dois ou mais métodos pode aumentar a performance e trazer melhores resultados (JØRGENSEN; INDAHL; SJØBERG, 2003; MACDONELL; SHEPPERD, 2003; MOLOKKEN; JØRGENSEN, 2003; JØRGENSEN, 2007; BASTEN; MELLIS, 2011).

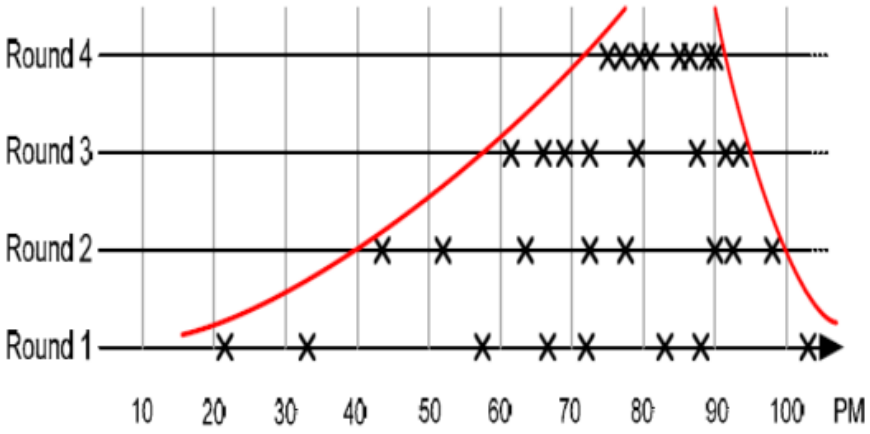
2.2.2 Opinião Especializada (Expert Judgment)

O método de opinião especializada baseia-se na consulta de especialistas no assunto para se obter uma estimativa do esforço necessário para um determinado projeto. Algumas vezes também conhecido na literatura por método Delphi (HALKJELSVIK; JØRGENSEN, 2012), é apresentado por várias pesquisas como o método mais utilizado pela indústria (MOLOKKEN; JØRGENSEN, 2003; JØRGENSEN, 2004; BASTEN; MELLIS, 2011) e um dos mais pesquisados pela academia (JØRGENSEN; SHEPPERD, 2007). O processo de obter a estimativa de custo e prazo de especialistas no assunto é conduzido por uma pessoa no papel de coordenador, que organiza reuniões dedicadas com os especialistas e a cada rodada os valores são refinados e confrontados uns com os outros. Os passos para obtenção da estimativa final são descritos a seguir (BOEHM; ABTS; CHULANI, 2000; KHATIBI; JAWAWI, 2011; SHARMA; BAJPAI; LITORIYA, 2012; RIJWANI; JAIN; SANTANI, 2014):

- O coordenador entrega um formulário de estimativa para cada especialista.
- Cada especialista apresenta sua estimativa, sem consultar seus pares.
- O coordenador compila todos os formulários, calcula a média e mediana e pede uma nova rodada de interação dos especialistas.
- Os passos anteriores são repetidos até que todos os especialistas aprovem o valor final.

Deste modo, a cada rodada os valores mais distorcidos vão sendo eliminados conforme a estimativa se torna mais precisa e, por tanto, a opinião especializada do grupo é prevalecida em detrimento de um único indivíduo. A Figura 2.2 apresenta um exemplo de aplicação da técnica, na qual oito especialistas contribuem com suas estimativas (pontos “x” no eixo PM) e, ao final de 4 interações (Rounds), convergem para um valor final. O eixo y representa as rodadas de interações. O eixo x representa o valor estimado por cada especialista, em PM (*Person-Month*, ou pessoa-mês).

Figura 2.2 – Exemplo de aplicação da técnica Delphi



Fonte: Khatibi e Jawawi (2011).

Basten e Mellis (2011) apontam em seu estudo que os profissionais escolhem um método de estimativa de esforço principalmente em função de haver obtido sucesso em experiências passadas com tal método. Por este ser o método mais amplamente utilizado e pesquisado, conclui-se então que esta abordagem

se mostra eficaz em sua aplicação. A revisão elaborada por Jørgensen (2004) reforça essa conclusão, e evidencia que em situações onde o especialista tem grande domínio do assunto, este método se mostra mais preciso em sua estimativa do que métodos algorítmicos por exemplo.

Assim como outros métodos, este também possui suas vantagens e desvantagens:

Vantagens:

- Os especialistas podem comparar as diferenças de experiências passadas ou requisitos do projeto atual (SHARMA; BAJPAI; LITORIYA, 2012; RIJWANI; JAIN; SANTANI, 2014).
- Os especialistas podem levar em consideração os impactos de novas tecnologias, arquitetura, aplicação e linguagens de programação no novo projeto a ser estimado, além de considerações de pessoal e equipe (SHARMA; BAJPAI; LITORIYA, 2012; RIJWANI; JAIN; SANTANI, 2014).
- Obtém-se o resultado da estimativa de forma rápida e com baixo custo (KHATIBI; JAWAWI, 2011; ABBAS et al., 2012).

Desvantagens:

- O especialista por estar influenciado por uma experiência passada super otimista ou pessimistas (HALKJELSVIK; JØRGENSEN, 2012; RIJWANI; JAIN; SANTANI, 2014).
- É difícil documentar os fatores de comparação utilizado pelos especialistas (RIJWANI; JAIN; SANTANI, 2014).
- A estimativa é tão boa quanto a opinião do especialista e mesmo anos de experiência não se traduzem em altos níveis de competência (BOEHM; ABTS; CHULANI, 2000; JØRGENSEN; SJOBERG, 2000).
- Por natureza, o resultado é subjetivo (ABBAS et al., 2012).
- Para o mesmo problema, diferentes especialistas darão diferentes resultados (ABBAS et al., 2012).

- O nível de experiência do especialista influencia o resultado da estimativa (ABBAS et al., 2012).
- Torna-se difícil convencer o cliente do custo apenas baseado na opinião de uma pessoa (ABBAS et al., 2012).

Um dos principais problemas deste método é a informação tendenciosa que pode surgir da opinião especializada. Quem nunca ouviu a expressão “a primeira impressão é a que fica”? Jørgensen e Løhre (2012) compararam quatro estudos diferentes examinando a influência da primeira impressão na estimativa de esforço de projetos de desenvolvimento de software e concluíram que é fácil de se criar uma primeira impressão incorreta sobre o esforço necessário para se resolver determinada atividade e que tal impressão se torna resistente à mudanças. Os mesmos autores concluem ainda que uma primeira impressão tendenciosa causada por uma solicitação prematura de um cliente ou gerente, pode ser muito difícil de ser revertida.

Outro ponto levantando por Grimstad e Jørgensen (2007) é a inconsistência da estimativa de esforço gerada pelo mesmo especialista. No estudo conduzido por eles, a mesma informação apresentada para o mesmo especialista em diferentes ocasiões chegou a apresentar diferenças de até 71% entre as estimativas. Eles concluíram ainda que o maior fator de erros de estimativa se dá na própria inconsistência os especialistas do que, por exemplo, a uma tendência de otimismo. Outra conclusão interessante destes autores é que não existem evidencias de que os especialistas historicamente mais precisos são também mais consistentes.

Além da inconsistência de um mesmo especialista descrita no parágrafo anterior, a opinião de um grupo deles pode ser mais divergente ainda, uma vez que um mesmo evento é percebido de maneira diferente por pessoas diferentes. Jørgensen e Sjoberg (2000) listaram alguns dos problemas da utilização de experiências passadas na estimativa de esforço de software e um deles é que muito de nossa experiência não se aplica para projetos futuros, uma vez que (1) mudanças das condições atuais invalidam os dados históricos; (2) experiência é dependente do contexto e o contexto é difícil, ou até mesmo

impossível, de se descrever. Estes mesmo autores reforçam ainda a influência de uma das fraquezas humanas: a tendência de fazermos previsões otimistas.

Por fim, mas não menos importante, é o fato deste método ser influenciado até mesmo pela unidade de tempo escolhida para se realizar a estimativa de esforço (JØRGENSEN, 2015, 2016). Jørgensen (2015) conduziu dois experimentos diferentes e concluiu que em ambos a unidade de tempo produziu um grande efeito no resultado final da estimativa. Segundo ele, estimativas em minutos são maiores que em segundos, e estimativas em dias de trabalho são maiores que em horas de trabalho.

Mesmo seguindo todos as práticas e *guidelines* sugeridos na literatura, este continua sendo um método baseado na experiência passada de pessoas. Por este motivo é que Boehm et al. (2000) reforça que este método é mais útil na ausência de dados quantitativos e empíricos. E é justamente sobre métodos que utilizam dados e métricas que trata a próxima seção.

2.3 Métodos Algorítmicos

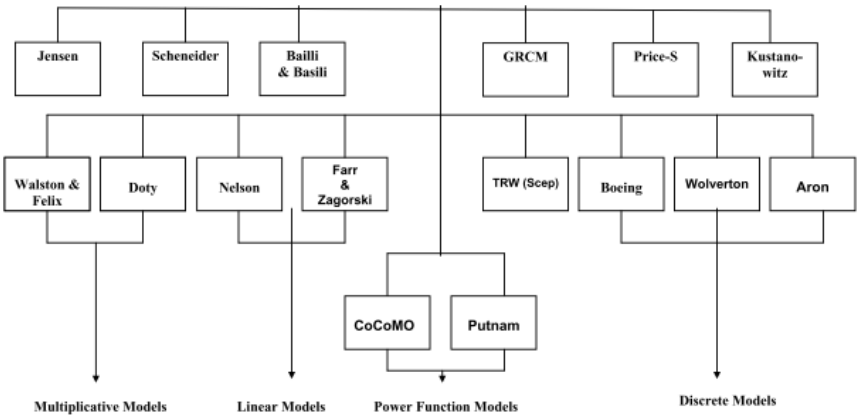
Métodos algorítmicos são aqueles que utilizam algum tipo de equação matemática para relacionar métricas de software com o esforço necessário. Em geral, a principal entrada do algoritmo é o tamanho do software, medidos em quantidade de linhas de código por exemplo, que é então processado pelo modelo e convertido na estimativa de esforço requerido, em geral uma medida de horas totais de desenvolvimento.

Existem diversos tipos de métodos e técnicas classificados como algorítmicos. Conforme apresentado na Tabela 2.1, a classificação de um método de estimativa de esforço pode variar conforme autor.

Para os autores Abbas et al. (2012), por exemplo, dentre os métodos algorítmicos estes existem ainda algumas subcategorias: modelos multiplicativos, lineares, modelos de potência de função e modelos discretos.

A Figura 2.3 apresenta todos os métodos algorítmicos e suas subclassificações explorados por estes autores.

Figura 2.3 – Métodos algorítmicos



Fonte: Abbas et al. (2012).

Alguns dos métodos apresentados na Figura 2.3 caíram em desuso ou são pouco pesquisados, por serem proprietários e vinculados a ferramentas comerciais. O Modelo de Putnam (PUTNAM, 1978) por exemplo, criado em 1978, foi um dos primeiros (BOEHM; VALERDI, 2008) e mais conhecidos métodos algorítmicos, porém hoje não tão explorado quanto outros métodos (JØRGENSEN; SHEPPERD, 2007). Por outro lado, alguns destes métodos estão em constante evolução e são objeto de pesquisa de várias universidades e pesquisadores. Dos 304 artigos pesquisados por Jørgensen e Shepperd (2007), por exemplo, 148 deles (ou 49%), tratavam de métodos algorítmicos. Nesta categoria se encontra por exemplo o consagrado COCOMO (COnstructive COst MOdel), desenvolvido por Boehm (1981) e principal método a ser explorado neste trabalho. Ao contrário de Abbas et al. (2012), será considerado como sendo *Model-Based* (Baseado em Modelo), uma vez que é a classificação atribuída pelo seu próprio criador, Boehm et al. (2000).

Deste modo, nas próximas seções detalha-se o modelo COCOMO, desde sua primeira versão (BOEHM, 1981) até sua a versão mais recente, o COCOMO II (BOEHM et al., 2000).

2.3.1 COCOMO'81

O modelo COCOMO é batizado em função das iniciais em inglês para *COnstructive COst MOdel*, ou Modelo de Custo Construtivo, foi originalmente proposto por Boehm (1981) e por isso também é conhecido como

COCOMO’81. Muitos autores o coloca como o método mais completo e mais bem documentado dentre todos os métodos existentes (ABBAS et al., 2012), além de ser um dos mais populares (KHATIBI; JAWAWI, 2011).

Para evitar repetições de referências, toda esta seção baseia-se no livro Software Engineering Economics (BOEHM, 1981), exceto quando especificado.

O método proposto por Boehm tem como base uma fórmula básica de regressão, cujo a entrada é o tamanho do software em número de linhas de código e os parâmetros são frutos de dados históricos de um banco de dados de 63 projetos selecionados. O modelo depende de duas equações principais, as equações (2.1) e (2.2) a seguir:

$$MM = x\ (KDSI)^{y} \quad (2.1)$$

$$TDEV = 2.5(MM)^{z} \quad (2.2)$$

Onde:

MM: Medida de Esforço em Man-Months (Pessoa-Mês). 1MM = 152 horas por pessoa.

KDSI: Delivered Source Instruction (Instrução de código entregue) x 1000.

TDEV: Tempo de desenvolvimento em meses.

x, y, z: são coeficientes obtidos dependendo do modo de desenvolvimento.

O método propõe três diferentes níveis de detalhamento para o cálculo da medida de esforço (MM): Básico, Intermediário e Detalhado:

- **Básico:** utiliza apenas as equações apresentadas anteriormente, com a única entrada sendo o tamanho do código em KDSI. É indicado para grande a maioria dos desenvolvimentos de software pequenos, por ser rápido e simples, porém sua precisão é limitada em função da incapacidade de considerar fatores importantes, como hardware, complexidade etc. Analisando o banco de dados do COCOMO 81, apenas em 29% dos casos as estimativas obtidas com o nível básico possuem fator de precisão de 1.3 quando comparados com os valores reais, e com fator de precisão 2 em 60% dos casos.

- **Intermediário:** nesta versão o modelo agrega 15 fatores multiplicadores de custo para o cálculo da medida de esforço. Esses fatores são obtidos de maneira subjetiva a partir de análises de atributos do software, hardware, equipe e atributos do projeto. Com este detalhamento, o modelo apresenta um fator de precisão de 1.2 em 68% dos projetos analisados.
- **Detalhado:** nesta versão o modelo herda os mesmos 15 multiplicadores da versão anterior, porém sendo calculados para cada fase do projeto (*Product Design, Detail Design, Coding and Unit Test, Integration and Test)* e para cada nível (módulo, subsistema e sistema).

O modelo propõe ainda três diferentes modos de desenvolvimento:

- **Modo Orgânico (Organic Mode):** este é o modo de desenvolvimento de software relativamente pequenos, desenvolvidos por times com poucas pessoas e de baixa complexidade. São projetos em geral com no máximo 50KDSI.
- **Modo Misto (Semidetached Mode):** este modo possui características tanto do modo orgânico quanto do modo embarcada. Trata-se de projetos considerados médios, chegando a ter até 300KDSI e desenvolvidos por equipes maiores e mais experientes.
- **Modo Embarcado (Embedded Mode):** a principal característica deste modo é o fato do software operar com restrições muito apertadas, seja de hardware, performance, regulamentações e normas. Exemplos de software deste tipo são software de controle de tráfego área, controle de aeronaves ou satélites.

A Tabela 2.2 apresenta as principais características de cada modo de desenvolvimento.

Tabela 2.2 – Características dos projetos

Modo	Características			
	Tamanho	**Nível de inovação**	**Restrições**	**Ambiente**
Orgânico	Pequeno	Nenhum/Pequeno	Nada/Pouco	Estável
Misto	Médio	Médio	Médio	Médio
Embarcado	Grade	Alto	Altas	Hardware Complexo

Fonte: Adaptado de Rijwani (2014).

No modo básico as equações apresentadas anteriormente são aplicadas sem qualquer modificação e seus coeficientes são aplicados considerando apenas os modos de desenvolvimento, básico, misto ou embarcado.

Tabela 2.3 – COCOMO Básico: coeficientes de Esforço e Tempo

Modo	Esforço (MM)	Tempo (TDEV)
Orgânico	$MM = 2.4\ (KDSI)^{1.05}$	$TDEV = 2.5(MM)^{0.38}$
Misto	$MM = 3.0\ (KDSI)^{1.12}$	$TDEV = 2.5(MM)^{0.35}$
Embarcado	$MM = 3.6\ (KDSI)^{1.20}$	$TDEV = 2.5(MM)^{0.32}$

Fonte: Adaptado de Boehm (1981).

Não se faz necessário um detalhamento maior das diferenças entre os níveis básico, intermediário e avançado, uma vez que esta versão do método já foi substituída pelo COCOMO II, que será detalhado mais adiante.

Por ser um método algorítmico, pode-se identificar as seguintes vantagens e desvantagens:

Vantagens:

- É determinístico, ou seja, para as mesmas entradas tem-se sempre as mesmas saídas (RIJWANI; JAIN; SANTANI, 2014).
- É fácil de modificar um dado de entrada, refinar e customizar suas fórmulas (RIJWANI; JAIN; SANTANI, 2014).
- É um método consagrado e muito utilizado pela indústria e pesquisadores (JØRGENSEN; SHEPPERD, 2007).

- Pode ser automatizado e sistematizado em um banco de dados (BOEHM; ABTS; CHULANI, 2000).
- Foi validado em um banco de dados de projetos reais (BOEHM, 1981).

Desvantagens:

- Principal critério de entrada é o tamanho do software, que pode ser difícil de estimar em fases iniciais do desenvolvimento (SHARMA; BAJPAI; LITORIYA, 2012).
- Não considera fatores relacionado aos requisitos ou documentação (RIJWANI; JAIN; SANTANI, 2014).
- Alguns autores não observaram grandes melhoras na aplicação dos níveis intermediário ou detalhado, quando comparados com o básico (SHARMA; BAJPAI; LITORIYA, 2012).
- Por ser um método criado em 1981, não considera fatores modernos do ciclo de vida de desenvolvimento de software, como reuso, métodos ágeis, orientação a objeto, níveis de maturidade etc (SHARMA; BAJPAI; LITORIYA, 2012). Tampouco considera novas tecnologias, como geração automática de código, desenvolvimento baseado em modelo etc.

Em função de tais desvantagens, Boehm et al. (2000) criaram o modelo COCOMO II, que será detalhado na próxima seção.

2.3.2 COCOMO II

O modelo do COCOMO II foi criado a partir da sua primeira versão COCOMO 81 e incorpora cerca de 20 anos de evolução da disciplina de Engenharia de Software em geral. As primeiras pesquisas que originaram esta atualização começaram em 1994 em função da rápida evolução no mundo de software e das recentes tecnologias que borbulhavam àquela época, como processo de desenvolvimento ágil, reengenharia, abordagem de reuso, orientação a objeto etc (BOEHM; ABTS; CHULANI, 2000), porém foi publicado em sua versão finalizada apenas no ano 2000, o que o deixou conhecido também como

COCOMO.2000. Assim como na primeira versão, este método também se mostra bastante popular , amplamente utilizado e com resultados satisfatórios (KHATIBI; JAWAWI, 2011).

Para evitar repetições de referências, toda esta seção baseia-se no livro *Software Cost Estimation with Cocomo II* (BOEHM et al., 2000), exceto quando especificado.

Segundo Boehm et al. (2000), o COCOMO II tem como principal objetivo cobrir os pontos fracos recebidos como *feedback* dos usuários do COCOMO'81 nos últimos 20 anos que antecederam seu lançamento, são eles:

- Prover uma estimativa de custo e prazo precisa para projetos tanto já em andamento quanto futuros. O modelo COCOMO'81 foi criado baseado no processo de desenvolvimento conhecido como "*waterfall"*, porém muitas organizações migram para processos mais modernos, como ágeis, cíclicos ou híbridos.
- Permitir que as organizações possam calibrar ou ajustar o método para sua realidade.
- Prover uma cuidadosa e simples definição das entradas, saídas e premissas do modelo.
- Prover um modelo construtivo. Já era um objetivo do COCOMO'81 e continua sendo nesta versão.
- Prover um modelo normativo e flexível, que permita que novas capacidades sejam agregadas e que naturalmente seja evoluído. O modelo COCOMO'81, por sua vez, não havia sido planejado para ser evolutivo.

Para cumprir todos os objetivos, o modelo também utiliza como entrada principal o tamanho do software em questão, medido pelo número de instruções de código ou, nesta versão mais moderna, por Pontos de Função. O método prevê ainda um conjunto de 17 fatores multiplicativos de custo (na versão COCOMO'81 eram 15) e 5 fatores de escala para determinação das características do projeto (na versão COCOMO'81 existem os modos de

desenvolvimento: orgânico, misto e embarcado). A aplicação desses fatores varia conforme a aplicação dos submodelos apresentados pelo COCOMO II, são eles:

- ***Early Design Model (Modelo de Definição Inicial):*** Este é o modelo mais macro, de alto nível, que explora alternativas para definição da arquitetura do projeto a ser desenvolvido. Neste modelo, tipicamente pouco se sabe ainda do produto e não há realizar uma estimativa precisa ou detalhada. Este modelo baseia-se em pontos de função, ou linhas de código, e utiliza os 5 fatores e escala e apenas 7 dentre os 17 fatores multiplicativos de custo (BOEHM; ABTS; CHULANI, 2000).
- ***Post-Architecture Model (Modelo de Pós-Arquitetura):*** Esta é a versão mais detalhada do modelo e deve ser utilizada quando o projeto está pronto para ser desenvolvido, possui um ciclo de vida e arquitetura bem definidos, na qual irão prover as informações necessárias para a estimativa dos fatores de custo. Neste cenário, uma boa precisão é possível em função do detalhe das informações que já estão disponíveis. Deste modo, todos os 17 fatores multiplicativos são considerados, bem como os 5 fatores de escala.

Apesar dos dois modelos apresentarem propósitos diferentes, ambos compartilham a mesma abordagem para definição do tamanho do software (incluindo questões de reuso) e utilização dos fatores de escala, compartilham ainda as fórmulas para estimativa de esforço e prazo para o desenvolvimento de um determinado projeto.

As fórmulas para o cálculo do esforço e prazo são apresentadas a seguir.

Cálculo do Esforço:

A medida de esforço em *PM* (*Person-months)* é definida pelas equações (2.3) a seguir:

$$PM_{NS} = A * Size^{E} * \prod_{i=1}^{n=16} EM_i \qquad (2.3)$$

Onde:

PM_{NS}: Person-months (1PM = 152h) nominal-schedule;

A: Constante baseada na calibração do modelo, conforme Tabela 2.4;

Size: Tamanho do software estimado em KSLOC (*Thousands* of *Source Lines of Code*) ou Pontos de função;

E: conforme equação (2.4);

n: Número de EM's considerado, n=16 para a versão nominal;

EM: Effort Multiplier;

$$E = B + 0.01 * \sum_{j=1}^{5} SF_j \quad (2.4)$$

Onde:

B: Constante baseada na calibração do modelo, conforme Tabela 2.4;

SF: Fator de escala (*Scale Factor*)

O termo NS na equação significa *Nominal-Schedule*, pois exclui o fator multiplicativo denominado SCED, com isso *n=16*. Caso existam restrições impostas ao cronograma, o valor de PM não nominal pode ser calculado considerando o fator multiplicativo SCED, sendo então n=17, conforme equação (2.5):

$$PM \quad = A * Size^{E} * \prod_{i=1}^{17} EM_i \quad (2.5)$$

Segundo o modelo COCOMO II, uma pessoa-mês é a quantidade de tempo que uma pessoa gasta trabalhando no projeto de desenvolvimento de software por um mês. O modelo COCOMO II considera 1PM = 152 horas. Esse número exclui o tempo normalmente dedicado a feriados, férias e folga no final de semana.

Cálculo do Prazo:

O cálculo do prazo de desenvolvimento *TDEV* (*Time to Development)* é definido pela equação (2.6) e é dado em meses.

$$TDEV_{NS} = C * (PM_{NS})^{F} \quad (2.6)$$

Onde:

$TDEV_{NS}$: *Time to Development* em sua versão nominal, em meses;

C: Constante baseada na calibração do modelo, conforme Tabela 2.4;

PM_{NS}: *Person-months* nominal;

F: conforme equação (2.7);

$$F = D + 0.2 * 0.01 * \sum_{j=1}^{5} SF_j = D + 0.2 * (E - B) \quad (2.7)$$

Onde:

D, E, B: Constantes baseada na calibração do modelo, conforme Tabela 2.4;

O termo *SF* significa *Scale Factor* e, para os dois modelos, todos os 5 fatores são considerados.

O termo NS na equação significa *Nominal-Schedule*, pois exclui o fator multiplicativo denominado SCED, que será detalhado mais adiante. Caso existam restrições impostas ao cronograma, o cálculo do TDEV não nominal deve ser feito conforme Equação (2.8):

$$TDEV \ = TDEV_{NS} * \frac{SCED\%}{100} \quad (2.8)$$

Os valores das constantes A, B, C, e D das equações são apresentadas a seguir. O valor de "n" representa o número de fatores multiplicativos de custo considerados em cada versão do modelo COCOMO II.

Tabela 2.4 – COCOMO II: Constantes

Versão do Modelo	A	B	C	D	n
Early Design Model	2.94	0.91	3.67	0.28	6[1]
Post-Architecture Model					16[2]

Fonte: Adaptado de Boehm et al. (2000).

[1] A versão nominal da fórmula exclui o fator multiplicativo SCED, por isso n=6.
[2] A versão nominal da fórmula exclui o fator multiplicativo SCED, por isso n=16.

Os valores da Tabela 2.4 foram obtidos a partir da calibração pelo método *Bayesian* de valores reais de esforço e prazo obtidos de um banco de dados com 161 projetos de empresas dos segmentos comercial, aeroespacial, governo s instituições não governamentais (BOEHM; ABTS; CHULANI, 2000).

Assim como no COCOMO'81, as equações anteriores deixam claro que a principal entrada do modelo é o tamanho (SIZE) do software, que recebe um expoente especial (E), cujo valor é obtido através da agregação dos cinco fatores de escala (SF).

Pela matemática, pode-se concluir que se E<1.0, o projeto exibe economia de escala, ou seja, se o SIZE dobrar, o esforço não aumenta na mesma proporção. A produtividade do projeto aumenta em função do tamanho. A tabela a seguir resumo a influência do fator E no projeto.

Tabela 2.5 – Influência do fator E nos custos e economias do projeto

E < 1.0	**Projeto apresenta economia de escola.**
E = 1.0	As economias e custos em escala estão balanceadas.
E > 1.0	Projeto apresenta aumento de custos conforme aumenta a escala.

Fonte: Adaptado de Boehm et al. (2000).

Para melhor entendimento do expoente E, a seção 2.3.2.1 detalha os fatores de escala.

Cálculo do tamanho médio da equipe:

Uma vez calculados os parâmetros PM e TDEV, é possível então estimar o tamanho médio da equipe – FSP médio (*Full-time equivalent Software Personnel*) – por meio da equação (2.9):

$$FSP_{\bar{x}} = \frac{PM}{TDEV} \tag{2.9}$$

Por meio da distribuição de *Rayleigh*, o modelo sugere o cálculo da distribuição do time ao longo dos meses do projeto (TDEV), FSP aproximado, conforme apresentando na equação (2.10) a seguir:

$$FSP = PM\left(\frac{0.15 * TDEV + 0{,}7t}{0.25 * (TDEV)^2}\right) e^{\frac{(0.15*TDEV+0{,}7t)^2}{0{,}5*(TDEV)^2}} \tag{2.10}$$

Onde *t* é o tempo em meses ao longo do projeto, variando de 0 (zero) até TDEV.

2.3.2.1 Fatores de Escala

Conforme mencionando anteriormente, são 5 os fatores de escala apresentado pelo modelo COCOMO II e eles basicamente determinam as economias e custos do projeto em desenvolvimento. Para facilitar o entendimento e evitar erros na tradução em função dos termos técnicos adotados, os cinco fatores não serão traduzidos.

- **PREC (Precedentedness):** Indica o nível de similaridade do projeto atual com projetos anteriores, ou seja, o nível de experiência da empresa neste projeto (KHATIBI; JAWAWI, 2011).
- **FLEX (Development Flexibility):** Indica o nível de flexibilidade do software em função do processo de desenvolvimento, das restrições de interface, requisitos e prazo.
- **RESL (Architecture / Risk Resolution):** Reflete o resultado da análise de risco (KHATIBI; JAWAWI, 2011). O modelo COCOMO II uma relação deste fator com um percentual de riscos capturados e analisados pela equipe do projeto.
- **TEAM (Team Cohesion):** Indica o nível de integração da equipe e fatores humanos.
- **PMAT (Process Maturity):** Indica o nível de maturidade dos processos da empresa, em alinhamento com o CMMI publicado pela Software Engineering Institute (SEI, 2010) ou estimado.

Cada fator de escola possui 6 níveis, de muito baixo para extremamente alto, e cada nível possui um peso associado, conforme tabela a seguir.

Tabela 2.6 – COCOMO II: Valores dos fatores de escala

Fator de Escala	Muito Baixo	Baixo	Nominal	Alto	Muito Alto	Extra Alto
PREC	Totalmente novo 6.20	Muito novo 4.96	Relativamente novo 3.72	Relativamente familiar 2.48	Muito familiar 1.24	Totalmente familiar 0.0
FLEX	Rigoroso 5.07	Pouca flexibilidade 4.05	Flexibilidade razoável 3.04	Conformidade geral 2.03	Alguma conformidade 1.01	Objetivos gerais 0.0
RESL	Pouco (20%) 7.07	Algum (40%) 5.65	Frequente (60%) 4.24	Generalizado (75%) 2.83	Provável (90%) 1.41	Completo (100%) 0.0
TEAM	Muito difícil interação 5.48	Difícil interação 4.38	Cooperativo 3.29	Bastante cooperativo 2.19	Altamente cooperativo 1.10	Cooperação contínua 0.0
PMAT	SW-CMM-1 Baixo 7.80	SW-CMM-1 Alto 6.24	SW-CMM-2 4,68	SW-CMM-3 3.12	SW-CMM-4 1.56	SW-CMM-5 0.0

Fonte: Adaptado de Boehm et al. (2000).

Para cada nível, a versão completa do método ainda orienta com mais detalhes como selecionar cada um dos níveis mais adequados para o projeto a ser estimado.

Pela Tabela 2.6 pode-se concluir que projetos com todos os fatores de escala extremamente altos resulta em $\sum sf = 0.$ Pela equação (2.11) obtém-se então que $E = 0.91 + 0{,}01 \times (0)$, ou seja, E = 0.91. Conclui-se então pela Tabela 2.5 que E < 1.0, um projeto com todos os fatores de escala extremamente alto apresentará ganhos de escala. Isto significa que se o tamanho do projeto dobrar, seu esforço requerido será menos que o dobro.

Como exemplo, considera-se um projeto com 100KSLOC, com fatores de escala e fatores multiplicativos de custo todos nominais (EM=1):

Exemplo de Cálculo do Esforço:

$$E = B + 0.01 * \sum_{j=1}^{5} SF_j \tag{2.11}$$

Da Tabela 2.4 tem-se que B = 0,91. Da Tabela 2.6 tem-se os valores de cada Scale Factor quando considerado “nominal”. Com isso:

$$= 0{,}91 + 0{,}01 * (3{,}72 + 3{,}04 + 4{,}24 + 3{,}29 + 4{,}68) = 1{,}0997$$

$$PM_{NS} = A * Size^{E} * \prod_{i=1}^{n} EM_i = 2.94 * (100)^{1{,}0997} * 1 = 465 \quad (2.12)$$

Exemplo Cálculo do Prazo:

$$F = D + 0.2 * (E - B) = 0{,}28 + 0{,}2 * (1{,}0997 - 0{,}91) = 0{,}31794 \quad (2.13)$$

$$TDEV_{NS} = C * (PM_{NS})^{F} = 3{,}67 * (465)^{0{,}31794} = 25{,}87 \quad (2.14)$$

Neste exemplo relata-se então um projeto com duração estimada de 25,87 meses e custo de 465 pessoas/mês. O tamanho médio da equipe é medido por PM/TDEV, ou seja, 465/25,87, o que dará aproximadamente 18 pessoas.

O exemplo anterior é válido para os dois modelos do COCOMO II, *Early Design* e *Post-Architecture*, porém em sua versão nominal desconsidera todos os fatores multiplicativos de custo. Para maior precisão e detalhamento do modelo, tais fatores serão detalhados na próxima seção.

2.3.2.2 Fatores Multiplicativos de Esforço (EM)

Conforme mencionado anteriormente, neste ponto do modelo COCOMO II é onde se apresenta a diferença entre *Post-Architecture Model* e *Early Design Model.* O primeiro deles possui 17 fatores multiplicativos de custo e o segundo apenas 5.

2.3.2.2.1 Post-Architecture Model – Cost Drivers

Esta é a versão mais detalhada do modelo, na qual todos os fatores multiplicativos são considerados. Assim como no COCOMO'81, eles estão organizados em quatro categorias:

- **Fatores de Produto:** Nesta categoria estão os fatores que podem causar variação no esforço necessário para desenvolver o software relacionados com as características do produto em desenvolvimento. A complexidade, requisitos para alta confiabilidade ou grandes bancos de dados são exemplos de preocupações deste item.
- **Fatores de Hardware:** Nesta categoria estão os fatores relacionados ao hardware em que o software será executado.

- **Fatores Humanos:** Nesta categoria estão os fatores relacionados à influência que a equipe pode ter no esforço necessário para o desenvolvimento do software.
- **Fatores de Projeto:** Nesta categoria estão os fatores do projeto que exercem influência no esforço, como por exemplo o uso de ferramentas modernas, a localização do time, nível de interação etc.

A tabela a seguir apresenta todos os fatores relacionados com cada categoria:

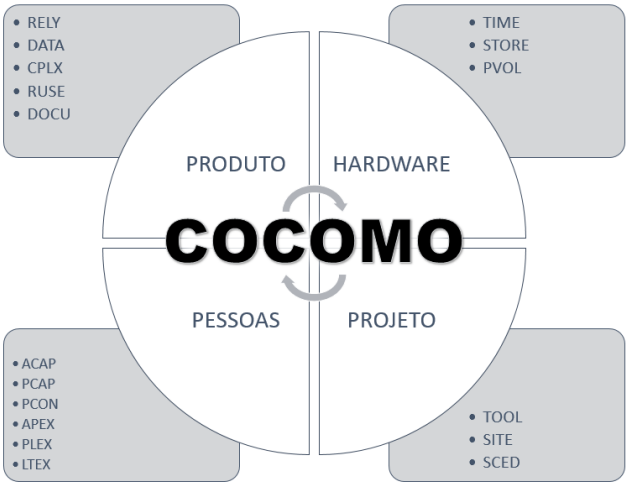
Tabela 2.7 – COCOMO II: Cost Drivers

Categorias	Atributo	Descrição
PRODUTO	RELY	*Required Software Reliability*
	DATA	*Data base Size*
	CPLX	*Product Complexity*
	RUSE	*Developed for Reusability*
	DOCU	*Documentation Needs*
HARDWARE	TIME	*Execution Time Constraint*
	STOR	*Main Storage Constraint*
	PVOL	*Platform Volatility*
PESSOAS	ACAP	*Analyst Capability*
	PCAP	*Programmer Capability*
	PCON	*Personnel Continuity*
	APEX	*Application Experience*
	PLEX	*Platform Experience*
	LTEX	*Language and Tool Experience*
PROJETO	TOOL	*Use of Software Tools*
	SITE	*Multisite Development*
	SCED	*Required Development Schedule*

Fonte: Adaptado de Boehm et al. (2000).

A Figura 2.4 apresenta o diagrama relacionando os fatores e atributos do modelo COCOMO II.

Figura 2.4 – Diagrama de dimensões do COCOMO II



Fonte: Adaptado de Boehm et al. (2000).

Assim como no COCOMO'81, cada um desses atributos determinam um fator de multiplicação que estima o efeito deles no esforço de desenvolvimento do software. Tais atributos são classificados em uma escala de seis níveis, de "muito baixo (*very low*) " a "extremamente alto (*extra-high*)", e para nível é atribuído um peso, conforme apresentado resumidamente na Tabela 2.8 e datalhado nas próximas subseções.

Tabela 2.8 – COCOMO II: Fatores de multiplicação do Esforço

	Classificação					
	Very Low	**Low**	**Nominal**	**High**	**Very High**	**Extra High**
RELY	0.82	0.92	1.00	1.10	1.26	n/a
DATA	n/a	0.90	1.00	1.14	1.28	n/a
CPLX	0.73	0.87	1.00	1.17	1.34	1.74
RUSE	n/a	0.95	1.00	1.07	1.15	1.24
DOCU	0.81	0.91	1.00	1.11	1.23	n/a
TIME	n/a	n/a	1.00	1.11	1.29	1.63
STOR	n/a	n/a	1.00	1.05	1.17	1.46
PVOL	n/a	0.87	1.00	1.15	1.30	n/a
ACAP	1.42	1.19	1.00	0.85	0.71	n/a
PCAP	1.34	1.15	1.00	0.88	0.76	n/a
PCON	1.29	1.12	1.00	0.90	0.81	n/a

Continua

Tabela 2.8 – Conclusão

APEX	1.22	1.10	1.00	0.88	0.81	n/a
PLEX	1.19	1.09	1.00	0.91	0.85	n/a
LTEX	1.20	1.09	1.00	0.91	0.84	n/a
TOOL	1.17	1.09	1.00	0.90	0.78	n/a
SITE	1.22	1.09	1.00	0.93	0.86	0.80
SCED	1.43	1.14	1.00	1.00	1.00	n/a

Fonte: Adaptado de Boehm et al. (2000).

Para cada atributo, o próprio modelo sugere como classificar entre *very low* e *extra-high.* A tabela a seguir exemplifica como seria a aplicação do fator DOCU.

Tabela 2.9 – Fator DOCU: Exemplo de aplicação

Fator	**Classificação**					
	Very Low	Low	Nominal	High	Very High	Extra High
DOCU	Pouca necessidade ao longo do ciclo de vida	Alguma necessidade	Necessidade de tamanho ideal para o ciclo de vida	Excessiva necessidade ao longo do ciclo de vida	Muito excessiva necessidade ao longo do ciclo de vida	n/a
	0.81	0.91	1.00	1.11	1.23	n/a

Fonte: Adaptado de Boehm et al. (2000).

Dentre os fatores multiplicativos de custo apresentados, deve se destacar neste momento o fator *RELY - Required Software Reliability.* De todos os fatores, é o único que trata a questão da confiabilidade e efeito de falha do produto de software, diretamente relacionado com as características de software *Safety-Critical*. Para o fator RELY, o método sugere a seleção do nível de classificação da seguinte maneira, conforme apresentado na Tabela 2.10.

Tabela 2.10 – Fator RELY: Exemplo de aplicação

Fator	Classificação					
	Very Low	Low	Nominal	High	Very High	Extra High
RELY	Leve inconveniência	Baixa, perdas facilmente recuperadas	Moderado, perdas facilmente recuperadas	Grande prejuízo financeiro	Risco para a vida humana	n/a
	0.82	0.92	1.00	1.10	1.26	n/a

Fonte: Adaptado de Boehm et al. (2000).

A aplicação dos níveis para cada um dos fatores apresentados pelo COCOMO II está disposta no APÊNDICE A.

2.3.2.2.2 Early Design Model – Cost Drivers

Conforme já mencionado, a versão simplificada do COCOMO II, chamada de *Early Design Model* compartilha as mesmas equações da versão com completa, *Post-Architecture Model.* Os cinco fatores de escala também são aplicados da mesma maneira. A principal diferença desta versão está no conjunto reduzido de fatores multiplicativos de custo, os *Effort Multipliers.* Pelo fato deste modelo ser utilizado nos estágios iniciais do desenvolvimento de software, quando pouco se sabe do produto a ser desenvolvido, o conjunto completo de 17 fatores multiplicativos da versão *Post-Architecture* foi mapeado em apenas 7 fatores para a versão Early Design, conforme exibido a seguir.

Tabela 2.11 – Early Design e Post-Architecture: Effort Multipliers

Early Design Cost Driver	Paralelo no Post-Architecture Model
RCPX	RELY, DATA, CPLX, DOCU
RUSE	RUSE
PDIF	TIME, STOR, PVOL
PERS	ACAP, PCAP, PCON
PREX	APEX, PLEX, LTEX
FCIL	TOOL, SITE
SCED	SCED

Fonte: Adaptado de Boehm et al. (2000).

Assim como no *Post-Architecture Model*, cada um desses atributos determinam um fator de multiplicação que estima o efeito deles no esforço de desenvolvimento do software. Tais atributos são classificados em uma escala

de seis níveis, de “muito baixo (*very low*) ” a “extremamente alto (*extra-high*)”, e para nível é atribuído um peso.

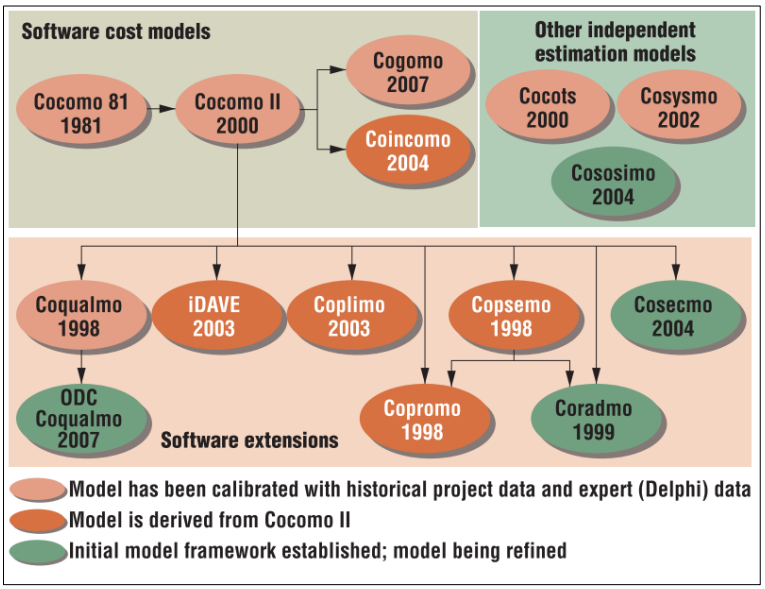
2.3.2.3 Vantagens e Desvantagens

Vantagens:

Uma vez detalhado as duas versões do modelo COCOMO, pode-se afirmar que as mesmas vantagens do COCOMO’81 podem ser consideradas também para o COCOMO II, porém esta última versão cobre e tenta resolver algumas desvantagens da primeira, como por exemplo a consideração do peso de documentação e a própria evolução tecnológica, por ser um método mais moderno. Adicionalmente, o COCOMO II é um método que permite sua constante evolução em função da facilidade de calibração para organizações específicas.

Pode-se reforçar também como principal vantagem do COCOMO II o fato dele ter se tornado um *framework* para a criação de diversos outros modelos, sendo referência e sofrer constante evolução. Na Figura 2.5 a seguir é possível verificar alguns dos desdobramentos mais conhecidos do COCOMO. As datas indicam quando o primeiro artigo sobre o modelo foi publicado.

Figura 2.5 – COCOMO framework de modelos



Fonte: Boehm e Valerdi (2008).

Além desses indicados na Figura 2.5, outras versões também foram exploradas a partir do COCOMO, como o AGILE COCOMO II (SHARMA; BAJPAI; LITORIYA, 2012), COCOMO FUZZY (IDRI; ABRAN; KJIRI, 2000; F.; ALJAHDALI, 2013; SARNO; SIDABUTAR; SARWOSRI, 2015), ADA COCOMO (BOEHM; ABTS; CHULANI, 2000), 2CEE (LUM; MENZIES; BAKER, 2008), NEURAL NETWORK COCOMO (KAUSHIK; SONI; SONI, 2012), EEpred (VELARDE et al., 2016), etc.

Desvantagens:

Mesmo na versão mais atual do modelo COCOMO, existem alguns pontos fracos quando considerado os aspectos de desenvolvimento de software crítico, como é o caso de software embarcados em sistemas aeronáuticos ou aeroespaciais:

- Conforme destacado por Rijwani et al. (2014), o método "*excessivamente simplifica o impacto dos aspectos de segurança*". As preocupações com a criticidade das funções a serem executadas pelo software se desdobram em diversas atividades de análise, revisão, inspeção ou testes que demandam tempo e pessoas.
- O método não considera o peso dos requisitos de software de maneira adequada. A única consideração aparece no fator de escala FLEX. Porém, em caso de software críticos, a quantidade e complexidade dos requisitos é fundamental para determinar o esforço necessário para seu desenvolvimento. Esta limitação também é destaca por Rijwani et al. (2014).
- O método ignora questões de certificação e o envolvimento de agências regulamentadores durante o processo de desenvolvimento. No processo de certificação de software aeronáutico, por exemplo, diversos eventos de auditoria são agendados ao longo do projeto, consumindo tempo e recursos para a correta demonstração do cumprimento de todos os objetivos necessários.

- O método também não considera pontos facilitadores, como por exemplo a possibilidade de qualificação de alguma ferramenta (RTCA INC, 2011b) para executar alguma atividade de maneira automática. Nestes casos, o esforço necessário é não-recorrente, pois uma vez a ferramenta pronta, a atividade que seria recorrente passa a ser automatizada.
- Pode-se destacar ainda que outro ponto facilitador não explorado pelo método é o processo de desenvolvimento baseado em modelo, o MBD (RTCA INC, 2011c). Com o uso de ferramentas modernas, essa técnica permite a geração automática do código fonte baseado no modelo, o que reduz o tempo de desenvolvimento, verificação e revisão de tal código (PAZ; EL BOUSSAIDI, 2016). Além disso, o processo de simulação e testes também é muito favorecido (ESTRADA; SASAKI; DILLABER, 2013). Estas abordagens modernas podem influenciar uma redução do esforço necessário para o desenvolvimento de software.

Por fim, após uma extensa análise e leitura de artigos e livros relacionados às técnicas e métodos de estimativa de esforço em projetos de desenvolvimento de software, alguns pontos podem ser destacados como limitações da literatura em geral:

- Mesmo para o conhecido modelo COCOMO, não existem registros de estudos e resultados obtidos com sua aplicação em abundância nos bancos de dados científicos. Essa constatação também é destacada por Jørgensen e Shepperd (2007) em sua completa e detalhada revisão. Em 2011, porém, Khatibi e Jawawi (2011) publicam os resultados de uma aplicação real do modelo COCOMO II e com os resultados considerados por eles satisfatórios. Uma das hipóteses para tal ausência de estudos desse tipo pode ser o fato de que as empresas que o aplicam não divulgam os resultados por questões comerciais ou propriedade intelectual. Tal hipótese pode ser motivo para um trabalho futuro de pesquisa nas empresas.

- Também destacado por Jørgensen e Shepperd (2007), há uma indicação de que o número atual de pesquisadores ativos e experientes em estimativa de custo de projetos de software é baixo quando comparado com a quantidade de artigos sobre este mesmo assunto. Isso pode tendências as pesquisar sobre esse assunto e causar a ausência de artigos de alta qualidade.

Espera-se com este trabalho reduzir tais limitações em função da integração da norma DO-178C ao modelo COCOMO II, que passa então a complementá-lo com as questões de desenvolvimento de software crítico.

O próximo capítulo descreve o conceito de software crítico, apresenta uma introdução sobre o regulamento aeronáutico produzido pela RTCA, a DO-178C, além de uma breve evolução desde sua primeira versão.

3 SOFTWARE CRÍTICO

A NASA possui um guia para desenvolvimento de software crítico chamado *NASA Software Safety Guidebook* (NASA, 2004a), que define software crítico da seguinte maneira: "*O software é considerado crítico para a segurança se ele controla ou monitora hardware ou software perigoso ou crítico para a segurança*". Em geral tais aplicações são aplicações embarcadas e consideradas de tempo real, ou seja, operam à uma frequência extremamente alta e possuem requisitos de reposta rigorosos nos quais uma simples falha pode gerar eventos catastróficos (MCCORMICK; SINGHOFF; HUGUES, 2011).

Um bom exemplo de software crítico de tempo real é o sistema *Fly-By-Wire* que os aviões modernos possuem. A aeronave provavelmente irá cair se o sistema não completar os cálculos necessários para mover as superfícies de controle dentro de um determinado prazo, por exemplo, dado uma rajada de vento o sistema deve controlar o avião e comandar as superfícies na ordem de milissegundos. Outro exemplo é o sistema de controle de órbita de um satélite, na qual uma falha pode causar a perda do satélite no espaço e o prejuízo comercial de impacto significativo para a instituição.

Posto então a importância deste tipo de software e as características que o tornam mais desafiadores quanto comparados com outras aplicações, as normas de desenvolvimento de software crítico se apresentam como guias de processos e requisitos a serem seguidos, de modo garantir que o desenvolvimento de tais aplicações siga critérios estabelecidos com o nível apropriado de confiabilidade. Neste contexto apresentam-se três das normas mais utilizadas em seus segmentos, a DO-178C (RTCA INC, 2011a) para o setor aeronáutico, a NASA-STD-8719.13C (NASA, 2013) para o setor aeroespacial e a ISO/IEC62304 (BSI, 2006) para setor médico.

Segundo a DO-178C, a determinação da criticidade do software é feita por um processo de avaliação da segurança do sistema, conhecido em inglês como *System Safety Assessment Process* (RTCA INC, 2011a). Basicamente o software será crítico se ele opera dentro de um sistema considerado crítico e

sua falha pode contribuir para um evento catastrófico, perigoso ou de maior impacto (RTCA INC, 2011a).

Diante das diversas normas e guias para o desenvolvimento de software crítico, adota-se como referência neste trabalho e de maneira mais detalhada a DO-178C (RTCA INC, 2011a) pelos seguintes motivos:

- É a norma mais madura e experimentada pela indústria, tendo sua primeira versão publicada em 1980 (RTCA INC, 1980).
- Foi desenvolvida por um comitê multidisciplinar, com participação de membros não apenas da indústria aeronáutica, mas também da espacial, como a NASA e seus fornecedores.
- A DO-178 é citada pela NASA-STD-8719.13 como documento de referência.
- A DO-178C se apresenta como o documento mais completo, englobando os conceitos das outras duas normas.
- Por fim, a DO-178C se apresenta ainda como um documento mais flexível, classificando o software crítico em cinco níveis diferentes.

Deste modo, a próxima seção apresenta uma breve descrição da evolução e dos conceitos da DO-178.

3.1 A NORMA DO-178

Desde sua primeira edição, a norma DO-178 (RTCA INC, 1980) passou por outras três revisões, DO-178A (RTCA INC, 1985), DO-178B (RTCA INC, 1992) e DO-178C (RTCA INC, 2011a). Para evitar repetições de referências, toda esta seção baseia-se nas próprias revisões da norma, exceto quando especificado.

A primeira edição da norma DO-178 foi criada em 1980 pela RTCA - *Radio Technical Commission for Aeronautics* (RTCA INC, 1980) - e foi por muito tempo o primeiro e único padrão para desenvolvimento de software na indústria aeronáutica (LEE; WONG; GAO, 2014). Nesta primeira edição, a norma introduziu o conceito de que o rigor aplicado ao processo de desenvolvimento

de software pode variar conforme a criticidade do sistema em questão, porém como foi escrita de maneira muito alto nível e conceitual, não conseguir cumprir a missão de ser um guia para projetos de desenvolvimento de software crítico.

Já em 1985, a RTCA divulgou a segunda versão da norma, DO-178A (RTCA INC, 1985), que incorporou várias *feedbacks* e lições aprendidas da comunidade de usuários da primeira versão. Ao contrário de um *guideline* vago (LEE; WONG; GAO, 2014), como foi a primeira versão, esta versão mais recente introduz um conjunto de métodos e técnicas para o desenvolvimento de software críticos. Porém, ainda haviam algumas fraquezas e dificuldade de interpretação dos diagramas e objetivos. Em uma época de constante evolução tecnológica, a DO-178A não conseguiu acompanhar a indústria (LEE; WONG; GAO, 2014).

Foi então a partir da década de 90, quando a aviação começou a incorporar cada vez mais a tecnologia de software, que a RTCA se reuniu e modernizou a norma organizando-a em 5 grupos de processos: (1) integração e produção de documentos; (2) problemas relativos ao sistema; (3) desenvolvimento de software; (4) verificação de software; (5) controle de configuração de software e garantia da qualidade. E então a edição DO-178B (RTCA INC, 1992) foi então publicada em 1992 e promoveu grandes mudanças em relação à sua versão anterior.

Umas das grandes evoluções da DO-178B em relação a versão DO-178A foi a alteração da categorização de criticidade do efeito das falhas na tripulação ou no avião, de 3 para 5 níveis. Na versão DO-178A as falhas eram classificadas em (RTCA INC, 1985):

- **Não-essencial (Nível 3):** são funções cujas falhas ou erros de projeto não degradam significativamente o avião ou a capacidade da tripulação.
- **Essencial (Nível 2):** são funções cujas falhas ou erros de projeto podem reduzir a capacidade do avião ou da tripulação em atuar em situações de operação adversa.

- **Crítica (Nível 1):** são funções cujas falhas ou erros de projeto inviabilizar a continuidade do voo ou pouso com segurança.

Já versão DO-178B, as 5 novas categorias de condições de falha são (RTCA INC, 1992):

- **Sem efeito (*No effect)*:** são condições de falhas que não afetam a capacidade de operação do avião tampouco aumentam a carga de trabalho da tripulação.
- **Menor (*Minor)*:** são condições de falhas que não degradam de maneira significativa a segurança da aeronave, mas demandaria ações da tripulação.
- **Maior (*Major)*:** são condições de falhas que podem reduzir a capacidade do avião ou da tripulação em atuar em situações de operação adversa, com aumento significativo da carga de trabalho da tripulação, podendo gerar algum desconforto ou lesões para os passageiros.
- **Perigoso/Severo (*Hazardous/Severe-Major*):** são condições de falhas que podem reduzir a capacidade do avião ou da tripulação em atuar em situações de operação adversa, com aumento muito significativo da carga de trabalho da tripulação, podendo efeitos severos nos ocupantes, incluindo lesões fatais a alguns deles.
- **Catastrófico (*Catastrophic*):** são funções cujas falhas ou erros de projeto podem causar múltiplas fatalidades e perda total da aeronave.

Em função dos efeitos de cada falha descritos anteriormente, a DO-178B classificou então o nível do software conforme a contribuição dele para as potenciais falhas do sistema. A definição de cada nível é detalhada a seguir:

- **Nível A:** Software cujo comportamento anormal causaria ou contribuiria para uma falha da função do sistema resultando em uma condição de falha **catastrófica** para a aeronave.

- **Nível B:** Software cujo comportamento anormal causaria ou contribuiria para uma falha da função do sistema resultando em uma condição de falha **severa** para a aeronave.
- **Nível C:** Software cujo comportamento anormal causaria ou contribuiria para uma falha da função do sistema resultando em uma condição de falha **maior** para a aeronave.
- **Nível D:** Software cujo comportamento anormal causaria ou contribuiria para uma falha da função do sistema resultando em uma condição de falha **menor** para a aeronave.
- **Nível E:** Software cujo comportamento anormal causaria ou contribuiria para uma falha da função do sistema **sem qualquer efeito** sobre a capacidade operacional da aeronave ou sobre a carga de trabalho do piloto.

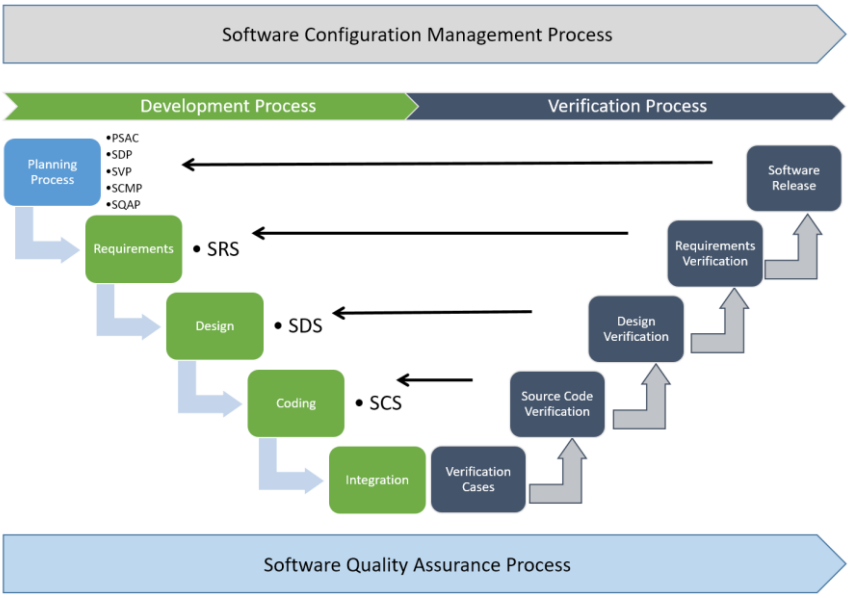
A determinação do nível do software é feita por um processo de avaliação da segurança do sistema, a identificação de potenciais falhas, perdas de função ou mau funcionamento, bem como seus impactos. Segundo a DO-178B (RTCA INC, 1992), o nível de software implica que o nível de esforço necessário para mostrar a conformidade com os requisitos de certificação varia com a categoria de condição de falha.

A DO-178B é primeiramente um documento orientado ao processo do ciclo de vida de desenvolvimento do software (YAN, 2009). Para cada nível de software (A ao E), a norma apresenta então um conjunto de objetivos que devem ser atendidos em cada um dos processos. Os principais processos abordados são:

- Processo de Planejamento do Software
- Processo de Desenvolvimento do Software
- Processo de Verificação do Software
- Processo de Controle de Configuração do Software
- Processo de Garantia da Qualidade do Software
- Processo de Comunicação com a Certificação do Software

A figura abaixo apresenta uma possível representação da interação entre os processos definidos pela DO-178B.

Figura 3.1 – Ciclo de vida do processo da DO-178B



Fonte: Adaptado de RTCA (2011a).

Para cada um dos processos descritos, a norma apresenta um conjunto de atividades e objetivos conforme o nível do software. A Tabela 3.1 deixa claro que o nível A torna o processo mais complexo, sendo necessário o cumprimento de 66 objetivos.

Tabela 3.1 – DO-178B: Quantidade de objetivos por nível de software

Nível	Objetivos
A	66
B	65
C	57
D	28
E	0

Fonte: Adaptado de RTCA (1992).

A Tabela 3.1 também deixa claro que há uma pequena diferença entre os níveis A e B, com a exclusão de apenas um objetivo entre eles. Isso vai ao

encontro da definição de cada um dos níveis, onde em ambos podem ocorrer a perda de vidas humanas.

Os objetivos de cada um dos processos conforme os níveis de software são apresentados em formatos de tabelas, indicando qual objetivo deve ser atendido em cada nível. Como são diversas tabelas e com detalhes específicos da norma, não se faz necessário reproduzi-las aqui, bem como listar cada um dos objetivos. Fica a disposição do leitor a consulta de cada um dos objetivos e tabelas por meio da referência (RTCA INC, 1992).

Com esta abordagem, finalmente a versão DO-178B tornou-se então o padrão adotado amplamente pela indústria e principal meio de demonstração de cumprimento de requisitos de certificação (YAN, 2009).

Mesmo com tamanho sucesso, a DO-178B ainda apresentava algumas fraquezas, conforme apontado por Lee et al. (2014):

- A norma assume um processo de desenvolvimento linear, desde os requisitos até integração e testes, como uma perfeita cascata (*waterfall).*
- A distinção dos níveis de requisitos, se são se sistemas, projetos ou de software, foi por muito tempo motivo de confusão e dúvidas de interpretação.
- Questões como qualificação de ferramentas e automatizações eram ambíguas e complexas.
- A norma foi muito criticada por travar o processo de inovação das empresas.
- O correto entendimento da norma era visto de maneira diferente pelas diversas empresas.

A dificuldade de interpretação levou o departamento de aviação americano FAA - *Federal Aviation Administration* - a criar em 2003 um outro documento apenas sobre orientação de como interpretar a norma, o ORDER 8110.49 (FAA, 2003). No ano seguinte, em 2004, a mesma entidade publicou um outro documento de esclarecimento, o *Job Aid* (FAA, 2004a), com o objetivo apenas

de orientar o processo de revisão, auditoria e expectativas da autoridade de certificação em cada fase do processo de desenvolvimento. A versão DO-178B e estes documentos auxiliares garantiram uma vida longa desta versão e somente após quase 20 anos é que surge então a DO-178C (RTCA INC, 2011a), mais atual e última versão da norma até a presente data.

Enquanto a mudança da versão A para B apresentou uma completa reformulação a norma, a revisão C manteve o a estrutura principal, incorporando apenas correções editoriais, erratas da versão anterior e pequenas alterações para melhor entendimento (LEE; WONG; GAO, 2014). A quantidade de objetivos por nível de software também foi levemente alterada, conforme Tabela 3.2 a seguir.

Tabela 3.2 – DO-178C: Quantidade de objetivos por nível de software

Nível	Objetivos
A	71
B	69
C	62
D	26
E	0

Fonte: Adaptado de RTCA (2011a).

Novamente, os objetivos e tabelas não serão reproduzidos neste trabalho, ficando à disposição do leitor a consulta de cada um dos objetivos e tabelas por meio da referência (RTCA INC, 2011a).

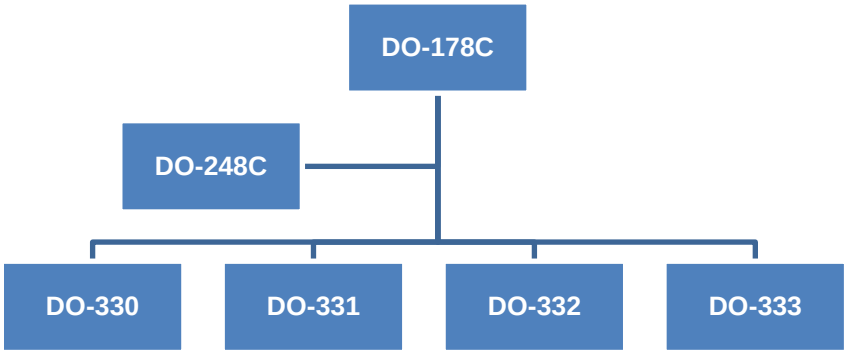
A mudança mais significativa da revisão DO-178C foi a inclusão de quatro suplementos específicos por tecnologia, provendo um guia para aplicação de novas tecnologias no processo de desenvolvimento de software. Os suplementos são os seguintes:

- **DO-330:** Software Tool Qualification Considerations (RTCA INC, 2011b)
- **DO-331:** Model-Based Development and Verification (RTCA INC, 2011c)
- **DO-332:** Object-Oriented Technology and Related Techniques (RTCA INC, 2011d)

- **DO-333:** Formal Methods Supplement to DO-178C (RTCA INC, 2011e)

Além desses, a RTCA publicou junto com a versão DO-178C um documento de esclarecimentos e respostas à dúvidas frequentes, a DO-248C (RTCA INC, 2011f). A Figura 3.2 a seguir apresenta a estrutura da norma e seus suplementos.

Figura 3.2 – Estrutura da DO-178C e suplementos



Fonte: Produção do autor.

Os suplementos apresentados pela RTCA na versão da DO-178C foram de grande importância para a indústria. Mesmo antes de sua publicação, o desenvolvimento baseado em modelos (*MBD – Model-Based Development*) já era uma realidade e com eficiência comprovada (ESTRADA; SASAKI; DILLABER, 2013; PASTOR; CAULÍN; MARTÍNEZ, 2014). Modernas ferramentas de software permitem a geração automática do código fonte com base no modelo, diminuindo significativamente a carga de trabalho no processo de revisão e certificação do software (PAZ; EL BOUSSAIDI, 2016).

O nível de maturidade atingido pela DO-178C e suplementos, além do nível de confiança que este documento conquistou da indústria e agências de certificação, fizeram com que o principal órgão mundial de certificação aeronáutica, a agência americana FAA (*Federal Aviation Administration*) reconhecesse em 2013 todos esses documentos como meios válidos para demonstração de cumprimento das regulamentações para certificação de aspectos de software, por meio da emissão da AC20-115C (FAA, 2013).

Apesar da AC20-115C colocar que a DO-178C é um meio válido, mas não o único meio válido de demonstração do cumprimento dos requisitos de

certificação, não há registros de utilização de quaisquer outros meios com sucesso (YAN, 2009).

Definidos então os dois principais conceitos utilizados ao longo dessa dissertação, estimativa de esforço em projetos de software e o conceito de software crítico, o próximo capítulo apresenta os trabalhos correlatos à essa dissertação, bem como uma análise comparativa entre eles.

4 TRABALHOS CORRELATOS

Estimativa de esforço em projetos de desenvolvimento de software é um assunto amplamente pesquisado pela academia. Jørgensen e Shepperd (2007) identificaram 304 artigos sobre este assunto em uma extensa revisão sistemática. As diversas técnicas de estimativa, bem como seus problemas de imprecisão tem sido objeto de estudo de diversos pesquisadores. Diversos modelos ou variações vem sendo criados e aperfeiçoados ao longo dos anos.

Em Idri et al. (2000), os autores propõem uma adaptação ao COCOMO utilizando lógica Fuzzy; A mesma abordagem foi discutida por Aljahdali (2013) e Sarno et al. (2015). No trabalho dos autores Boehm e Valerdi (2008), são apresentadas as diversas variações do COCOMO ao longo dos anos e os novos métodos que foram derivados deste em função de calibrações específicas para um determinado segmento.

Em Sheta (2006) o autor apresenta duas novas estruturas de modelos para estimar o esforço necessário para o desenvolvimento de projetos de software e aplica estes novos métodos para obter novos valores para os parâmetros apresentados pelo COCOMO. A performance deste novo método é então testada em um banco de dados de projetos da NASA.

Os autores Ramin et al. (2018) buscam a melhoria da precisão do próprio modelo COCOMO analisando cada fator multiplicativo de custo usando algoritmos meta-heurísticos. O mesmo objetivo tem os autores Goyal e Parashar (2018), porém utilizando redes neurais e *Machine Learning Application.*

Um estudo de caso utilizando o COCOMO II e pontos de função para estimativa do esforço do desenvolvimento de software é apresentado pelos autores (CHANDRASEKARAN; VENKATESH KUMAR, 2012). Os autores Khatibi e Jawawi (2011) também apresentam os resultados do uso do COCOMO como método de estimativa de esforço em um projeto real.

De Barcelos Tronto et al. (2008) e Finnie et al. (1997) propõem uma melhor precisão nas técnicas de estimativa de esforço envolvendo redes neurais. Em

Velarde et al. (2016) os autores propõem um método baseado também em linhas de código, uma das entradas do modelo COCOMO.

Em Molokken et al. (2004) os autores apresentam uma pesquisa na indústria norueguesa sobre estimativa de esforço em projetos de desenvolvimento de software. Foram pesquisadas ao todo 52 diferentes projetos de software, abrangendo 18 empresas do setor. Foi descoberto que, em média, os projetos ultrapassam em cerca de 41% do valor estimado. Este valor é similar ao apresentado no estudo de caso B desta dissertação.

Na pesquisa realizada pelos autores Molokken e Jorgensen (2003) conclui-se que o método de estimativa mais utilizado é o baseado em opinião especializada. Tal método, de certo modo, também é utilizado em conjunto com o COCOMO, uma vez que a identificação dos níveis de cada um dos fatores multiplicativos de custo é realizada por meio de opinião especializada.

Em função da complexidade de aplicações *Safety-Critical* e da quantidade de atividades e processos apresentados pela DO-178C, diferentes métricas e abordagens de gerenciamento para tais projetos são estudadas por pesquisados e profissionais da área de engenharia de software. No trabalho apresentado pelos autores Estrada et al. (2013), são apresentadas diferentes abordagens e melhores práticas para desenvolvimento de software baseado em modelos (*MBD – Model Based Development*) em projetos de certificados conforme a norma DO-178C. Essa técnica é encarada por alguns autores como um elemento facilitador do projeto e pode reduzir custos e tempo de desenvolvimento (RIERSON, 2013).

Já os autores Paz e El Boussaidi (2016) apresentam o desenvolvimento baseado em modelos como ferramenta de suporte para projetos aderentes à norma DO-178C.

Finalmente, um conjunto de métricas de conformidade para os objetivos da DO-178C, com o objetivo de evitar atrasos no cronograma de certificação do projeto, foi proposto no trabalho apresentado por Yelisetty et al. (2015). Deve-se mencionar, no entanto, que o esforço de desenvolvimento de software não

era o objetivo de sua pesquisa, mas sim métricas relativas aos aspectos técnicos do produto de software em si.

4.1 Tabela de Correlação

A Tabela 4.1 apresenta a relação dos trabalhos apresentados neste capítulo.

- A coluna Autor apresenta o nome dos autores e ano de cada publicação.
- As demais colunas apresentam o contexto das abordagens utilizadas em cada um dos trabalhos, subdividida em:
    - Adaptação/Especificação (A/E): trabalhos que apresentam algum tipo de adaptação, variação ou comparação ao modelo COCOMO.
    - Métricas (M): trabalhos que apresentam formas de medir a complexidade de projetos de software.
    - Pesquisa (P): trabalhos que apresentam uma revisão sistemática ou avaliação de métodos de estimativa de esforço.
    - *Safety-Critical* (S/C): trabalhos que abordam as complexidades ou técnicas facilitadoras em projetos de software *Safety-Critical,* bem como métricas específicas.
    - Estudo de Caso (E/C): trabalhos que apresentam um estudo de caso de estimativa de esforço em projetos de desenvolvimento de software.

Tabela 4.1 – Comparação dos trabalhos correlatos

Autor	A/E	M	P	S/C	E/C
(JØRGENSEN; SHEPPERD, 2007)			X		
(IDRI; ABRAN; KJIRI, 2000)	X				
(F.; ALJAHDALI, 2013)	X				
(SARNO; SIDABUTAR; SARWOSRI, 2015)	X				
(BOEHM; VALERDI, 2008)	X				
(SHETA, 2006)	X				
(RAMIN SALJOUGHINEJAD; VAHID KHATIBI, 2018)	X				
(GOYAL; PARASHAR, 2018)	X				
(CHANDRASEKARAN; VENKATESH KUMAR, 2012)					X
(KHATIBI; JAWAWI, 2011)					X
(DE BARCELOS TRONTO; DA SILVA; SANT'ANNA, 2008)	X				
(FINNIE; WITTIG; DESHARNAIS, 1997)	X				
(VELARDE et al., 2016)	X	X			
(MOLOKKEN-OSTVOLD et al., 2004)			X		
(MOLOKKEN; JØRGENSEN, 2003)			X		
(ESTRADA; SASAKI; DILLABER, 2013)				X	
(PAZ; EL BOUSSAIDI, 2016)		X		X	
(YELISETTY; MARQUES; TASINAFFO, 2015)		X		X	

Fonte: Produção do autor.

Avaliando a Tabela 4.1, pode-se observar que a maioria dos diferentes autores buscam adaptar ou especificar novos modelos de estimativa de custo, com o foco específico em determinadas necessidades. Observa-se que:

- Adaptação/Especificação (A/E): foi abordado em 10 dos 18 trabalhos pesquisados, sendo que 8 deles também usam o modelo COCOMO como base para adaptação. Os outros 2 trabalhos, utilizam o COCOMO como referência para comparação da estimativa por eles proposta.
- Métricas (M): foi abordado em 3 trabalhos, sendo que dois deles discutem métricas especificas para projetos que seguem a DO-178C.
- Pesquisa (P): foi abordado em 3 trabalhos uma revisão sistemática ou avaliação de métodos de estimativa de esforço, sendo que todos eles citam e colocam o COCOMO como referência.

- *Safety-Critical* (S/C): foi abordado em 3 trabalhos a complexidade ou técnicas facilitadoras para projetos de software *Safety-Critical.* Um deles faz uma análise de custo para projetos que seguem a norma DO-178C.
- Estudo de Caso (E/C): dois trabalhos apresentam um estudo de caso de estimativa de esforço em projetos de desenvolvimento de software, também aplicando o COCOMO II.

Pode-se observar que os trabalhos concentram seus estudos métodos algorítmicos, especificamente o COCOMO II. Os trabalhos que criam ou adaptam um modelo de estimativa de esforço, parte do COCOMO ou o utiliza como referência.

O próximo capítulo apresenta o estudo de caso de aplicação do modelo COCOMO II em um projeto real, de desenvolvimento de software crítico que seguiu os processos definidos pela norma DO-178C para o cumprimento de todos os objetivos de um software crítico nível A.

5 APLICAÇÃO DO MODELO COCOMO II

Relata-se neste capítulo um estudo de caso de aplicação do modelo COCOMO II em um projeto de desenvolvimento de software em uma empresa do setor aeronáutico. Inicia-se pela contextualização da indústria aeronáutica, ambiente no qual o estudo de caso é realizado, passando pela indústria internacional e nacional. São apresentados ainda uma descrição do projeto e o processo de desenvolvimento de software adotado. Finalmente apresentam-se os resultados estimados pelo modelo COCOMO II e o esforço que foi realmente necessário para a realização do projeto em questão.

5.1 A Indústria Aeronáutica

Historicamente poucos países no mundo dominam o processo completo de fabricação de aeronaves. A Segunda Guerra Mundial foi o grande propulsor desta indústria, acelerando o desenvolvimento de tecnologia e os processos de fabricação. Não por caso, até a primeira metade do século XX os principais fabricantes encontravam-se, basicamente, na Europa e Estados Unidos. Neste momento da história, a aviação militar era quase que a totalidade da indústria. Com o término da segunda guerra mundial a aviação civil se desenvolveu paralelamente à aviação militar, dividindo então a indústria em dois grandes ramos, o militar e o civil (MIRANDA, 2007).

Atualmente menos de 20 países dominam o ciclo completo de desenvolvimento de um avião e, entre eles, está o Brasil (MIRANDA, 2007). Os Estados Unidos são os maiores fabricantes de aviões do mundo, tanto no segmente militar quanto civil. Miranda (2007) apresenta que das quatro empresas líderes na aviação militar, três são norte-americanas. São elas a *Boeing, Lockheed* e *Northrop*. A quarta empresa desta lista é a europeia *EADS (European Aeronautic Defense and Space Company)*. Na aviação civil a competição é mais acirrada, tendo sua liderança sendo alterada a cada ano (NEWHOUSE, 2008). Os Estados Unidos, com a Boeing, disputam tal liderança com a Europa (Alemanha, França, Reino Unido e Espanha), representada pela Airbus (NEWHOUSE, 2008). Hoje essas duas empresas dividem quase que sozinhas

o mercado de aviões acima de 120 lugares, tendo suas aeronaves voando em países de todo o mundo.

A disputa por mercado entre Boeing e Airbus é tão grande que muitas vezes, para vencer a concorrência de uma grande venda, são concedidos descontos de até 30% no valor do produto. Não é raro que uma venda seja feita com margem zero de lucro, tamanho é o desconto concedido. A vantagem em vender produtos sem lucro e aumentar a fatia de mercado é garantir no futuro o fornecimento de peças de reposição, uma grande fonte de receita dessas empresas e o momento em que o lucro retorna. Além disso, o “prejuízo” de uma venda de um determinado produto também pode ser compensado pela venda de produtos de sucesso, que dominam o mercado mundial e garantem a receita dessas empresas, como é o caso do 737 para a Boeing, e do A-320 da Airbus. O sucesso desses dois produtos e quantidade de vendas garantidas permitem que essas se deem ao luxo de vender outros produtos com margem zero ou até negativa, com o único objetivo de ganhar fatia de mercado e fidelizar mais um cliente (NEWHOUSE, 2008).

O mercado de aviação regional, ou seja, para aparelhos de até 120 lugares, é dividido basicamente pela canadense Bombardier e pela brasileira Embraer, hoje líder deste segmento. Porém, esse mercado possui também fabricantes de menores como a francesa ATR, além de já possuir novos entrantes, como os chineses com a COMAC, os japoneses com a Mitsubishi e os russos com a Sukhoi.

Nota-se que, com exceção da Embraer, todas as demais empresas estão em países desenvolvidos e tidos de primeiro mundo, ou seja, dispõem de uma infraestrutura mais moderna, governos mais ricos que fazem grandes encomendas, mão de obra especializada e em abundância, além de melhor acesso a recursos financeiros e tecnológicos. Mesmo em um cenário bastante adverso e em desvantagem com relação a seus competidores, a Embraer conseguiu consolidar-se no cenário internacional e ocupar hoje o posto de terceira maior fabricante de aviões do mundo, liderando o mercado de aviação regional (MIRANDA, 2007).

5.2 A Empresa X

A Empresa X é um nome fantasia, de modo a proteger o nome real da empresa a ser retratada nesta seção, por razões comerciais e de propriedade intelectual. A Empresa X possui mais de 20 anos no mercado de desenvolvimento de software no Brasil e se especializou em software embarcados em sistemas de missão para aeronaves de defesa e aplicações de tempo real para sistemas críticos.

Desde sua fundação vem crescendo exponencialmente e atualmente possui cerca de 100 (cem) funcionários, tendo como principais clientes empresas de avião do mundo todo, que subcontratam seus serviços nos projetos de defesa realizados pela Marinha, Exército e Aeronáutica, bem como desenvolvimento de aplicações para sistemas de controle de voo, denominados *Fly-by-wire*.

Além de software embarcado, completam o portfólio da empresa sistemas de simulação e software desktop para o planejamento e *debriefing* (relatório de resultado da missão) de missões táticas realizadas pelas forças armadas.

5.3 O Projeto do Software Embarcado

Atualmente a Empresa X vêm se especializando no desenvolvimento de software embarcado para as aeronaves de seus clientes. A iniciativa começou no início dos anos 2000 com o projeto de desenvolvimento de software de missão para uma aeronave do segmento de defesa da Força Aérea de um determinado país.

Um software de grande complexidade, porém não crítico para segurança de voo, o que simplifica o processo de desenvolvimento e o envolvimento de agências de certificação. Em meados de 2005 que empresa iniciou um projeto de pesquisa para desenvolver software embarcado em aeronaves civis, o que torna o processo muito mais complexo e há então a exigência de cumprimento com a norma DO-178, na época em sua revisão B.

O projeto de pesquisa foi considerado um sucesso, dando então sequência para o desenvolvimento de um software de aplicação real, para ser embarcado em uma das aeronaves da empresa cliente. O primeiro programa escolhido foi

uma aeronave de defesa de um grande cliente e a primeira aplicação escolhida foi o software de controle de voo do sistema de *Fly-By-Wire*. Este programa, por ser militar, possui um prazo de desenvolvimento mais flexível, com baixa pressão do mercado para uma data de lançamento, cenário ideal para se mitigar os riscos inerentes ao desenvolvimento de um software complexo.

Com a experiência adquirida no desenvolvimento do software embarcado neste programa, foi então que Empresa X dominou a tecnologia e o processo e assumiu também o desenvolvimento de software embarcado do sistema de *Fly-By-Wire* de um programa civil, aqui denominado Harpia (nome fantasia por razões de propriedade intelectual), o que seria o primeiro avião comercial com software desenvolvido pela empresa e cumprindo todos os objetivos da DO-178C.

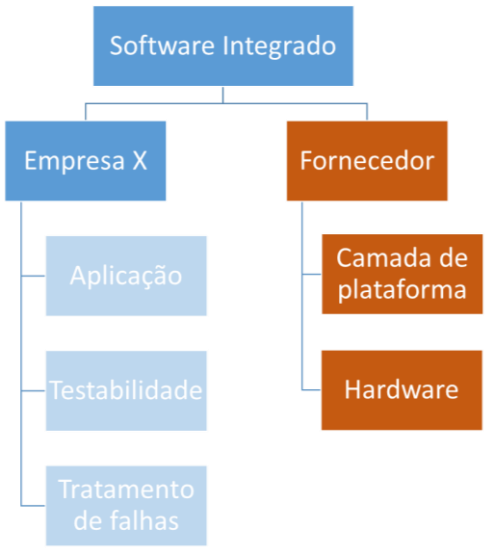
O primeiro projeto, por ser de aplicação militar, não pode ser objeto de estudo de caso deste trabalho. Deste modo, detalha-se o projeto de desenvolvimento de software embarcado do sistema de *Fly-By-Wire* do programa Harpia. O ciclo completo de desenvolvimento do projeto foi de junho de 2013 até dezembro de 2017, totalizando 4,5 anos e consumindo no pico 45 engenheiros de software por tempo integral.

5.4 Escopo do Projeto

O projeto de desenvolvimento de software embarcado do sistema de *Fly-By-Wire* do programa Harpia foi dividido em duas empresas. Um fornecedor norte-americano foi contratado para o desenvolvimento do hardware e do software de baixo nível, responsável pelas funções básicas de interface com o hardware e gerenciamento das entradas e saídas de dados através dos canais de comunicação digital.

A Empresa X, por sua vez, ficou responsável pelo desenvolvimento da aplicação da lei de controle, das funções de testabilidade e do tratamento de falhas. A Figura 5.1 a seguir exibe a estrutura do projeto e a Tabela 5.1 detalha a responsabilidade de cada uma das partes no projeto.

Figura 5.1 – Estrutura do Projeto



Fonte: Produção do autor.

Tabela 5.1 – Escopo Empresa X e Fornecedor

EMPRESA X	FORNECEDOR
Aplicação: Software responsável pela lei de controle e comandos para atuação das superfícies de sustentação da aeronave. **Testabilidade:** Software responsável por funções exclusivas de testabilidade, como monitoramento de variáveis e manipulação de memória. **Tratamento de falhas:** Software responsável pelo monitoramento de falhas críticas e registro na memória não-volátil.	**Camada de plataforma:** Software responsável pela configuração do processador, interface com o hardware e gerenciamento dos barramentos de comunicação digital, além da conversão para dados de engenharia. **Hardware:** especificação de desenvolvimento do hardware, bem como definição do processador e capacidade de memória.

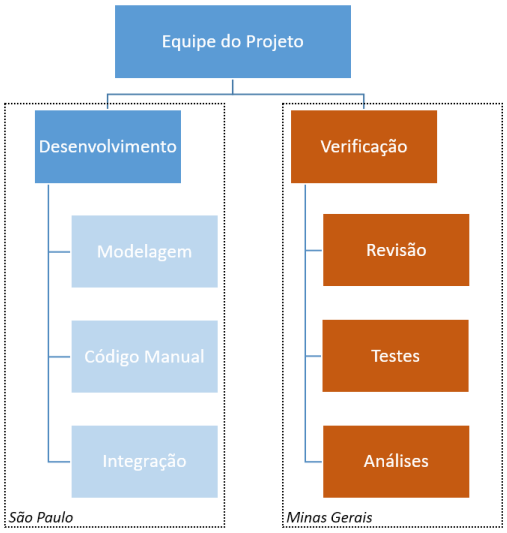
Fonte: Produção do autor.

É possível observar que o fornecedor é responsável por uma camada genérica do software, ou seja, independente da aplicação final em si, o que torna a propriedade intelectual do software final protegida e sob domínio do

conhecimento limitado à Empresa X. Deste modo, o escopo deste estudo de caso concentra-se então no pacote de software destinado à Empresa X.

Por ser um software crítico, de aplicação civil e responsável pelo controle da aeronave, o processo de certificação de tal software requer o cumprimento de todos os objetivos estabelecidos pela norma DO-178C, ou seja, nível A, conforme definido pelo FAA (*Federal Aviation Administration*) na AC 20-115C (FAA, 2013). Conforme apresentado no capítulo 3, o processo estabelecido pela norma define objetivos de desenvolvimento e verificação. Para isso, a Empresa X organizou o time do projeto conforme apresentado na Figura 5.2:

Figura 5.2 – Organização do time



Fonte: Produção do autor.

A Figura 5.2 exibe não apenas a organização lógica do time, mas também física. É possível notar que os times de desenvolvimento estão separados fisicamente, estando o time de desenvolvimento no Estado de São Paulo e o de verificação em Minas Gerais.

Esse distanciamento é muito bem visto pelas autoridades de certificação, pois endereça a preocupação de que o time de desenvolvimento pode exercer alguma influência nos testes, fazendo com que eles sejam mais orientados ao sucesso. A independência de pessoas quanto à elaboração dos artefatos de desenvolvimento e verificação é um dos objetivos da DO-178C e segregação física contribuí para tal atingimento. Além disso, estudos mostram que o

desenvolvimento distribuído não contribui para a queda da qualidade ou surgimento de falhas no produto (BIRD et al., 2009).

Além da organização lógica e física da equipe do projeto, a Figura 5.2 também introduz as principais atribuições de cada um dos times, conforme detalhado na Tabela 5.2:

Tabela 5.2 – Papel dos times

DESENVOLVIMENTO	VERIFICAÇÃO
Modelagem: A atividade de modelagem consiste no desenvolvimento de software baseado em modelos, conforme padronizado pela DO-331 (RTCA INC, 2011c). **Código Manual:** A atividade de código manual consiste no desenvolvimento de software baseado no paradigma tradicional, ou seja, baseado em requisitos textuais, conforme padronizado pela DO-178C (RTCA INC, 2011a). **Integração:** A atividade de integração consiste no carregamento e testes de maturidade do software no computador real no qual o software será executado. Essa atividade visa antecipar possíveis problemas e corrigi-los antes do software ser liberado para o uso.	**Revisão:** A atividade de revisão consiste na verificação formal dos artefatos de desenvolvimento pelo time de verificação. Neste momento o software é revisado contra os padrões e requisitos dos quais tal software foi desenvolvido. **Testes:** A atividade de testes consiste na elaboração e revisão dos casos de testes que são desenvolvidos contra os requisitos. O objetivo é que todos os requisitos sejam devidamente testados. **Análise:** A atividade de análise consiste em verificar as características do software quanto ao seu tempo de execução, consumo de memória, pilha, mapa de endereçamento etc.

Fonte: Produção do autor.

Observa-se pela Tabela 5.2 que cada time possui seu papel e responsabilidade muito bem definido e com critérios de transição que devem ser claros. Obviamente que todas as evidências de testes e verificações, bem como os artefatos de desenvolvimento, devem ser devidamente controladas, de modo que seja fácil identificar modificações entre uma versão já verificada e outra recém modificada.

5.5 Estimativa de Esforço do Projeto

Durante a fase de planejamento do projeto, no início do desenvolvimento, foi adotado o modelo COCOMO II para auxiliar na estimativa de esforço e tempo de desenvolvimento necessários para a conclusão do software.

O modelo COCOMO II foi então aplicado em sua essência, sem a realização de calibração dedicada para a empresa, e com os fatores de escala e fatores multiplicativos de custo sendo respondidos sob a luz do pacote de software destinado apenas à Empresa X, sem considerar o pacote destinado ao fornecedor.

Para a estimativa do tamanho do software, o parâmetro KLOC do modelo COCOMO II, foi estimado o valor de 140KLOC. Este valor foi baseado no método de opinião especializada e analogia, utilizando valores de projetos anteriores similares. Além disso, algumas premissas foram assumidas para a definição deste valor, são elas:

- A quantidade de linhas de código por função foi estimada baseada em um código protótipo previamente desenvolvido.
- O software em questão teria um crescimento linear em considerando o número de funções previstas para serem implementadas.
- A complexidade de se desenvolver software baseado em modelo é similar à baseada no paradigma tradicional, baseada em requisitos textuais e código desenvolvido manualmente.

Com tais premissas sendo consideradas, os fatores de escala do modelo COCOMO II foram analisados por um time de engenheiros de software experientes e selecionados conforme indicado na Tabela 5.3. Uma vez

identificado o nível para cada fator de escala, seu respectivo valor é então obtido automaticamente conforme apresentado pelo próprio método. Como referência, a Tabela 2.6 apresenta todos os valores para cada seleção dos fatores de escala.

Tabela 5.3 – Fatores de Escala –Escopo Empresa X

Fatores de Escala	Seleção	Valor
PREC (Precedentedness)	Nominal	3,72
FLEX (Development Flexibility)	Muito Baixo	5,07
RESL (Architecture / Risk Resolution)	Alto	2,83
TEAM (Team Cohesion)	Nominal	3,29
PMAT (Process Maturity)	Muito Alto	1,56

Fonte: Produção do autor.

Pela própria natureza do modelo COCOMO II, a seleção dos fatores de escala do projeto é algo subjetivo. Deste modo, se faz necessário detalhar as premissas e considerações levadas em conta pela equipe do projeto durante a seleção.

O parâmetro **PREC**, que indica o nível de similaridade do projeto atual com projetos anteriores, foi selecionado como nominal, uma vez que a empresa possui apenas um projeto similar previamente desenvolvido, porém passou por uma etapa de pesquisa e amadurecimento muito intensa antes de, de fato, entrar no desenvolvimento específico. Este balanceamento entre anos de pesquisa, mas apenas um projeto anterior, levou a equipe a selecionar a opção nominal.

O parâmetro **FLEX**, que indica o nível de flexibilidade do software em função do processo de desenvolvimento, das restrições de interface, requisitos e prazo, foi selecionado como muito baixo, uma vez que necessidade de certificação de tal software, bem como o cumprimento das normas DO-178C e DO-331 tornam o processo pouco flexível. Além disso, o projeto nasceu com um cronograma apertado, pouca flexibilidade para ajuste do prazo em função da data de necessidade de lançamento do produto no mercado, além do fato de envolver o desenvolvimento por parte de um fornecedor externo, o que limita as alterações de interface.

O parâmetro **RESL**, que reflete o resultado da análise de risco, foi selecionado como alto (aproximadamente 75% dos riscos mapeados), uma vez que a empresa possui um bom gerenciamento de risco, porém o baixo índice de precedência e experiência anterior, identificado pelo fator PREC, pode introduzir riscos ocultos e não mapeados.

O parâmetro **TEAM**, que indica o nível de integração da equipe e fatores humanos, também foi selecionado como nominal, uma vez que na fase de planejamento do projeto o time ainda não estava todo formado, as pessoas não se conheciam ou eram novas na empresa, o que dificultou a análise deste parâmetro. Sendo assim, decidiu-se pelo nominal para evitar algum tipo de influencie positiva ou negativa do parâmetro no cálculo da estimativa.

Finalmente, o parâmetro **PMAT**, que indica o nível de maturidade dos processos da empresa, foi selecionado como muito alto em função da experiência acumulada e da maturidade dos processos desenvolvidos e experimentados nos projetos de pesquisa anteriores ao desenvolvimento deste produto.

Uma vez definidos os fatores de escala, foi possível então calcular o parâmetro "E" conforme equação (2.4):

$$E = B + 0.01 * \sum_{j=1}^{5} SF_j$$

*E = 0,91 + 0,01 * (3,72+5,07+ 2,83+3,29+1,56) = 1,0747.*

Observa-se então que o projeto apresenta E > 1,0. Conforme Tabela 2.5, tem-se então um projeto com aumento de custos conforme aumento de escala. Isto significa que se o tamanho do projeto dobrar, por exemplo, seus custos mais que dobram.

Continuando a aplicação do método, o mesmo grupo de pessoas dedicou-se então na identificação dos fatores multiplicativos de esforço do projeto. Eles foram todos analisados e respondidos conforme Tabela 5.4. Similar aos fatores de escala, uma vez identificado o nível para cada fator multiplicativo de esforço, seu respectivo valor é então obtido automaticamente conforme apresentado

pelo próprio método. A Tabela 2.8 apresenta todos os valores para cada seleção dos fatores multiplicativos de esforço.

Tabela 5.4 – Fatores Multiplicativos de Custo

Cost Drivers (as in COCOMO II)	Seleção	Valor
Product Attributes		
Required Software Reliability (RELY)	Muito Alta	1,26
Database Size (DATA)	Nominal	1
Product Complexity (CPLX)	Muito Alta	1,34
Developed for Reusability (RUSE)	Nominal	1
Documentation Match to Life-Cycle Needs (DOCU)	Muito Alta	1,23
Computer Attributes		
Execution Time Constraint (TIME)	Alta	1,11
Main Storage Constraint (STOR)	Nominal	1
Platform Volatility (PVOL)	Nominal	1
Personnel Attributes		
Analyst Capability (ACAP)	Baixa	1,19
Programmer Capability (PCAP)	Muito Alta	0,76
Personnel Continuity (PCON)	Muito Alta	0,81
Applications Experience (APEX)	Baixa	1,1
Platform Experience (PLEX)	Nominal	1
Language and Tool Experience (LTEX)	Nominal	1
Project Attributes		
Use of Software Tools (TOOL)	Muito Alta	0,78
Multisite Development (SITE)	Muito Baixa	1,22
Required Development Schedule (SCED)	Baixa	1,14

Fonte: Produção do autor.

Assim como nos fatores de escala, a seleção dos fatores multiplicativos de custo do projeto também é algo subjetivo. Deste modo, novamente se faz necessário detalhar as premissas e considerações levadas em conta pela equipe do projeto durante a seleção.

O parâmetro **RELY**, que mede o efeito de uma falha de software na função em que ele deve executar durante um período de tempo, foi selecionado como muito alto, pois por ser um software de controle de voo, os requisitos e confiabilidade e segurança do produto são dos mais elevados. Com isso, este

certamente é um dos fatores que contribuem para a multiplicação dos custos do projeto.

O parâmetro **DATA**, que relaciona o efeito de testes de grandes quantidades de dados no desenvolvimento do produto, foi selecionado como nominal de modo a não influenciar na estimativa, uma vez que pouco se conhecia ainda a quantidade de dados a testes que seriam necessários.

O parâmetro **CPLX**, que mede a complexidade do produto dividida em cinco áreas: (1) controle de operação; (2) operações computacionais; (3) operações com dispositivos; (4) gerenciamento de dados e (5) operações de interface com usuário; foi selecionada como muito alta. Apesar da aplicação ser extremamente complexa, seguindo a orientação do método o projeto em questão não possui as características que o qualificariam ser selecionado como extremamente alto, uma vez que não existe processamento distribuído ou interface gráfica.

O parâmetro **RUSE**, que relaciona o esforço adicional que será necessário para desenvolver software com intenção de ser reutilizado por projetos futuros, foi selecionado como nominal para não interferir no cálculo da estimativa, uma vez que não foi planejado nenhum tipo de desenvolvimento para ser reusado no futuro.

O parâmetro **DOCU**, que é avaliado em termos da necessidade de documentação ao longo do ciclo de vida do projeto em desenvolvimento, foi selecionado como muito alto em função da quantidade de documentos necessários ao longo do ciclo de vida de um projeto de desenvolvimento de software nível A, conforme a DO-178C (RTCA INC, 2011a).

O parâmetro **TIME**, que é a medida relacionada com a restrição para o tempo de execução do software, foi selecionado como alta, uma vez que já havia um requisito claro de que o software deveria consumir no máximo 70% do tempo de processamento no momento da certificação.

O parâmetro **STOR**, que representa o grau de restrição do principal meio de armazenamento de dados do sistema, foi selecionado como nominal de modo

a não influenciar na estimativa, uma vez que pouco se conhecia ainda a quantidade de dados que seriam necessários.

O parâmetro **PVOL**, que é medido sob o ponto de vista da quantidade de mudanças de plataforma, foi selecionado como nominal de modo a não influenciar na estimativa, uma vez que o software de plataforma é de responsabilidade do fornecedor.

O parâmetro **ACAP**, que analisa a habilidade do analista, eficiência e rigor, e habilidade de comunicação e cooperação com os demais membros do time no que diz respeito aos requisitos, arquitetura e detalhamento do projeto, foi selecionado como baixo uma vez que a equipe, conforme já mencionado, estava ainda em formação e diversos membros do time com pouca experiência.

O parâmetro **PCAP**, que analisa a habilidade do analista, eficiência e rigor, e habilidade de comunicação e cooperação com os demais membros do time no que diz respeito à programação, foi selecionado como muito alta, pois mesmo se tratando de um time novo e recém-formado, todos possuem experiência prévia com programação e são da área de software,

O parâmetro **PCON**, que é selecionado em termos do *turnover* anual da equipe do projeto, ou seja, o % de pessoas que deixam o time ao longo do desenvolvimento, foi selecionado como muito alto, o que neste caso, deve ser encarado como redução de esforço no projeto, uma vez que o índice de *turnover* da empresa é menor que 3% ao ano.

O parâmetro **APEX**, que mede o nível de experiência na aplicação que possui o time de desenvolvimento do software em questão, foi selecionado como baixo pelos mesmos motivos citados anteriormente, ou seja, a equipe estava ainda em construção e com diversos membros recém contratados.

Os parâmetros **PLEX** e **LTEX**, que consideram a influência na produtividade do nível de experiência do time de desenvolvimento na plataforma em questão, foram selecionados como nominal de modo a não influenciar na estimativa, uma vez que o software de plataforma é de responsabilidade do fornecedor.

O parâmetro **TOOL**, que considera o uso de ferramentas sofisticadas no desenvolvimento do software, foi selecionado como muito alto, em função do projeto estar utilizando ferramentas sofisticadas e qualificadas pela DO-330, como por exemplo gerados de código automático. O uso de tais ferramentas de fato deve ser tratado como um fator de redução de custo, uma vez que seu valor é < 1.0, ou seja, reduz o valor final do produto de todos os fatores multiplicativos de custo.

O parâmetro **SITE**, que mede o efeito da globalização e da distribuição dos times de desenvolvimento, foi selecionado como muito baixo pelo fato do desenvolvimento envolver três diferentes localizações (São Paulo, Minas Gerais e Estados Unidos) e as ferramentas de comunicação principalmente entre Brasil e Estados Unidos serem limitadas.

Por fim, o parâmetro **SCED**, que considera as restrições de cronograma impostas sob o time de desenvolvimento de software, foi selecionado inicialmente como baixo (segundo o COCOMO II, 85% do cronograma ideal), em função do cronograma do projeto ser desafiador e considerado menor que o ideal. Porém, como será detalhado mais adiante, este fator não será considerado na estimativa.

Definidos então todos as entradas necessárias, é possível calcular a estimava de quantidade de pessoas/mês, conforme Equação (2.3). Neste primeiro momento, calcula-se o valor nominal, ou seja, excluindo-se o parâmetro SCED do produto de todos os fatores multiplicativos de custo. Deste modo, n=16 na Equação (2.3).

$$PM_{NS} = A * Size^{E} * \prod_{i=1}^{n=16} EM_i$$

Sendo que, da Tabela 5.4 obtêm-se os valores de cada fator multiplicativo de esforço, resultando no produto a seguir:

$$\prod_{i=1}^{16} EM_i = 1{,}767$$

A constante é “A” é obtida pela Tabela 2.4. O parâmetro SIZE foi dimensionado em 140KSLOC. O Fator “E” é calculado conforme Equação 2.4. Deste modo:

PM_{NS} = 2,94 * (140)^1,0747 * (1,767) = **1052 pessoas/mês.**

Cabe aqui lembrar ao leitor que o parâmetro PM se refere ao valor estimado de *Person-Months* (Pessoas por mês). Um PM é a quantidade de tempo que uma pessoa dedica trabalhando no desenvolvimento de software durante um mês. O modelo COCOMO adota como valor de referência o fator de 152 horas por PM (BOEHM et al., 2000), ou seja, 1PM = 152 horas.

Calcula-se então neste momento o valor de PM não nominal, ou seja, incluindo o fator multiplicativo de custo SCED, conforme equação (2.5):

$$\prod_{i=1}^{17} EM_i = 2{,}015$$

PM = 2,94 * (140)^1,0747 * (2,015) = 1200 pessoas/mês.

O aumento de 1052PM para 1200PM evidencia o efeito do achatamento do cronograma em função do fator SCED.

Conforme Equação (2.6), calcula-se então o tempo de desenvolvimento TDEV, ainda de maneira nominal:

$$TDEV_{NS} = C * (PM_{NS})^F = 3{,}67 * (1052^F)$$

Onde, F = 0,28 + 0.2 * (1,0747 - 0,91) = 0,31294.

Deste modo:

$TDEV_{NS}$ = 3,67 * (1052) ^{0,31294} = **32,3 meses.**

Porém, considerando o parâmetro SCED apontando para 85% do cronograma original, obtém-se um novo TDEV, desta vez não nominal, conforme equação (2.8):

TDEV = 32,3 meses x 0,85 = 27,5 meses.

Uma vez calculados os parâmetros PM e TDEV, é possível então calcular o tamanho médio da equipe, considerando o cronograma nominal:

$EQUIPE_{NS}$ = PM_{NS} / $TDEV_{NS}$ = 1052 / 32,3 = **32 pessoas.**

Considerando o efeito o parâmetro SCED também no tamanho médio da equipe, tem-se então a seguinte quantidade de pessoas:

EQUIPE = PM / TDEV = 1200 / 27,5 = 44 pessoas.

O aumento do número de pessoas de 32 para 44 reflete a redução do prazo em função do parâmetro SCED, com isso é necessária uma equipe maior para executar o mesmo trabalho. Deste modo, a equipe do projeto, decidiu então assumir como estimativa apenas os valores nominais calculados. Tal decisão se deu pelos seguintes motivos:

- O projeto ainda estava em uma fase de grandes incertezas, com diversas indefinições no escopo, deste modo assumir um cronograma de apenas 85% do ideal poderia ser precipitado;
- Caso tal redução do cronograma não se confirmasse, considerar o parâmetro SCED na estimativa poderia levar à contração de mão de obra em excesso, causando prejuízo e descontentamento na equipe;
- Uma possível necessidade de mão de obra adicional, caso fosse confirmada, poderia ser facilmente satisfeita com a liberação de horas adicionais de trabalho ou contratação de recursos terceirizados de forma pontual.

A Tabela 5.5 sumariza então o resulta de todos os principais parâmetros estimados.

Tabela 5.5 – Valores Estimados pelo COCOMO II

Parâmetro	**Esforço Estimado**
KLOC	140
PM_{NS} (Pessoa-mês)	1052
$TDEV_{NS}$ (Tempo de desenvolvimento)	32,3 meses
Tamanho médio da Equipe	32 pessoas

Fonte: Produção do autor.

Por meio da distribuição de Rayleigh, o próprio modelo sugere o cálculo do FSP (*Full-time equivalent Software Personnel*), que se trata da alocação da equipe ao longo dos meses do projeto de maneira aproximada, conforme

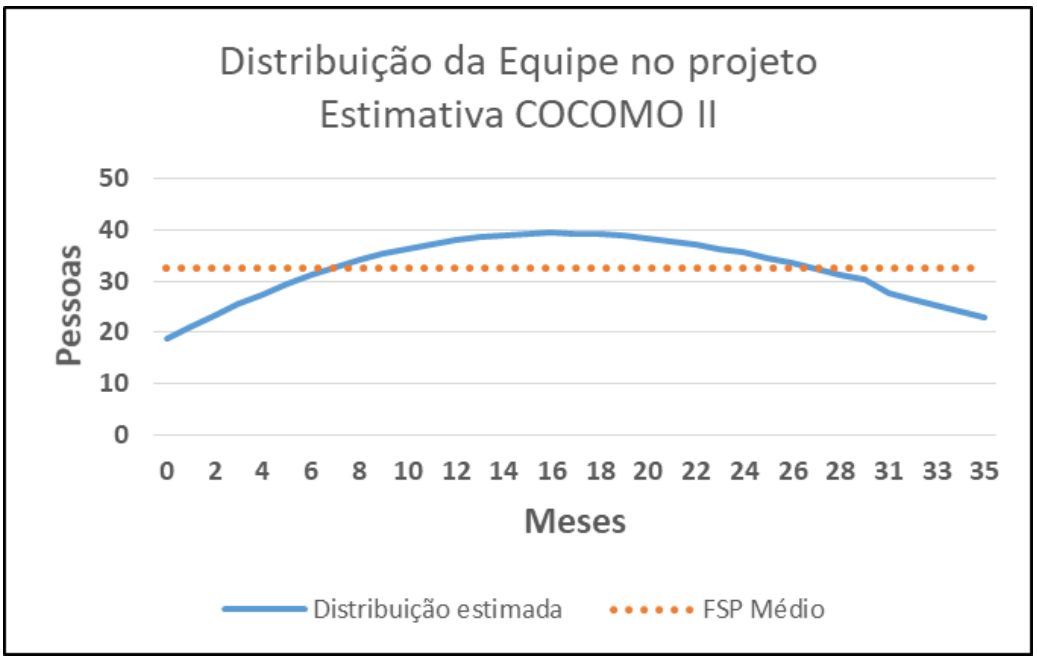
apresentando na equação (2.10) na seção 2.3.2. Para facilitar a leitura, a equação está repetida abaixo:

$$FSP = PM\left(\frac{0.15 * TDEV + 0{,}7t}{0.25 * (TDEV)^2}\right) e^{\frac{(0.15*TDEV+0{,}7t)^2}{0{,}5*(TDEV)^2}}$$

Onde *t* é o tempo em meses ao longo do projeto.

Considerando então os valores já calculados para PM e TDEV, e aplicando então a equação de Rayleigh para valores de t = 0 (zero) até t = 35 (trinta e cinco), obtém-se finalmente a distribuição estimada apresentada na Figura 5.3. A distribuição calculada mês a mês está disponível no APÊNDICE B.

Figura 5.3 – Distribuição estimada da equipe – escopo empresa X



Fonte: Produção do autor.

A Figura 5.3 apresenta o tamanho médio de 32 pessoas, conforme calculado anteriormente, porém sugere um período de pico, chegando até 40 pessoas. Esse tipo de análise antecipada ajuda os gestores de projetos a preverem orçamento para hora extra ou contratação de mão de obra terceirizada em momentos pontuais do desenvolvimento.

O modelo COCOMO prevê ainda a distribuição da equipe ao longo das fases do projeto. Utilizando como referência o modelo de desenvolvimento de

software conhecido como *Waterfall*, a distribuição teórica da equipe ficaria conforme apresentado na Tabela 5.6 a seguir.

Tabela 5.6 – Distribuição da Equipe por Fase

Fase	Esforço %	Tamanho da Equipe
Planos e Requisitos[1]	7%	2
Projeto Detalhado	17%	6
Codificação	52%	17
Integração e Testes	31%	10
Transição[1]	12%	4

Fonte: Produção do autor.

Nota 1: Os percentuais das fases “Planos e Requisitos” e “Transição” são adicionais ao esforço estimado pelo COCOMO. Deste modo, a somatória ultrapassa os 100%.

Uma vez apresentado como foi a estimativa de esforço do projeto, a próxima seção apresenta como de fato foi a demanda de esforço do projeto, bem como os desvios quanto à estimativa.

5.6 Esforço Requerido do Projeto

Uma vez apresentado o esforço estimado do projeto, esta seção descreve o esforço real que foi necessário para o desenvolvimento do software objeto deste estudo de caso.

Conforme apresentado na seção anterior, o modelo COCOMO II utiliza como referência o modelo de desenvolvimento de software conhecido como *Waterfall* (BOEHM et al., 2000). O projeto em questão seguiu o processo estabelecido pela norma DO-178C (RTCA INC, 2011a). Deste modo, antes de se comparar o real versus o estimado, é preciso relacionar as fases dos dois processos de desenvolvimento, conforme apresentado na Tabela 5.7 a seguir.

Tabela 5.7 – Relação das fases *Waterfall* e processos DO-178C

Fase do modelo Waterfall	Processo da DO-178C
Planos e Requisitos	Processo de Planejamento Processo de Requisitos
Projeto Detalhado	Processo de Design
Codificação	Processo de Codificação
Integração e Testes	Processo de Integração Processo de Verificação
Transição	-

Fonte: Produção do autor.

Adotando então os processos estabelecidos pela DO-178C, o projeto demandou a seguinte quantidade de pessoas, conforme Tabela 5.8 a seguir.

Tabela 5.8 – Esforço real exigido pelo projeto

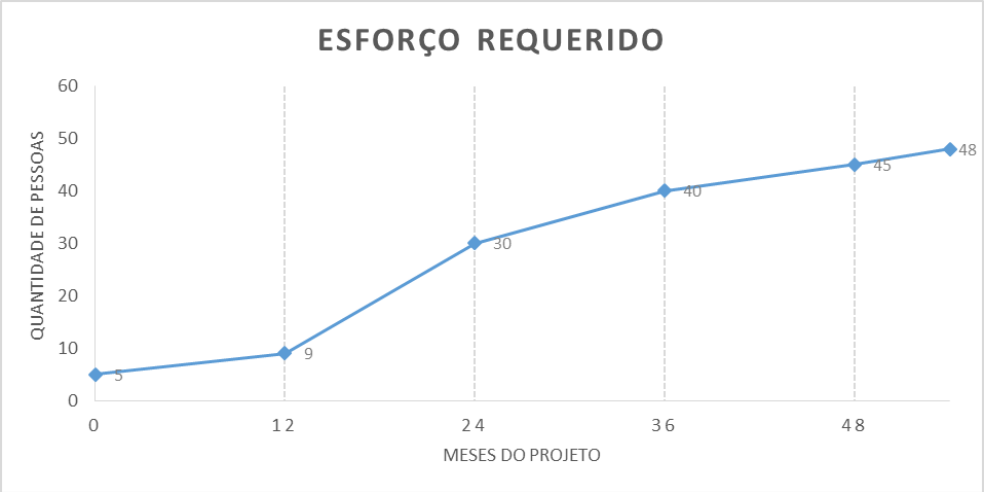
Processo da DO-178C	Quantidade de Pessoas
Processo de Planejamento	5
Processo de Requisitos	9
Processo de Design	30
Processo de Codificação	40
Processo de Integração	45
Processo de Verificação	48

Fonte: Produção do autor.

A Tabela 5.8 evidencia que o pico de quantidade de pessoas se deu na fase de verificação, chegando à um total de 48 pessoas. A necessidade maior neste processo vai ao encontro dos objetivos exigidos pela DO-178C, dos quais 43 dentre os 71 objetivos são relacionados com atividades de verificação de software, ou seja, a grande maioria.

Com relação ao tempo de desenvolvimento, o projeto demandou um total de 4,5 anos (54 meses) para ser desenvolvido até o final, desde a concepção até à certificação. A Figura 5.4 a seguir apresenta graficamente a quantidade de pessoas exigidas pelo projeto ao longo dos 54 meses de desenvolvimento.

Figura 5.4 – Esforço Requerido ao longo do ciclo de vida



Fonte: Produção do autor.

Após a certificação do software, é natural uma queda na demanda de pessoal, uma vez que as atividades passam a ser limitadas às pequenas correções ou modificações menores. A fase de manutenção deste software foge ao escopo deste estudo de caso, portanto não há informação disponível sobre como foi a queda na distribuição de pessoal nas etapas posteriores.

Considerando então apenas a fase de desenvolvimento e a quantidade de pessoas mês a mês, desde o primeiro mês até o último do projeto, houve uma demanda de um total de 232.864 horas de desenvolvimento. Adotando-se o valor de 152 horas por PM (pessoa-mês), resulta então em um valor de PM = 1532 pessoas-mês (232.864 / 152).

O tamanho médio da equipe é obtido do mesmo modo como no método de estimativa, PM / TDEV. Deste modo tem-se 1532 / 54 meses, que resulta em um valor médio de aproximadamente 28 pessoas.

Durante os 54 meses, foram desenvolvidos ao todo aproximadamente 170KLOC no pacote de software destinado à Empresa X e todos os objetivos da DO-178C foram atingidos, número relativamente próximo dos 140KLOC utilizados como entrada do modelo COCOMO na estimativa de tamanho do software. A Tabela 5.9 sumariza o resulta de todos os principais parâmetros do projeto.

Tabela 5.9 – Valores reais demandados pelo projeto

Parâmetro	Esforço Real
KLOC	170
PM (Pessoa-mês)	1532
TDEV (Tempo de desenvolvimento)	54 meses
Tamanho médio da Equipe	28 pessoas

Fonte: Produção do autor.

Apresentados então o esforço estimado e o real esforço exigido pelo projeto, a seção seguinte apresenta uma discussão dos resultados obtidos com a aplicação do modelo COCOMO II no projeto estudado.

5.7 Avaliação da Estimativa

Conforme adotado por diversos outros autores (JØRGENSEN; SJØBERG, 2001; TRONTO, 2006; GRIMSTAD; JØRGENSEN, 2007; JØRGENSEN; SHEPPERD, 2007; JØRGENSEN, 2010; BASTEN; MELLIS, 2011; KHATIBI; JAWAWI, 2011), adota-se neste trabalho como medida para avaliar a precisão do modelo de estimativa em relação ao realizado a Magnitude do Erro Relativo (MRE - *Magnitude of Relative Error*), onde:

$$MRE = |esforço\ real - esforço\ estimado| \ / \ esforço\ real.$$

A Tabela 5.10 a seguir apresenta lado a lado os valores obtidos pela estimativa e os valores realmente demandados pelo projeto.

Tabela 5.10 – Comparação da estimativa x realizado

Parâmetro	Esforço Estimado	Esforço Real
KLOC	140	170
PM_{NS} (Pessoa-mês)	1052	1532
$TDEV_{NS}$ (Tempo de desenvolvimento)	32,3 meses	54 meses
Tamanho médio da Equipe	32 pessoas	28 pessoas

Fonte: Produção do autor.

Analisando os números friamente, observa-se na Tabela 5.10 que o projeto vai ao encontro das estatísticas apresentadas pelo CHAOS Report (1995) e por Moløkken (2003), pois há claramente um estouro no prazo e orçamento do projeto quando comparados com os valores estimados. Em muitos casos, tal

diferença de 32,3 meses por exemplo, para 54 meses necessários para a conclusão, pode custar a sobrevivência do projeto, ou até mesmo da empresa. Há de se considerar, no entanto, que tamanho “atraso” no projeto seja em função de uma estimativa super otimista, que pode acarretar em queda na qualidade do projeto (DE BARCELOS TRONTO; DA SILVA; SANT’ANNA, 2008).

Aplicando-se então o cálculo de MRE para análise da precisão do valor estimado de esforço (PM), resulta-se que:

MRE = | 1532 – 1052 | / 1532 = 0,3130

O cálculo do MRE mostra que a estimativa apresentou um erro relativo de cerca de 31,3% em relação ao projeto de fato realizado. Tal valor, quando comparado com resultados obtidos por outros projetos, pode ser considerado uma como uma estimativa ruim, uma vez não ser difícil encontrar na literatura valores de MRE abaixo dos 25% (JØRGENSEN; INDAHL; SJØBERG, 2003; GRIMSTAD; JØRGENSEN, 2006; KHATIBI; JAWAWI, 2011). Algumas técnicas de análise estatística que considera a performance da estimativa de diversos projetos em conjunto, como por exemplo PRED(25), não considera em sua amostra projetos com MRE maiores que 25% (BASTEN; MELLIS, 2011). Por outro lado, um MRE de 31,3% obtido com o modelo COCOMO II sem qualquer tipo de calibração exclusiva para a empresa e não considerando o fato de ser um projeto de software considerado *safety critical*, pode ser considerado razoável, uma vez que na literatura existem diversos exemplos com valores de MRE’s também na faixa dos 30%, ou até mesmo maiores que 50% (HUGHES; CUNLIFFE; YOUNG-MARTOS, 1998; GRIMSTAD; JØRGENSEN, 2007; VELARDE et al., 2016).

Por fim, este estudo de caso apresenta um caso de real de utilização do COCOMO II em um projeto de desenvolvimento de software considerado crítico e seguindo o processo da DO-178C (RTCA INC, 2011a). Ao mesmo que o resultado por ser considerado razoável, fica evidente a necessidade de uma adaptação ao método para considerar os objetivos e caraterísticas específicas de projetos que devem seguir tal norma.

Sendo assim, o próximo capítulo apresenta a definição da proposta de um modelo de estimativa de custo adaptado para considerar os aspectos do processo de desenvolvimento de software crítico definido pela DO-178C.

6 DEFINIÇÃO DO MODELO PROPOSTO

O modelo de estimativa de esforço de software apresentado neste capítulo tem o objetivo de ser aplicado no escopo de um projeto de software embarcado, considerado *safety-critical.* O software embarcado tem características específicas e deve ser desenvolvido utilizando processos de desenvolvimento específicos, seguindo um padrão estabelecido, consolidado e já aceito pelas autoridades de certificação, como é o caso dos processos definidos pela DO-178C.

No mercado aeroespacial, as autoridades de certificação de tais software se apresentam como um *stakeholder* diferente de em outros mercados. O software embarcado de um aparelho de celular, por exemplo, não sofre o mesmo rigor. Caso ele seja lento ou pouco robusto, será um produto fracassado e gerará apenas prejuízo para o fabricante. Porém, software embarcados em aeronaves ou satélites, ou mesmo equipamentos médicos, podem gerar fatalidades ou enormes prejuízos para a sociedade caso não apresente a qualidade satisfatória. Deste modo, o cumprimento com rigorosos padrões de qualidade e normas como a DO-178C é algo auditado e certificado pelas agências de homologação de cada país. Esse fator se apresenta como um dos grandes multiplicativos de custo no desenvolvimento de software crítico.

Portanto, não é objetivo deste trabalho alterar o processo de desenvolvimento de software, seja ele guiado pela DO-178C ou por qualquer outra norma de processo de desenvolvimento. Propõem-se nesta dissertação a aplicação do modelo COCOMO II, nos mesmos termos e tal como escrito, em conjunto com a norma DO-178C para a obtenção de uma estimativa de esforço que considere os aspectos relacionado ao desenvolvimento de software crítico.

A escolha pelo COCOMO II como base para o modelo proposto ocorreu a partir do estudo e análise de suas principais vantagens, podendo ser destacadas as seguintes:

- O modelo COCOMO II é o mais completo e mais bem documentado método utilizado dentre todos os métodos de estimativa de esforço (ABBAS et al., 2012).
- O modelo COCOMO II apresenta melhores resultados quando comparado com outros métodos, como por exemplo Bailey & Basili, Walston & Felix and Doty (ABBAS et al., 2012), ou SLIM (BRITTO et al., 2014).
- O modelo COCOMO II é consagrado, com diversas conquistas reconhecidas e influenciou diversos pesquisadores e desdobramento para outros modelos (BOEHM; VALERDI, 2008).
- O modelo COCOMO II continua sendo muito pesquisado e passou a ser referência e base para diversas variações de métodos e aplicações específicas (BOEHM et al., 2005; BRITTO et al., 2014).
- O modelo COCOMO II vem sendo aperfeiçoado e calibrado para aplicações específicas de modo a aumentar ainda mais seu nível de precisão (KASHYAP; MISRA, 2013; SARNO; SIDABUTAR; SARWOSRI, 2015; VELARDE et al., 2016).
- O banco de dados utilizado pelo COCOMO'81 e COCOMO II é de livre acesso e utilizado por outros pesquisadores para calibrar suas variações e validar novos métodos oriundos do COCOMO (DE BARCELOS TRONTO; DA SILVA; SANT'ANNA, 2008; LUM; MENZIES; BAKER, 2008; KAUSHIK; SONI; SONI, 2012; KASHYAP; MISRA, 2013; VELARDE et al., 2016).
- Vários pesquisadores vem utilizando de técnicas modernas para aprimorar e aumentar a precisão do COCOMO II, como por exemplo e emprego de lógica Fuzzy (IDRI; ABRAN; KJIRI, 2000; F.; ALJAHDALI, 2013; SARNO; SIDABUTAR; SARWOSRI, 2015) e redes neurais (KAUSHIK; SONI; SONI, 2012).

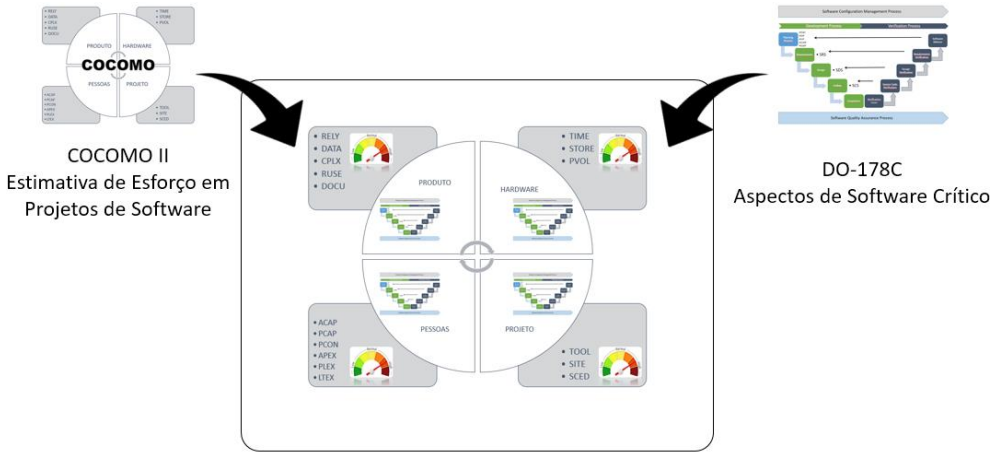
- Além de diversas indústrias e universidades, o COCOMO II também é utilizado por órgãos governamentais, como a NASA no Estados Unidos (SHETA, 2006) e governo Chinês (BOEHM; VALERDI, 2008).
- Conforme observado no capítulo 4, o COCOMO II é o modelo utilizado como referência para adaptação por diversos outros autores.

Apesar de amplo uso deste método na indústria e academia, não foram encontrados por este autor registros sobre o seu uso juntamente com a norma DO-178C. Portanto, o novo método proposto consiste na relação entre os fatores multiplicativos de esforço apresentados pelo COCOMO II e o nível de criticidade do software em questão, bem como com os meios utilizados para o cumprimento dos objetivos de tal norma.

A adoção da norma DO-178C como guia para o processo de desenvolvimento de software se dá em função agência americana FAA (*Federal Aviation Administration*) reconhecer este guia como um meio válido para demonstração de cumprimento das regulamentações para certificação de aspectos de software *safety-critical* AC20-115C (FAA, 2013).

O diagrama a seguir ilustra a junção dos conceitos do COCOMO II (Figura 2.4) com os da DO-178C (Figura 3.1).

Figura 6.1 – Diagrama do método proposto



Fonte: Produção do autor.

Considerando que a norma DO-178C endereça as mesmas, ou até mais rigorosas, preocupações de segurança no desenvolvimento do software crítico

quando comparada com outras normas, o método proposto pode ser aplicado em futuros projetos não apenas do setor aeronáutico, mas também para o aeroespacial, médico ou qualquer outro segmento onde haja necessidade de um software *safety-critical*.

Um método de estimativa de esforço específico para desenvolvimento de software críticos pode contribuir com o sucesso do projeto, diminuindo estatísticas de fracasso como as reportadas pelo CHAOS Report (THE STANDISH GROUP, 1995) ou pelos autores Moløkken e Jørgensen (2003). Além disso, o correto dimensionamento da equipe do projeto é um dos processos fundamentais destacados pelo PMBOK (PMI, 2008), contribuindo também para a viabilidade econômica do projeto.

Finalmente, a aplicação do modelo proposto pode contribuir ainda com as organizações que sofrem com a chamada “fuga de cérebros”, seja pela livre concorrência do mercado ou por outros fatores, como restrições orçamentárias ou aposentadorias em massa, uma vez o método algorítmico é menos dependente de pessoas altamente experientes.

6.1 Visão Geral

A aplicação do método proposto se dará da seguinte forma: o novo projeto a ter seu esforço estimado é submetido à um processo de desenvolvimento de sistemas. Existem diversos modelos de desenvolvimento de sistemas na literatura, adota-se neste trabalho como referência o guia para desenvolvimento de sistemas aeroespaciais apresentado pela SAE *Aerospace* ARP-4754A (SAE, 2010).

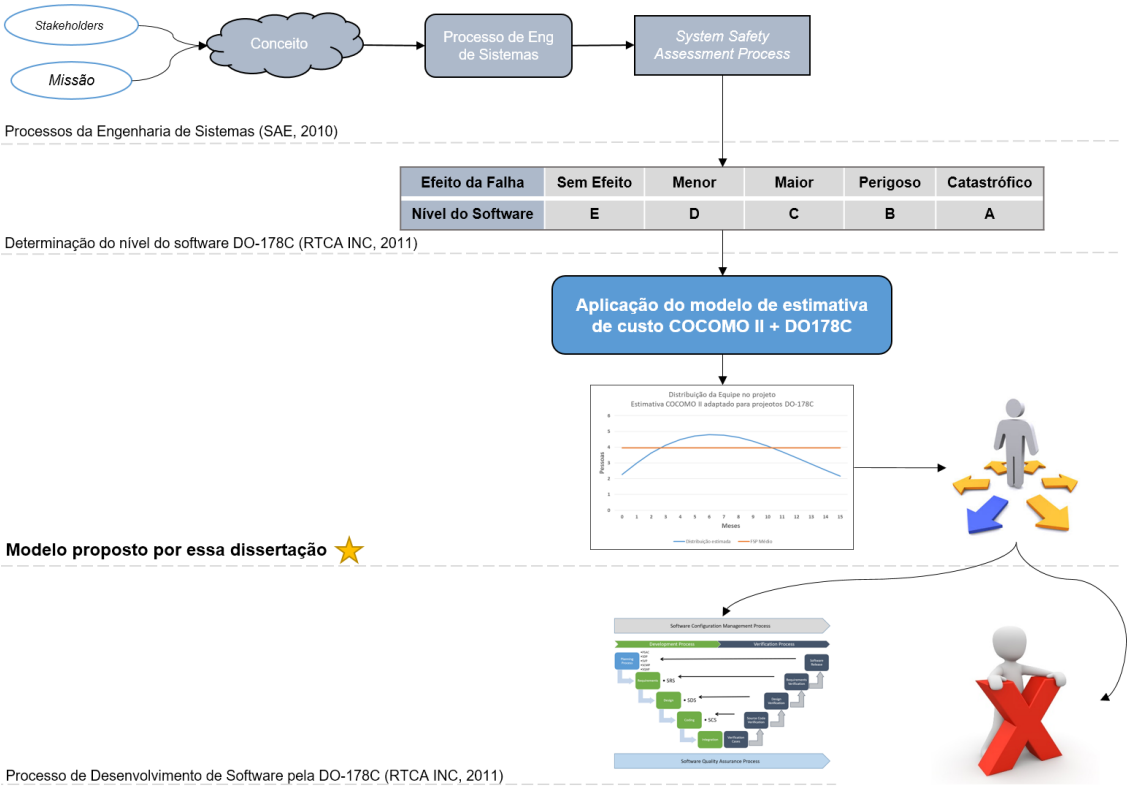
Um dos processos de desenvolvimento de sistemas estabelecidos ARP-4754A é o chamado *System Safety Assessment Process,* onde é estabelecido o efeito no sistema da falha do componente de software. Esta informação é então entrada para o processo de desenvolvimento de software estabelecido pela DO-178C, no qual a partir da condição da falha obtém-se o nível de criticidade do software em questão (A, B, C, D ou E).

Uma vez determinada a classificação do software sob os aspectos de *safety*, na fase de planejamento do projeto de desenvolvimento de software é aplicado

então o método proposto para estimativa do esforço nos mesmos moldes sugeridos pela versão original do COCOMO II, porém adaptado para ter seus fatores multiplicativos vinculados à criticidade do software.

O diagrama apresentado pela Figura 6.2 ilustra as camadas anteriores à aplicação do modelo de estimativa de esforço de desenvolvimento proposto por essa dissertação.

Figura 6.2 – Modelo Proposto no Ciclo de Desenvolvimento



Fonte: Produção do autor.

A primeira camada representa a aplicação do processo de Engenharia de Sistemas (SAE, 2010), cuja saída é então *input* para a determinação do nível do software, conforme (RTCA INC, 2011a), representada na segunda camada da Figura 6.2. Uma vez determinado o nível do software, aplica-se então o modelo de estimativa de esforço adaptado do COCOMO II, conforme detalhado no próximo capítulo e representado na terceira camada da figura.

Um exemplo prático de tal adaptação pode ser apresentado na Tabela 6.1, que consiste na alteração do fator DOCU em seu formato original (Tabela 2.9

apresenta a versão original) de modo a criar uma relação subjetiva entre ele e o nível de criticidade do software estabelecido pela DO-178C.

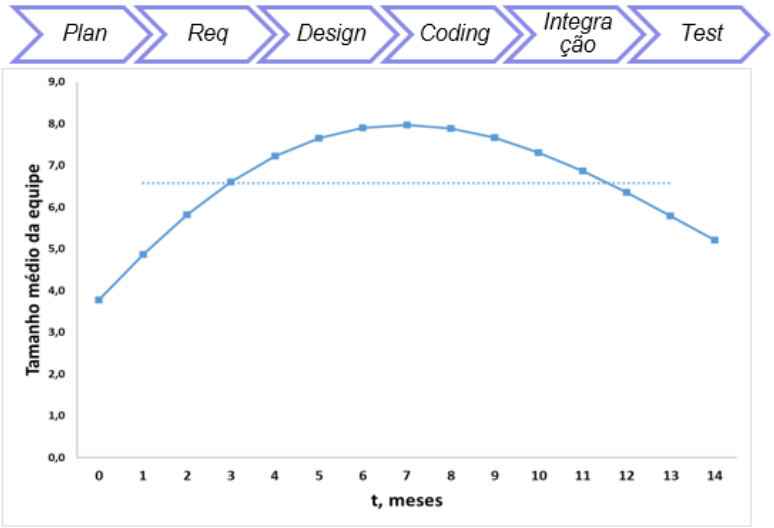
Tabela 6.1 – Fator DOCU conforme nível da DO-178C

	Fator	Classificação				
		Very Low	Low	Nominal	High	Very High
↑ Tabela original	**DOCU**	Pouca necessidade ao longo do ciclo de vida	Alguma necessidade	Necessidade de tamanho ideal para o ciclo de vida	Excessiva necessidade ao longo do ciclo de vida	Muito excessiva necessidade ao longo do ciclo de vida
		0.81	0.91	1.00	1.11	1.23
↓ Proposta adaptação	**DO-178C Level**	n/a	n/a	n/a	D	A,B,C

Fonte: Adaptado de Boehm et al. (2000).

Esta mesma relação é realizada para todos os dezessete fatores multiplicativos de esforço estabelecidos pelo COCOMO II. Ao final da aplicação do novo modelo se dará então uma estimativa de esforço e tempo de desenvolvimento calculados da mesma maneira prevista originalmente, porém considerando a DO-178C e endereçando as preocupações e práticas de desenvolvimento de software críticos. Um exemplo de estimativa de esforço e tempo de desenvolvimento é apresentado na Figura 6.3 seguir. A curva mostra o crescimento estimado da equipe ao longo do desenvolvimento do projeto, passando por todas as fases desde Planejamento até Integração e testes.

Figura 6.3 – Exemplo de estimativa de esforço e prazo



Fonte: Produção do autor.

Após a aplicação do modelo proposto, é obtida então a estimativa de esforço para o projeto em questão em pessoa-mês, tamanho médio da equipe, prazo em meses para desenvolvimento completo, bem como a distribuição da quantidade de pessoas ao longo do ciclo de desenvolvimento. Tal estimativa deve então ser analisada pelos gestores do projeto para a tomada de decisão sobre a continuação ou não do projeto, bem como detalhes de contratação, terceirização etc. Caso o projeto seja aprovado, o processo de desenvolvimento de software segue seu ciclo de vida natural, porém agora com uma melhor visibilidade para os gestores do projeto sobre os custos e prazos associados.

As próximas seções detalham os passos no processo de desenvolvimento para a determinação do nível do software conforme a DO-178C.

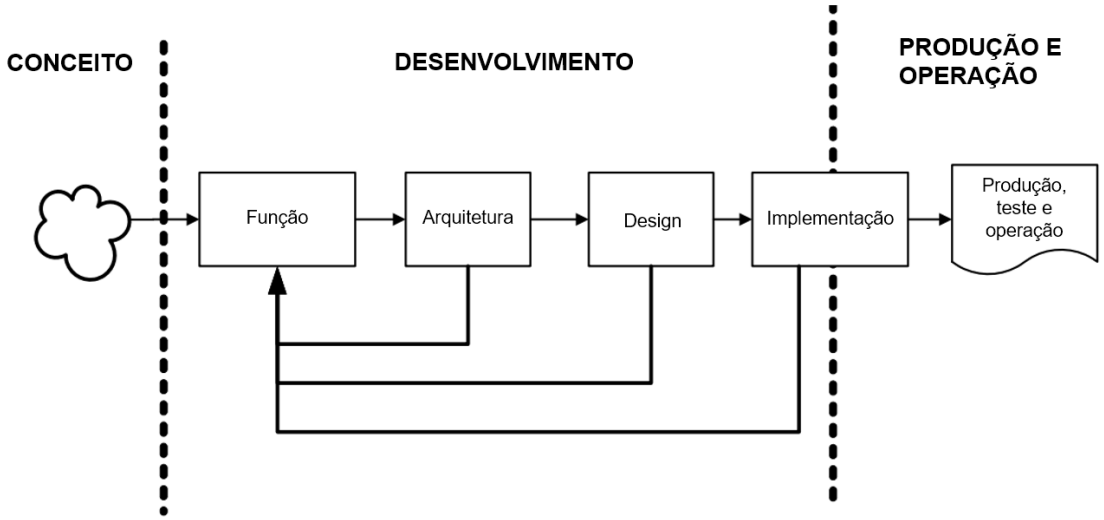
6.2 Ciclo de Desenvolvimento de Sistemas

Para o correto entendimento da determinação do nível do software, é preciso situar o leitor no completo ciclo de desenvolvimento de um sistema, desde as fases conceituais até a alocação de requisitos à software.

O ciclo de desenvolvimento é a realização sequencial de todas as fases e atividades do ciclo de vida. Um software embarcado é apenas um item parte de um sistema maior. O desenvolvimento de um sistema completo abrange diversas disciplinas e técnicas de gerenciamento dedicadas.

A literatura apresenta diversos modelos de ciclo de vida, processos de desenvolvimento, etapas e atividades. A Figura 6.4 apresenta o conceito do ciclo de vida conforme o guia de práticas aeroespaciais ARP-4754A (SAE, 2010).

Figura 6.4 – Ciclo de desenvolvimento



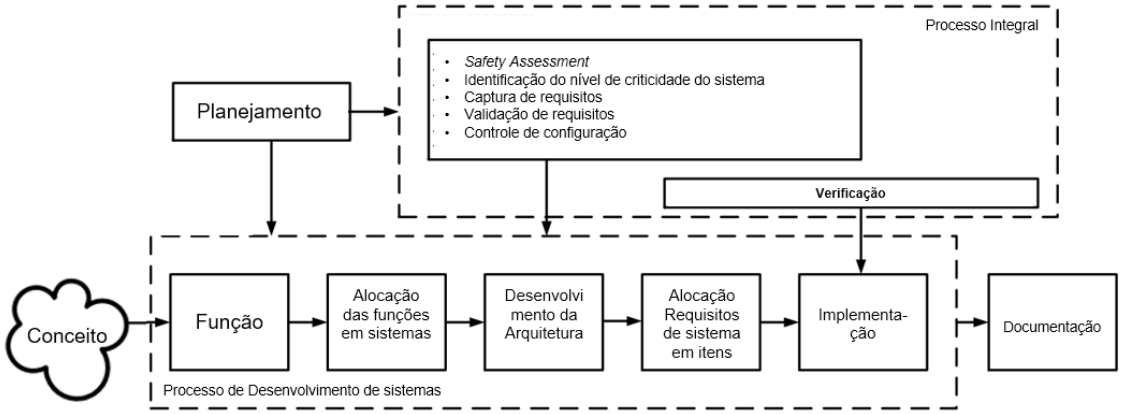
Fonte: Adaptado de SAE INTERNATIONAL (2010).

A fase de Conceito determina as necessidades iniciais do sistema, performance, configuração etc. No caso de aeronaves, nesta etapa determinam-se por exemplo o tamanho da aeronave, quantidade de passageiros, quantidade de motores, performance etc. A partir desses requisitos básicos, as funções do sistema começam a ser identificadas e requisitos específicos de cada função podem ser capturados.

Com a identificação de diversas funções, surgem também os requisitos de interface entre tais funções e interface com o ambiente externo, na qual o sistema é operado. Uma vez as funções identificadas, o processo sugerido pela (SAE, 2010) estabelece que tais funções devem ser agrupadas e os seus respectivos requisitos alocados à estas funções. É então que a fase de Arquitetura estabelece a estrutura e as fronteiras de cada função, bem como a alocação à software ou hardware, de modo que atendem aos requisitos de segurança e performance.

As fases de Design e Implementação seguem então seus respectivos processos, conforme a alocação de cada função. A Figura 6.5 detalha a fase de Desenvolvimento conforme o guia de práticas aeroespaciais.

Figura 6.5 – Modelo de desenvolvimento de Sistemas

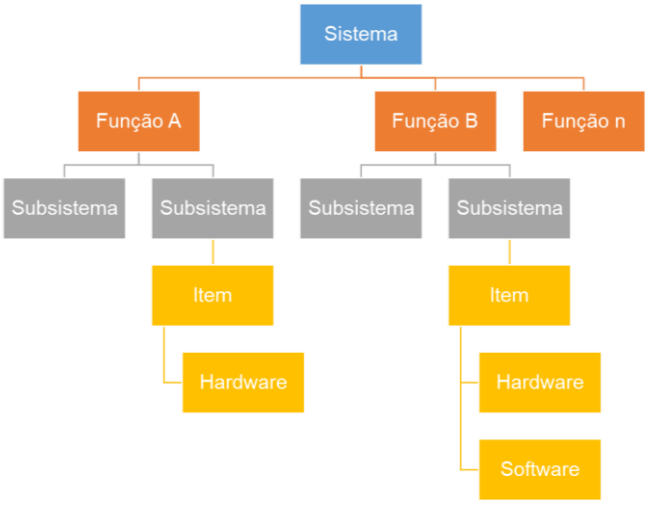


Fonte: Adaptado de SAE INTERNATIONAL (2010).

Durante este processo, a etapa de “Alocação Requisitos de sistema em itens”, identifica então os requisitos de sistema que serão alocados a software. Deste modo, funções atribuídas à software seguem então o processo estabelecido para desenvolvimento de software. Cabe aqui reforçar que existem inúmeros processos de desenvolvimento de software estabelecidos, adota-se, porém, neste trabalho os processos definidos pela DO-178C, conforme descrito na seção 3.1.

De maneira similar, funções atribuídas a hardware seguem normas e processos dedicados para hardware. A Figura 6.6 exemplifica uma estrutura de produto considerando um sistema, subsistemas e itens.

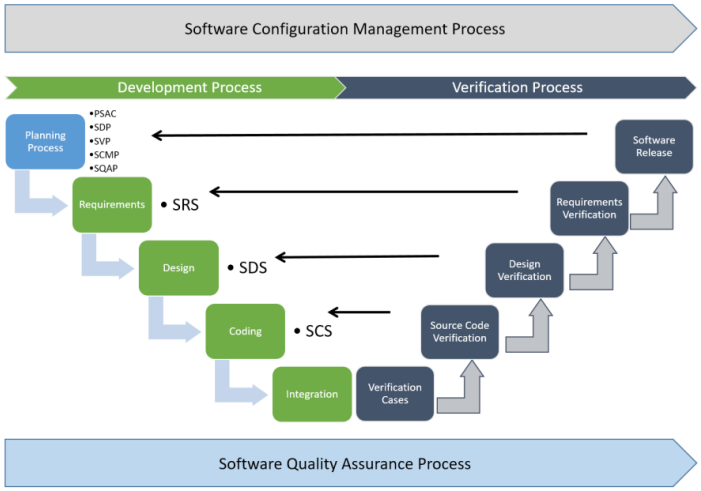
Figura 6.6 – Exemplo de estrutura de sistema



Fonte: Produção do autor.

As funções de sistemas então alocadas à software devem fluir para o processo de desenvolvimento de software, com o seu próprio ciclo de vida. A Figura 6.7 apresenta o ciclo de vista proposto pela DO-178C.

Figura 6.7 – Ciclo de vida do processo da DO-178C



Fonte: Adaptado de RTCA (2011a).

Conforme mencionado na seção 3.1, os objetivos da DO-178C a serem cumpridos por cada projeto varia conforme a determinação do nível do software. A próxima seção apresenta então esta etapa do processo.

6.3 Determinação do Nível do Software

Segundo a DO-178C, o que determina o nível do software quanto à sua criticidade é o processo de SSA - *System Safety Assessment* (avaliação da segurança do sistema) realizado nas fases anteriores ao processo de desenvolvimento de software. O processo de SSA determina o nível apropriado aos componentes de software de um determinado sistema com base na condição de falha que pode resultar do comportamento anômalo do software. Em outras palavras, é analisado o impacto da falha do software no sistema como um todo. Em alguns casos mínimas falhas do software podem ter impactos catastróficos no sistema. O simples *overflow* de uma variável de 32 bits por exemplo, destruiu um bilhão de dólares no ar no primeiro lançamento do veículo lançador de satélites francês, o Ariane 5 (TERRY BAHILL; HENDERSON, 2005). Por outro lado, grandes falhas de software podem ter impacto nulo no sistema em termos de segurança.

O impacto da perda de função e do mau funcionamento é analisado quanto ao seu efeito no sistema. Caso o mau funcionamento do software contribuía para uma falha catastrófica, então o nível do software é considerado A, ou seja, terá o processo de desenvolvimento mais rigoroso possível. Se o comportamento anômalo de um componente de software contribuir para mais de uma condição de falha, o componente de software deve receber o nível de software associado à condição de falha mais grave com a qual o software pode contribuir.

A agência de aviação americana – FAA (*Federal Aviation Administration*) – define a probabilidade mínima para cada condição de falha, conforme Tabela 6.2 a seguir.

Tabela 6.2 – Probabilidade exigida para cada categoria de falha

Categoria da Condição de falha	Probabilidade qualitativa	Probabilidade quantitativa
Catastrófico	Extremamente improvável	$<10^{-9}$
Perigoso	Extremamente remota	$<10^{-7}$
Maior	Remota	$<10^{-5}$
Menor	Provável	$<10^{-3}$
Sem efeito de segurança	Sem requisito de probabilidade	N/A

Fonte: Adaptado de FAA (2011).

Além do requisito de probabilidade, o FAA introduz ainda o conceito de *Single Failure Concept* (falha simples), que fiz que nenhuma falha simples deve gerar um evento catastrófico. Em termos práticos, isso significa que mesmo que um sistema tenha uma probabilidade de falha $< 10^{-9}$, sua simples falha não deve levar a um evento catastrófico. Deste modo, redundância de função deve ser aplicada no projeto do sistema, isto é, a mesma função sendo executada por dois subsistemas diferentes.

A Tabela 6.3 relaciona cada categoria de falha conforme o guia (SAE, 2010) com o respectivo nível de software conforme a DO-178C, bem como com a quantidade de objetivos a serem cumpridos durante o processo de desenvolvimento de tal software.

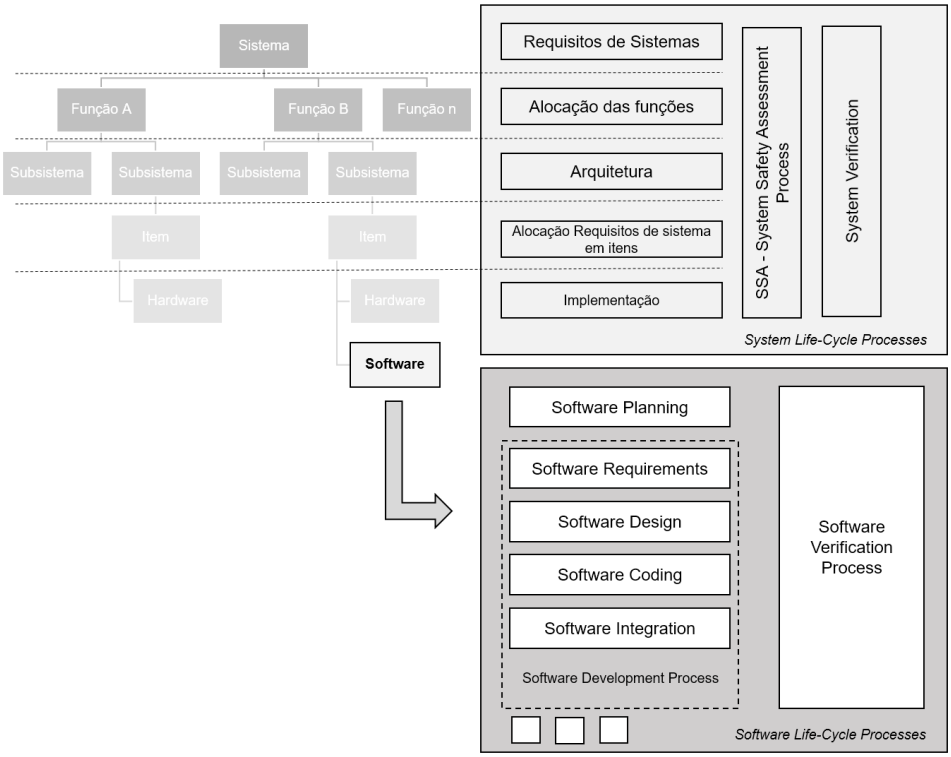
Tabela 6.3 – DO-178C: Condição de falha, nível e quantidade de objetivos

Categoria da Condição de falha	Nível do Software pela DO-178C	Quantidade de objetivos
Catastrófico	A	71
Perigoso	B	69
Maior	C	62
Menor	D	24
Sem efeito de segurança	E	0

Fonte: Adaptado de RTCA (2011a).

Sendo assim, fica claro uma relação entre os processos de sistemas e de software. A Figura 6.8 ilustra a interação entre os dois processos.

Figura 6.8 – Interação entre os processos de sistemas e software

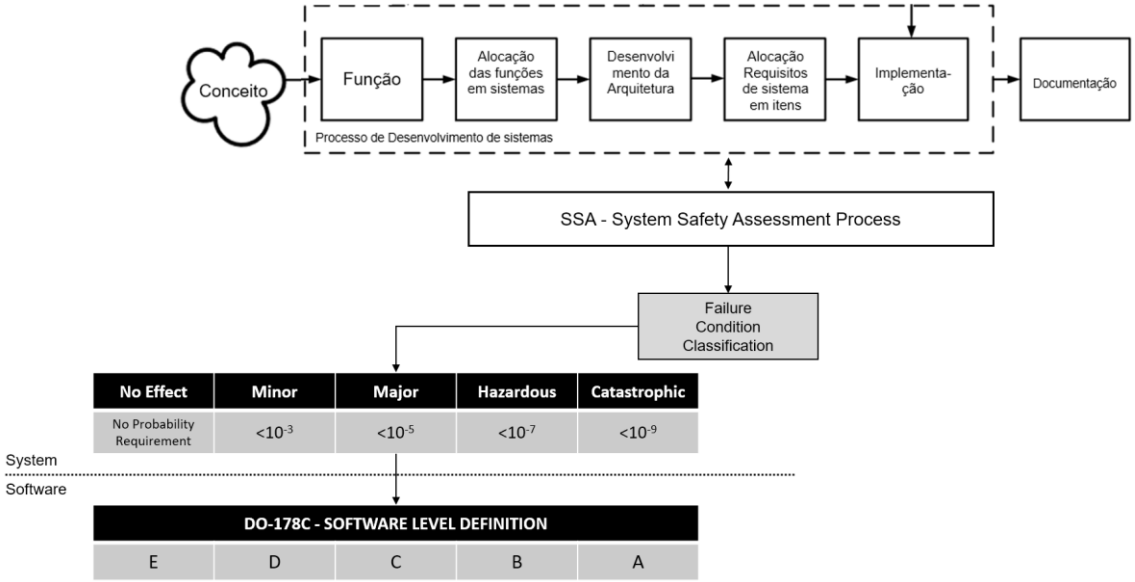


Fonte: Adaptado de RTCA (2011a).

A determinação do nível de software é então em função do efeito da anomalia do software no sistema e da sua probabilidade de ocorrência. Tal análise de segurança é feito pelo processo de sistemas e fornecido como entrada para o processo de software.

Uma vez então executado o processo de sistema e identificado a classificação do efeito da falha de um determinado sistema, as funções alocadas à software devem ter então o processo com o rigor relativo ao nível do software em questão, cumprindo com a quantidade de objetivos conforme cada nível. A Figura 6.9 ilustra as etapas para determinação do nível do software, na qual observa-se que as atividades são de responsabilidade da Engenharia de Sistemas. O resultado da atividade SSA (*System Safety Assessment)* fornece finalmente o nível exigido do software, conforme o efeito do seu mal funcionamento.

Figura 6.9 – Etapas para determinação do nível do software



Fonte: Adaptado de FAA (2011).

A classificação do nível do software já deve ser considerada na fase de planejamento, onde os objetivos a serem cumpridos devem ser conhecidos, bem como o plano proposto para se cumprir tais objetivos.

Conforme a Tabela 3.2, o nível do software tem impacto direto no esforço necessário para seu desenvolvimento, uma vez que a quantidade de objetivos da norma DO-178C varia em função do nível do software.

O próximo capítulo apresenta os passos para aplicação do modelo proposto por essa dissertação.

7 PASSOS PARA APLICAÇÃO DO MODELO PROPOSTO

O modelo proposto por esta dissertação deve ser aplicado seguindo o mesmo método que o COCOMO II original, aplicando as mesmas fórmulas e sequências estabelecidas. O que o modelo proposto neste trabalho difere do original é a relação feita entre o nível de software conforme a DO-178C e a classificação de cada um dos fatores multiplicativos de custo do COCOMO II.

Uma vez identificado então o nível do software, na fase de planejamento do projeto se espera obter uma estimativa do esforço necessário para desenvolvimento do software de modo que os gerentes do projeto possam garantir o prazo e recursos necessários para que se cumpra todos os objetivos da norma DO-178C e se conclua o projeto com a qualidade desejada.

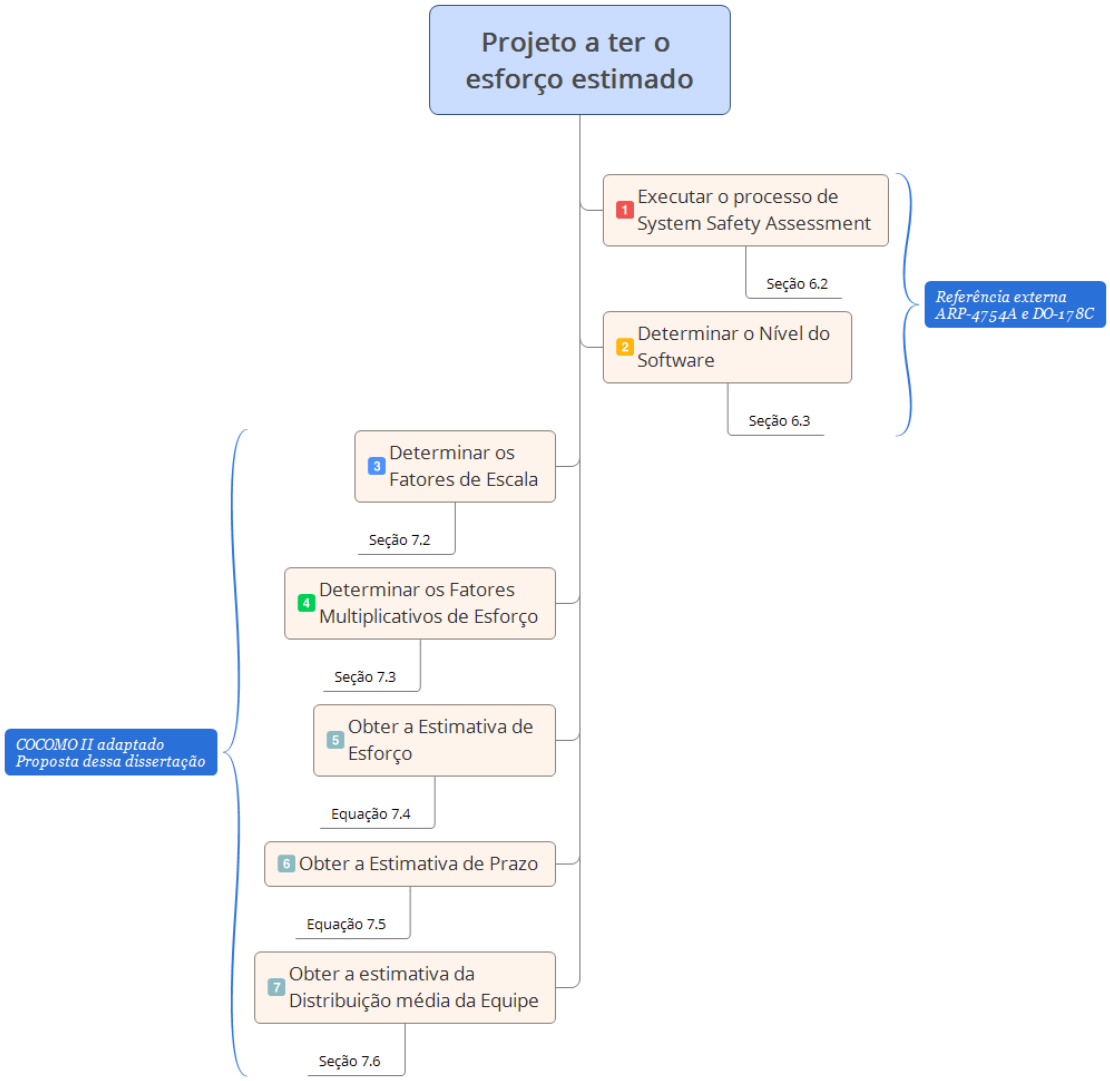
Como base, adota-se então o modelo COCOMO II *Post-Architecture Model* que deve ser aplicado nos mesmos termos conforme apresentado na seção 2.3.2, utilizando-se das mesmas fórmulas e sequência de atividades, iniciando com a identificação dos fatores de escala, em seguida dos fatores multiplicativos de custo, porém estes, adaptados conforme proposto a seguir.

A Figura 7.1 apresenta as principais etapas para aplicação do modelo proposto por essa dissertação, referenciado as seções deste documento das quais os detalhes de cada etapa estão descritos.

As duas primeiras etapas são processos externos ao modelo de estimativa de custo proposto por essa dissertação e, portanto, adota-se como referência o processo de Engenharia de Sistemas proposto por (SAE, 2010) e o processo de Desenvolvimento de Software proposto por (RTCA INC, 2011a).

As demais etapas (3, 4, 5, 6 e 7) da aplicação do modelo são baseadas no COCOMO II, porém com as adaptações propostas por essa dissertação.

Figura 7.1 – Passos para aplicação do modelo proposto



Fonte: Produção do autor.

A etapa 3 é detalhada na seção 7.1. A etapa 4, detalhada na seção 7.2. A etapa 5, é então detalhada na seção 7.3. Finalmente, as etapas 6 e 7 são detalhadas nas seções 7.4 e 7.5 respectivamente.

7.1 Fatores de Escala

Conforme a seção 2.3.2.1, os fatores de escala basicamente determinam as economias e custos do projeto em desenvolvimento. Pelo fato de tais fatores abordarem questões institucionais e relacionadas com gerenciamento de projetos, propõem-se a identificação da escala de cada fator, bem como do cálculo do parâmetro E, exatamente conforme apresentado na seção 2.3.2.1.

Na análise realizada por este trabalho, os fatores de escala não envolvem questões técnicas e podem ser utilizados para qualquer tipo de software, seja crítico ou não, embarcado ou não, de tempo real ou sem compromisso de tempo. Sendo assim, as particularidades inerentes ao desenvolvimento de software *safety-critical* guiado pela DO-178C serão abordadas nos fatores multiplicativos de custo, detalhados nas próximas seções.

7.2 Fatores Multiplicativos de Esforço (EM)

Apresentam-se nesta seção os fatores multiplicativos de esforço apresentados pelo COCOMO II (BOEHM et al., 2000) a adaptados para o novo modelo proposto por este trabalho.

7.2.1 RELY – Required Software Reliability

Este fator mede o efeito de uma falha de software na função em que ele deve executar durante um período de tempo. Para cada efeito da falha um nível é atribuído. Dentre os fatores multiplicativos de custo apresentados pelo COCOMO II, este é o que mais facilmente pode ser relacionado com a DO-178C.

A Tabela 7.1 apresenta a classificação para este fator conforme o nível do software.

Tabela 7.1 – RELY: Escala de Classificação adaptada

Fator	**Classificação**					
	Very Low	Low	Nominal	High	Very High	Extra High
RELY	Inconveniente	Baixo, com perdas facilmente recuperadas	Moderado, com perdas facilmente recuperadas	Alto risco financeiro	Alto risco para vidas humanas	n/a
	0.82	0.92	1.00	1.10	1.26	n/a
DO-178C	n/a	n/a	D	C	A, B	

Fonte: Adaptado de Boehm et al. (2000).

A justificativa para tal relação se dá pelo seguinte motivo: software níveis A ou B, segundo a DO-178C, contribuem respectivamente para falhas catastróficas ou perigosas. Conforme definição apresentada na seção 3.1, em ambos casos tais falhas podem levar a fatalidades ou sérios ferimentos. Deste modo, software nível A ou B devem ser selecionados como *Very High*, ou seja,

apresentam risco para a vida humana e, por tanto, o maior fator multiplicativo de custo possível.

No caso do software nível C, o mau funcionamento leva às falhas maiores (*major),* que segundo a DO-178C podem reduzir a capacidade do avião ou da tripulação em atuar em situações de operação adversa, com aumento significativo da carga de trabalho da tripulação, podendo gerar algum desconforto ou lesões para os passageiros. Portanto, não há um risco para a vida humana, porém pode gerar grandes prejuízos em função de insatisfação dos clientes (passageiros), custo manutenção, investigação do problema etc. Deste modo, tal software pode ser relacionado com o nível *High*, no qual segundo o COCOMO II apresenta apenas riscos financeiros.

No caso do software nível D o seu mau funcionamento leva às falhas menores, com impacto não significativo no produto ou na tripulação. Sendo assim, pode ser relacionado com a classificação Nominal, no qual ele não apresenta um fator multiplicativo de custo no projeto, pois seu valor é 1.

Finalmente, caso o software em questão não tenha impacto em segurança, ou seja, nível E, a seleção entre *Low* ou *Very Low* deve ser feita conforme orientação original do modelo COCOMO II.

7.2.2 DATA – Data base Size

Este fator relaciona o efeito de testes de grandes quantidades de dados no desenvolvimento do produto. O nível é atribuído com base na taxa de D/P, que significa a taxa de Bytes no banco de dados de teste (D) pela quantidade de linhas de código do software (P). Em outras palavras, DATA está capturando o esforço necessário para montar e manter os dados necessários para concluir o teste do software até o início da operação. A Tabela A.2 apresenta a relação entre D/P sugerida pelo COCOMO II para a seleção da classificação do fator multiplicativo de custo DATA.

Do ponto de vista da DO-178C, o tamanho do banco de dados de testes em bytes é algo irrelevante, tampouco sua relação com a quantidade de linhas de código. A abordagem de testes apresentada pela DO-178C é denominada *RBT – Requirements-Based Test* – testes baseados em requisitos, ou seja, não

há relação com o tamanho do software em linhas de código. Devido a abordagem RBT, mesmo um cenário com um banco de dados de testes extremamente grande e uma eventual relação D/P ≥ 1000, pode não ser suficiente para cumprir com todos os objetivos da DO-178C caso tais testes não testem todos os requisitos.

Para a DO-178C, a complexidade e quantidade de testes vai variar então conforme os requisitos do software. Quanto mais complexo o software, maior a quantidade de requisitos, bem como maior a quantidade de testes (YELISETTY; MARQUES; TASINAFFO, 2015). Além disso, quanto maior o nível do software maior é a quantidade de objetivos relacionados com o processo de verificação a serem demonstrados. A Tabela 7.2 apresenta a relação entre a quantidade de objetivos total da DO-178C com os objetivos relacionados ao processo de verificação, para cada nível de software conforme seção 3.1.

Tabela 7.2 – DO-178C: Quantidade de objetivos de verificação

Nível	Total de Objetivos	Objetivos de Verificação
A	71	43 (61%)
B	69	41 (59%)
C	62	34 (55%)
D	26	9 (35%)

Fonte: Adaptado de RTCA (2011a).

As tabelas apresentadas no ANEXO C, ANEXO D, ANEXO E, ANEXO F e ANEXO G são todas referentes aos objetivos relacionados com o processo de verificação. Nota-se que o grande peso e rigor dos processos estabelecidos pela DO-178C está no processo de verificação, seja por meio de revisões, analises ou testes.

Dado então o rigor da DO-178C no processo de verificação e a dificuldade em se cumprir com todos os objetivos relacionados em função da complexidade do software *safety-critical*, ferramentas modernas com o foco em verificação podem auxiliar na execução destas atividades, gerando e executando casos e

procedimentos de testes de maneira automática, bem como analisando o resultado final com base em critérios de “passou” ou “falhou”.

Deste modo, sugere-se então na Tabela 7.3 a seleção do fator DATA conforme o nível do software segundo a DO-178C e em função do fator TOOL, que relaciona o uso ou não de ferramentas qualificadas no processo de desenvolvimento. O conceito de qualificação de ferramenta, bem como seus critérios, será discutido no fator TOOL, seção 7.2.15.

Tabela 7.3 – DATA: Escala de Classificação adaptada

Fator	**Classificação**					
	Very Low	Low	Nominal	High	Very High	Extra High
DATA	n/a	0.90	1.00	1.14	1.28	n/a
DO-178C	n/a	A,B,C,D para TOOL “Nominal”, “High” ou “Very High”	D	C	A, B	n/a

Fonte: Adaptado de Boehm et al. (2000).

A justificativa para tal relação se dá pelo fato de que para projetos de desenvolvimento de software com o uso de ferramentas qualificadas, as atividades do processo de verificação podem ser simplificadas ou mesmo substituídas por tais ferramentas. Deste modo, recomenda-se então a seleção da classificação “low” caso existam tais ferramentas, independentemente do nível do software.

No cenário em que não se utiliza ferramentas qualificadas, ou seja, para o TOOL selecionado como “low” ou “very low”, a classificação do parâmetro DATA varia então conforme o nível do software, de modo a refletir o esforço necessário em função da maior quantidade de objetivos a serem demonstrados, conforme a Tabela 7.2.

7.2.3 CPLX – Product Complexity

A complexidade do produto é dividida em cinco áreas conforme o modelo COCOMO II: controle de operação, operações computacionais, operações com dispositivos, gerenciamento de dados, operações de interface com usuário. A combinação dessas áreas deve ser utilizada para chegar ao nível do CPLX. A Tabela A.3 apresenta a classificação deste fator conforme sugerido pelo modelo COCOMO II.

O modelo proposto por este trabalho sugere então a consideração do nível do software conforme a DO-178C para a definição da complexidade. A Tabela 7.4 a apresenta o cruzamento da classificação original sugerida pelo COCOMO II com o nível do software conforme DO-178C, resultando então na nova classificação a ser adotada na estimativa.

Tabela 7.4 – CPLX: Escala de Classificação adaptada

DO-178C	CPLX Very Low	CPLX Low	CPLX Nominal	CPLX High	CPLX Very High	CPLX Extra High
A	Nominal	Nominal	High	Very High	Extra High	Extra High
B	Nominal	Nominal	High	Very High	Extra High	Extra High
C	Nominal	Nominal	Nominal	High	Very High	Extra High
D	Very Low	Low	Nominal	High	Very High	Extra High

Fonte: Adaptado de Boehm et al. (2000).

Para efeito didático, tem-se a seguir os passos sugeridos para identificação da classificação deste fator:

- Obtém-se a classificação original do parâmetro CPLX conforme sugerido pelo modelo COCOMO II na Tabela A.3;
- Relaciona-se então a classificação original com o nível do software segundo a DO-178C, conforme apresentado na Tabela 7.4;
- O resultado do cruzamento é então a classificação final que deve ser adotada para o fator CPLX.

Observa-se pela Tabela 7.4 que a classificação original sugerida pelo COCOMO II é mantida apenas para o caso de software nível D, uma vez que um menor número de objetivos deve ser atendido. Porém, para o caso de software níveis A, B ou C recomenda-se então que a menor classificação possível seja a nominal. A justificativa para tal seleção se dá pelo fato de que para tais software, há uma complexidade associada pelo simples fato da utilização da norma DO-178C, bem como um aumento do custo associado. Para refletir este custo na estimativa, sugere-se então um valor sempre ≥ 1.

7.2.4 RUSE – Developed for Reusability

Este fator relaciona o esforço adicional que será necessário para desenvolver software com intenção de ser reutilizado por projetos futuros. Esse esforço é consumido com a criação de um design mais genérico de software, documentação mais elaborada e testes mais extensos para garantir que os componentes estejam prontos para uso em outros aplicativos. A Tabela A.4 apresenta a relação entre nível de reuso e classificação deste parâmetro conforme sugerido pelo COCOMO II. Do ponto de vista da DO-178C, não existem objetivos adicionais a serem cumpridos caso o software seja desenvolvimento com a intenção de ser reutilizado. No entanto, há uma complexidade adicional em termos de documentação imposta pelo FAA por meio da AC 20-148 (FAA, 2004b), que define o guia para desenvolvimento de componentes de software reutilizáveis, denominado em inglês de RSC – *Reusable Software Component*.

Deste modo, sugere-se então na Tabela 7.5 a seleção do fator RUSE conforme o nível do software segundo a DO-178C e em função de se tratar ou não de um RSC.

Tabela 7.5 – RUSE: Escala de Classificação adaptada

Fator	**Classificação**					
	Very Low	Low	Nominal	High	Very High	Extra High
RUSE	n/a	Nenhum, reuso	Reuso dentro do projeto	Reuso em vários projetos	Reuso em vários produtos	Reuso em várias áreas organizacionais
	n/a	0.95	1.00	1.07	1.15	1.24
DO-178C	n/a	A,B,C,D sem RSC	D	C	B	A

Fonte: Adaptado de Boehm et al. (2000).

A justificativa para tal relação se dá pela simples relação do nível de software da DO-178C com a complexidade do reuso. Do ponto de vista das autoridades de certificação, a conformidade do software é sempre com relação ao sistema em que ele é executado, sendo assim o reuso por diferentes áreas, produtos ou projetos não aumenta, tampouco alivia, o esforço necessário no desenvolvimento do software. O desenvolvimento para reuso em si,

independentemente de área, produto ou projeto, acarretará um esforço maior para software nível A ou menor para nível D.

Finalmente, para refletir a complexidade de se desenvolver um RSC seguindo a DO-178C, sugere-se então a relação com a classificação original do COCOMO II conforme apresentado na Tabela 7.5.

Conforme o próprio modelo COCOMO II estabelece, o desenvolvimento para reuso impõe restrições às classificações RELY e DOCU do projeto. A classificação RELY deve ser no máximo um nível abaixo da classificação RUSE. A classificação DOCU deve ser, pelo menos, *Nominal* para as classificações *Nominal* e *High* no fator RUSE e, pelo menos *High* para as classificações RUSE como *Very High* ou *Extra High*.

7.2.5 DOCU – Documentation Needs

Este fator é avaliado em termos da necessidade de documentação ao longo do ciclo de vida do projeto em desenvolvimento. De todos os fatores apresentados pelo COCOMO II, este é o de mais simples relação com a DO-178C, uma vez que ela é muita clara com relação aos documentos necessários para cada nível de software.

A Tabela 7.6 apresenta então a sugestão de classificação do fator DOCU conforme o nível de software da DO-178C.

Tabela 7.6 – DOCU: Escala de Classificação adaptada

Fator	**Classificação**					
	Very Low	Low	Nominal	High	Very High	Extra High
DOCU	Pouca necessidade ao longo do ciclo de vida	Alguma necessidade	Necessidade de tamanho ideal para o ciclo de vida	Excessiva necessidade ao longo do ciclo de vida	Muito excessiva necessidade ao longo do ciclo de vida	n/a
	0.81	0.91	1.00	1.11	1.23	n/a
DO-178C	n/a	n/a	n/a	D	A, B, C	

Fonte: Adaptado de Boehm et al. (2000).

A justificativa para tal relação se dá pela documentação exigida pela DO-178C nas fases de planejamento e desenvolvimento:

- PSAC - Plan for Software Aspects of Certification
- SDP - Software Development Plan
- SVP - Software Verification Plan
- SCMP - Software Configuration Management Plan
- SQAP - Software Quality Assurance Plan
- SRS - Software Requirements Standards
- SDS - Software Design Standards
- SCS - Software Code Standards
- SVR - Software Verification Results
- SRD - Software Requirements Data
- SDD - Software Design Description
- Trace Data

As tabelas DO-178C apresentadas no ANEXO A e ANEXO B deixam claro que todos esses artefatos são necessários para os níveis A, B e C, deste modo sugere-se a classificação mais elevada do COCOMO II para este parâmetro, *Very High.*

No caso de software nível D, alguns desses documentos não são necessários ou podem ser simplificados, uma vez que não há necessidade de se cumprir com alguns dos objetivos endereçados por eles.

7.2.6 TIME – Execution Time Constraint

Esta é a medida relacionada com a restrição para o tempo de execução do software. A medida é dada em função do percentual do tempo disponível do processador que se espera que seja consumido pelo software em questão. A Tabela A.6 apresenta a classificação deste fator conforme recomenda o modelo COCOMO II.

Para este fator, sugere-se então a consideração do nível do software conforme a DO-178C em função da complexidade de se obter e evidenciar o chamado

WCET – *Worst Case Execution Time*. Conforme definição da DO-248C, WCET é o tempo mais longo possível para concluir a execução de um conjunto de tarefas em um determinado processador em um ambiente computacional em tempo real. A tabela da DO-178C no ANEXO E, objetivo 6, menciona a atividade relacionada com a análise de WCET e é aplicável para os níveis A, B e C.

A Tabela 7.7 apresenta o cruzamento da classificação original sugerida pelo COCOMO II com o nível do software conforme DO-178C, resultando então na nova classificação a ser adotada na estimativa.

Tabela 7.7 – TIME: Escala de Classificação adaptada

DO-178C	TIME Very Low	TIME Low	TIME Nominal	TIME High	TIME Very High	TIME Extra High
A	n/a	n/a	High	Very High	Extra High	Extra High
B	n/a	n/a	High	Very High	Extra High	Extra High
C	n/a	n/a	High	Very High	Extra High	Extra High
D	n/a	n/a	Nominal	Nominal	High	Very High

Fonte: Adaptado de Boehm et al. (2000).

Para efeito didático, tem-se a seguir os passos sugeridos para identificação da classificação deste fator:

- Obtém-se a classificação original do parâmetro TIME conforme sugerido pelo modelo COCOMO II na Tabela A.6;
- Relaciona-se então a classificação original com o nível do software segundo a DO-178C, conforme apresentado na Tabela 7.7;
- O resultado do cruzamento é então a classificação final que deve ser adotada para o fator TIME.

Observa-se pela Tabela 7.7 que para o caso de software níveis A, B ou C recomenda-se então que a menor classificação possível seja a *High*, de modo a refletir a complexidade e necessidade da análise de WCET. Mesmo no cenário mais favorável, uma aplicação simples em um processador extremamente rápido, há de se considerar e esforço adicional para evidenciar tamanha margem de tempo de processamento com a execução da atividade de

WCET. Deste modo, para estes níveis de software sugere-se desconsiderar a classificação *Nominal*.

Para o caso do software nível D, mesmo no cenário mais desfavorável, no qual 95% do tempo do processador é consumido, sugere-se a maior classificação como *Very High*, uma vez que análise de WCET não precisa ser demonstrada e um possível esforço adicional para otimização e melhorias do software está mais relacionada com o domínio técnico e boas práticas de programação, com menos impacto no processo e documentação.

Deste modo, a estimativa de esforço reflete tal flexibilidade do software nível D no que diz respeito ao processo.

7.2.7 STOR – Main Storage Constraint

Esta é a medida representa o grau de restrição do principal meio de armazenamento de dados do sistema. A Tabela A.7 apresenta a classificação deste fator conforme recomenda o modelo COCOMO II. Dado o notável aumento na capacidade de processamento dos processadores modernos, bem como na disponibilidade de armazenamento, pode-se questionar se essas variáveis de restrição ainda são relevantes. No entanto, muitos aplicativos continuam a se expandir para consumir quaisquer recursos disponíveis, tornando esses direcionadores de custos ainda relevantes. No tocante ao software embarcado *safety-critical*, esse parâmetro é ainda mais relevante, uma vez que software embarcado em geral contam com grande restrição de recurso. No estudo de caso descrito no Capítulo 5 por exemplo, o software em questão é executado em um hardware com apenas 3MB de memória RAM e 4MB de ROM, valor este infinitamente menor que os smartphones atuais, que contam com cerca de 32GB em sua grande maioria.

Para este fator, sugere-se então a consideração do nível do software conforme a DO-178C em função da complexidade de se obter e evidenciar a chamada *Memory Analysis*. A análise de consumo de memória é mais simples que a WCET, podendo ser muitas vezes realizada de maneira estática, por meio da análise do *mapfile*. A tabela da DO-178C no ANEXO E, objetivos 6 e 7,

mencionam as atividades relacionada com a análise de memória e são aplicáveis para os níveis A, B e C.

A Tabela 7.8 apresenta o cruzamento da classificação original sugerida pelo COCOMO II com o nível do software conforme DO-178C, resultando então na nova classificação a ser adotada na estimativa.

Tabela 7.8 – STOR: Escala de Classificação adaptada

DO-178C	STOR Very Low	STOR Low	STOR Nominal	STOR High	STOR Very High	STOR Extra High
A	n/a	n/a	High	Very High	Extra High	Extra High
B	n/a	n/a	High	Very High	Extra High	Extra High
C	n/a	n/a	High	Very High	Extra High	Extra High
D	n/a	n/a	Nominal	Nominal	High	Very High

Fonte: Adaptado de Boehm et al. (2000).

Para efeito didático, tem-se a seguir os passos sugeridos para identificação da classificação deste fator:

- Obtém-se a classificação original do parâmetro STOR conforme sugerido pelo modelo COCOMO II na Tabela A.7;
- Relaciona-se então a classificação original com o nível do software segundo a DO-178C, conforme apresentado na Tabela 7.8;
- O resultado do cruzamento é então a classificação final que deve ser adotada para o fator STOR.

Similar ao parâmetro TIME, observa-se pela Tabela 7.8 que para o caso de software níveis A, B ou C recomenda-se então que a menor classificação possível seja a *High*, de modo a refletir a necessidade da análise de memória. Mesmo no cenário mais favorável, uma aplicação pequena em um hardware com grande capacidade de armazenamento, há de se considerar o esforço adicional para evidenciar tamanha margem de tempo de memória, bem como a análise de possíveis *overlaps* e margens de segurança. Deste modo, para estes níveis de software sugere-se desconsiderar a classificação *Nominal*.

Para o caso do software nível D, mesmo no cenário mais desfavorável, no qual 95% do armazenamento é consumido, sugere-se a maior classificação como

Very High, uma vez que análise de memória não precisa ser demonstrada e um possível esforço adicional para otimização e melhorias do software está mais relacionada com o domínio técnico e boas práticas de programação, com menos impacto no processo e documentação.

Deste modo, a estimativa de esforço reflete tal flexibilidade do software nível D no que diz respeito ao processo.

7.2.8 PVOL – Platform Volatility

O termo “plataforma” é utilizado pelo COCOMO II para traduzir a complexidade do hardware e software (sistema operacional, drivers etc) que o software em questão utiliza para executar suas tarefas, como por exemplo o sistema operacional em que ele está instalado, o banco de dados, software de baixo nível, interface com processador, compiladores etc. Se o software a ser desenvolvido é um sistema operacional, então a plataforma é o hardware do computador. Caso seja um sistema de gerenciamento de banco de dados, a plataforma é o hardware e o sistema operacional. A plataforma inclui quaisquer compiladores ou *assemblers* que suportem o desenvolvimento do software. A análise é feita sob o ponto de vista da quantidade de mudanças de tal plataforma. Esta classificação varia de baixa, onde há uma grande mudança a cada 12 meses, a muito alta, onde há uma grande mudança a cada duas semanas. A Tabela A.8 apresenta a classificação deste fator conforme recomenda o modelo COCOMO II.

Deve ser ressaltado, no entanto, que a frequência de modificação apresentada Tabela A.8 deve ser ajustada para o ciclo de vida do projeto em questão. A Tabela A.8 apresenta a sugestão original do modelo COCOMO II, porém para a realidade de projetos de longa duração pode não fazer sentido considerar mudanças a cada 2 dias, nem que sejam pequenas. Toma-se como exemplo o desenvolvimento de um software crítico conforme apresentado pelo estudo de caso no Capítulo 5, cuja duração foi de cerca de 5 anos. Em projetos desse tipo, não existem mudanças de plataforma a cada 2 dias, porém pode ser considerável pensar que a classificação *Very High* seja cabível a mudanças a cada 3 meses.

Deste modo, a Tabela 7.9 apresenta então a sugestão de classificação do fator PVOL considerando uma frequência de modificação para projetos de mais longa duração, de 4 a 5 anos. Cabe ao aplicante do modelo ajustar a frequência de alteração conforme a escala de classificação de acordo com a realidade esperada do projeto a ter o esforço estimado.

Tabela 7.9 – PVOL: Escala de Frequência adaptada

Fator	Classificação					
	Very Low	Low	Nominal	High	Very High	Extra High
PVOL	n/a	Mudanças grandes a cada 18 meses. Pequenas a cada 6 mês	Mudanças grandes a cada 12 meses. Pequenas a cada 6 semanas	Mudanças grandes a cada 6 meses. Pequenas a cada 3 meses	Mudanças grandes a cada 3 meses. Pequenas a cada mês.	n/a
	n/a	0.87	1.00	1.15	1.30	n/a

Fonte: Adaptado de Boehm et al. (2000).

Do ponto de vista da DO-178C, mudanças relativas à plataforma, ferramentas de desenvolvimento ou suporte, ou o ambiente onde o software é desenvolvido deve ser controlado pelo processo de controle de configuração – *Software Configuration Management Process (SCMP).* Qualquer mudança desse tipo deve passar por uma análise de impacto, para verificar o que deve ser alterado e testado novamente em função de tal mudança.

Deste modo, para cada mudança uma série de atividades e objetivos devem sem demonstrados novamente. A tabela da DO-178C no ANEXO H detalha os objetivos relacionados com esse processo.

A Tabela 7.10 apresenta o cruzamento da classificação obtida pela Tabela 7.9 com o nível do software conforme DO-178C, resultando então na nova classificação a ser adotada na estimativa.

Tabela 7.10 – PVOL: Escala de Classificação adaptada

DO-178C	PVOL Very Low	PVOL Low	PVOL Nominal	PVOL High	PVOL Very High	PVOL Extra High
A	n/a	Nominal	Nominal	High	Very High	n/a
B	n/a	Nominal	Nominal	High	Very High	n/a
C	n/a	Nominal	Nominal	High	High	n/a
D	n/a	Nominal	Nominal	Nominal	High	n/a

Fonte: Adaptado de Boehm et al. (2000).

Para efeito didático, tem-se a seguir os passos sugeridos para identificação da classificação deste fator:

- Obtém-se a classificação original do parâmetro PVOL conforme sugerido pelo modelo COCOMO II na Tabela A.8;
- Relaciona-se então a classificação original com o nível do software segundo a DO-178C, conforme apresentado na Tabela 7.10;
- O resultado do cruzamento é então a classificação final que deve ser adotada para o fator PVOL.

Uma vez determinada a frequência de modificação a ser considerada na Tabela 7.9, a justificativa para tal relação com o nível de software da DO-178C apresentado na Tabela 7.10 se dá pelo seguinte motivo: A tabela da DO-178C no ANEXO H deixa claro que todos os objetivos devem ser atingidos para todos os níveis de software, deste modo a menor classificação sugerida é a *Nominal.* O esforço adicional para executar a análise de impacto das mudanças, bem como possíveis testes de regressão, é então representada conforme o nível de software. Software níveis A e B possuem mais objetivos a serem demonstrados, mais testes para serem executados e mais análises de mudanças, como tempo e consumo de memória por exemplo. Deste modo, recebem a maior classificação sugerida pelo COCOMO II.

7.2.9 ACAP – Analyst Capability

Considera-se “analista” neste fator as pessoas que trabalham nos requisitos, arquitetura e detalhamento (*design detail*). Os principais atributos que devem ser considerados na análise deste fator são: habilidade do analista, eficiência e rigor, e habilidade de comunicação e cooperação com os demais membros do time. Esta análise não deve considerar o nível de experiência do analista, pois para isto existem os fatores APEX, LTEX e PLEX. Equipes de analistas que estejam dentro do 15º percentil, devem ser classificadas como “*very low”* e, por outro lado, aqueles que estejam no 90º percentil devem ser classificados como “*very high”.* A Tabela A.9 apresenta a classificação deste fator conforme recomenda o modelo COCOMO II.

Do ponto de vista da DO-178C, quanto maior o nível do software maior o rigor no processo de desenvolvimento e necessidade de registro e evidência das atividades executadas. As atividades executadas para demonstração do cumprimento dos objetivos da norma devem possuir instrução de trabalho e um passo a passo detalhado. Registrar tais atividades, obviamente, toma tempo dos analistas e engenheiros, por outro lado tornam a execução das atividades menos dependente das pessoas e mais vinculadas ao processo.

Deste modo, sugere-se então na Tabela 7.11 a relação com a DO-178C de modo que quanto maior o nível do software, menos dependente da capacidade pessoal.

Tabela 7.11 – ACAP: Escala de Classificação adaptada

DO-178C	ACAP Very Low	ACAP Low	ACAP Nominal	ACAP High	ACAP Very High	ACAP Extra High
A	Low	Low	Nominal	High	Very High	n/a
B	Low	Low	Nominal	High	Very High	n/a
C	Low	Low	Nominal	High	Very High	n/a
D	Very Low	Very Low	Low	Nominal	High	n/a

Fonte: Adaptado de Boehm et al. (2000).

Para efeito didático, tem-se a seguir os passos sugeridos para identificação da classificação deste fator:

- Obtém-se a classificação original do parâmetro ACAP conforme sugerido pelo modelo COCOMO II na Tabela A.9;
- Relaciona-se então a classificação original com o nível do software segundo a DO-178C, conforme apresentado na Tabela 7.11;
- O resultado do cruzamento é então a classificação final que deve ser adotada para o fator ACAP.

Observa-se na Tabela 7.11 que, para software níveis A, B ou C, a maior classificação é considerada *"low",* ou seja, mesmo com um time menos eficiente o fator multiplicativo de custo não deve ser selecionado como o maior sugerido pelo COCOMO II. Isso se deve pelo fato de que para os três níveis indicados, há a necessidade de documentação dos requisitos de alto-nível, arquitetura e requisitos de baixo nível. Para software nível D, no entanto, a

dependência da capacidade pessoal é maior, uma vez que o processo é menos documentado. Conforme a tabela da DO-178C no ANEXO B, para este nível de software, não existem os objetivos de verificação dos requisitos de baixo nível e código fonte. Sem a documentação de tais requisitos, a dependência da capacidade do analista pode ser maior. Com isso a menor classificação recomendada é a *High.*

7.2.10 PCAP – Programmer Capability

Neste fator a figura do programador deve ser analisada, porém com o ponto de vista no time e não especificamente em indivíduos. Assim como no ACAP, os principais atributos que devem ser considerados na análise deste fator são: habilidade do programador, eficiência e rigor, e habilidade de comunicação e cooperação com os demais membros do time. Novamente, a experiência do programador não deve ser considerada. A Tabela A.10 apresenta a classificação deste fator conforme recomenda o modelo COCOMO II.

Do ponto de vista da DO-178C e em projetos *safety-critical*, a figura do programador é possivelmente menos relevante que em outros projetos. O maior rigor da DO-178C está no processo e nas fases que antecedem a programação. Um bom processo de desenvolvimento de software seguindo a norma DO-178C possui o processo em si devidamente documentado nas fases iniciais, os requisitos capturados, revisados e rastreados contra os seus requisitos “pais”, ou seja, os requisitos de sistemas. Na fase de *Design*, a DO-178C exige uma boa documentação da arquitetura do software e a captura dos requisitos de baixo nível, que também podem ser entendidos como *Design Steps* (RIERSON, 2013), ou seja, o passo a passo sobre como implementar os requisitos de alto nível. Desta forma, o programador tem o papel de transformar tais requisitos de baixo nível em código, o que torna sua tarefa simplificada, uma vez que tudo o que ele deve fazer no código já está escrito, definido e revisado.

Deste modo, sugere então na tabela Tabela 7.12 a relação com a DO-178C de modo que quanto maior o nível do software, menos dependente da capacidade do programador.

Tabela 7.12 – PCAP: Escala de Classificação adaptada

DO-178C	PCAP Very Low	PCAP Low	PCAP Nominal	PCAP High	PCAP Very High	PCAP Extra High
A	Low	Low	Nominal	High	Very High	n/a
B	Low	Low	Nominal	High	Very High	n/a
C	Low	Low	Nominal	High	Very High	n/a
D	Very Low	Very Low	Low	Nominal	High	n/a

Fonte: Adaptado de Boehm et al. (2000).

Para efeito didático, tem-se a seguir os passos sugeridos para identificação da classificação deste fator:

- Obtém-se a classificação original do parâmetro PCAP conforme sugerido pelo modelo COCOMO II na Tabela A.10;
- Relaciona-se então a classificação original com o nível do software segundo a DO-178C, conforme apresentado na Tabela 7.12;
- O resultado do cruzamento é então a classificação final que deve ser adotada para o fator PCAP.

Observa-se na Tabela 7.12 que, para software níveis A, B ou C, a maior classificação é considerada *"low",* ou seja, mesmo com um time menos eficiente o fator multiplicativo de custo não deve ser selecionado como o maior sugerido pelo COCOMO II. Isso se deve pelo fato de que para os três níveis indicados, há a necessidade de documentação dos requisitos de alto-nível, arquitetura e requisitos de baixo nível. Para software nível D, no entanto, a dependência da capacidade pessoal é maior, uma vez que o processo é menos documentado. Conforme a tabela da DO-178C no ANEXO B, para este nível de software, não existem os objetivos de verificação dos requisitos de baixo nível e código fonte. Sem a documentação de tais requisitos, a dependência da capacidade do programador é maior, pois ele não conta com os *Design Steps* previamente documentados. Neste caso, sua tarefa será transformar um requisito de alto-nível direto em código fonte. Com isso a menor classificação recomendada é a *High.*

7.2.11 PCON – Personnel Continuity

A seleção do nível PCON é feita em termos do *turnover* anual da equipe do projeto, ou seja, o % de pessoas que deixam o time ao longo do desenvolvimento. Deste modo, objetiva-se avaliar então o nível de continuidade da equipe. A Tabela A.11 apresenta a classificação deste fator conforme recomenda o modelo COCOMO II.

Similar aos fatores ACAP e PCAP, do ponto de vista da DO-178C, quanto maior o nível do software maior o rigor no processo de desenvolvimento e necessidade de registro e evidência das atividades executadas. Essa necessidade traz a vantagem de tornar a execução das atividades menos dependente das pessoas e mais vinculadas ao processo.

Deste modo, sugere-se então na Tabela 7.13 a relação com a DO-178C de modo que quanto maior o nível do software, menos dependente da capacidade pessoal.

Tabela 7.13 – PCON: Escala de Classificação adaptada

DO-178C	PCON Very Low	PCON Low	PCON Nominal	PCON High	PCON Very High	PCON Extra High
A	Low	Low	Nominal	High	Very High	n/a
B	Low	Low	Nominal	High	Very High	n/a
C	Low	Low	Nominal	High	Very High	n/a
D	Very Low	Very Low	Low	Nominal	High	n/a

Fonte: Adaptado de Boehm et al. (2000).

Para efeito didático, tem-se a seguir os passos sugeridos para identificação da classificação deste fator:

- Obtém-se a classificação original do parâmetro PCON conforme sugerido pelo modelo COCOMO II na Tabela A.11;
- Relaciona-se então a classificação original com o nível do software segundo a DO-178C, conforme apresentado na Tabela 7.13;
- O resultado do cruzamento é então a classificação final que deve ser adotada para o fator PCON.

Observa-se na Tabela 7.13 que, para software níveis A, B ou C, a maior classificação é considerada *“low”,* ou seja, mesmo com um time menos experiente, em função do maior *turnover*, o fator multiplicativo de custo não deve ser selecionado como o maior sugerido pelo COCOMO II. Isso se deve pelos mesmos motivos apontados nos fatores ACAP e PCAP. Para software nível D, no entanto, a dependência da equipe é maior, uma vez que o processo é menos documentado. Uma maior rotatividade de pessoal exigirá um maior número de treinamentos e entendimento do código e arquitetura. Conforme a tabela da DO-178C no ANEXO B, para este nível de software, não existem os objetivos de verificação dos requisitos de baixo nível e código fonte. Sem a documentação de tais requisitos, a dependência da capacidade da equipe pode ser maior. Com isso a menor classificação recomendada é a *High*.

7.2.12 APEX – Application Experience

Este fator depende do nível de experiência na aplicação que possui o time de desenvolvimento do software em questão, medido em tempo. A análise deve ser feita sob o ponto de vista do time de desenvolvimento e não de uma pessoa específica. A Tabela A.12 apresenta a classificação deste fator conforme recomenda o modelo COCOMO II.

Similar aos fatores ACAP, PCAP e PCON, do ponto de vista da DO-178C, quanto maior o nível do software maior o rigor no processo de desenvolvimento e necessidade de registro e evidência das atividades executadas. Essa necessidade traz a vantagem de tornar a execução das atividades menos dependente das pessoas e mais vinculadas ao processo. Obviamente, há de se considerar que caso haja um time todo menos experiente, é recomendado pelo menos uma pessoa com nível mais sênior, de modo que possa atuar como mentor dos demais membros do time.

Deste modo, sugere-se então na Tabela 7.14 a relação com a DO-178C de modo que quanto maior o nível do software, menos dependente da experiência pessoal.

Tabela 7.14 – APEX: Escala de Classificação adaptada

DO-178C	APEX Very Low	APEX Low	APEX Nominal	APEX High	APEX Very High	APEX Extra High
A	Low	Low	Nominal	High	Very High	n/a
B	Low	Low	Nominal	High	Very High	n/a
C	Low	Low	Nominal	High	Very High	n/a
D	Very Low	Very Low	Low	Nominal	High	n/a

Fonte: Adaptado de Boehm et al. (2000).

Para efeito didático, tem-se a seguir os passos sugeridos para identificação da classificação deste fator:

- Obtém-se a classificação original do parâmetro APEX conforme sugerido pelo modelo COCOMO II na Tabela A.12;
- Relaciona-se então a classificação original com o nível do software segundo a DO-178C, conforme apresentado na Tabela 7.14;
- O resultado do cruzamento é então a classificação final que deve ser adotada para o fator APEX.

Observa-se na Tabela 7.14 que, para software níveis A, B ou C, a maior classificação é considerada *"low",* ou seja, mesmo com um time menos experiente o fator multiplicativo de custo não deve ser selecionado como o maior sugerido pelo COCOMO II. Isso se deve pelos mesmos motivos apontados nos fatores ACAP, PCAP e PCON. Para software nível D, no entanto, a dependência da equipe é maior, uma vez que o processo é menos documentado. Com isso a menor classificação recomendada é a *High*.

7.2.13 PLEX – Platform Experience

O modelo COCOMO II considera a influência na produtividade do nível de experiência do time de desenvolvimento na plataforma em questão, reconhecendo a importância de entender o uso de plataformas mais poderosas, incluindo mais recursos de interface gráfica do usuário, banco de dados, rede etc. A Tabela A.13 apresenta a classificação deste fator conforme recomenda o modelo COCOMO II.

Similar aos fatores ACAP, PCAP, PCON e APEX, do ponto de vista da DO-178C, quanto maior o nível do software maior o rigor no processo de desenvolvimento e necessidade de registro e evidência das atividades executadas. Essa necessidade traz a vantagem de tornar a execução das atividades menos dependente das pessoas e mais vinculadas ao processo.

Deste modo, sugere-se então na Tabela 7.15 a relação com a DO-178C de modo que quanto maior o nível do software, menos dependente da experiência pessoal.

Tabela 7.15 – PLEX: Escala de Classificação adaptada

DO-178C	PLEX Very Low	PLEX Low	PLEX Nominal	PLEX High	PLEX Very High	PLEX Extra High
A	Low	Low	Nominal	High	Very High	n/a
B	Low	Low	Nominal	High	Very High	n/a
C	Low	Low	Nominal	High	Very High	n/a
D	Very Low	Very Low	Low	Nominal	High	n/a

Fonte: Adaptado de Boehm et al. (2000).

Para efeito didático, tem-se a seguir os passos sugeridos para identificação da classificação deste fator:

- Obtém-se a classificação original do parâmetro PLEX conforme sugerido pelo modelo COCOMO II na Tabela A.13;
- Relaciona-se então a classificação original com o nível do software segundo a DO-178C, conforme apresentado na Tabela 7.15;
- O resultado do cruzamento é então a classificação final que deve ser adotada para o fator PLEX.

Observa-se na Tabela 7.15 que, para software níveis A, B ou C, a maior classificação é considerada *"low",* ou seja, mesmo com um time menos experiente na plataforma o fator multiplicativo de custo não deve ser selecionado como o maior sugerido pelo COCOMO II. Isso se deve pelos mesmos motivos apontados nos fatores anteriores, além do fato da DO-178C ser uma norma de processo, menos voltada aos aspectos técnicos de plataforma ou linguagem de programação. Para software nível D, no entanto, a

dependência da experiência da equipe é maior, uma vez que o processo é menos documentado. Com isso a menor classificação recomendada é a *High*.

7.2.14 LTEX – Language and Tool Experience

Este fator, por sua vez, considera o nível de experiência do time de desenvolvimento na linguagem de programação adotada e nas ferramentas de software utilizadas. Os fatores considerados aqui devem ser: ferramentas de criação de diagramas para representação da arquitetura, ferramentas de controle de configuração, gerenciamento de bibliotecas, padrão de código etc. A Tabela A.14 apresenta a classificação deste fator conforme recomenda o modelo COCOMO II.

Do ponto de vista da DO-178C, o ambiente de desenvolvimento deve ser estabelecido e controlado. As ferramentas utilizadas no ciclo de vida do desenvolvimento devem ter suas verões e possíveis problemas identificados e reportados. Quanto maior o nível do software, maior a necessidade de tal controle.

Obviamente, quanto maior a experiência da equipe em tais ferramentas, menor será o tempo gasto com a utilização das mesmas, bem como a curva de aprendizado de novos integrantes do time será menor, dado a presença de pessoas experientes na equipe. Todavia, o esforço investido para o controle destas ferramentas continua sendo necessário, independentemente do nível de experiência da equipe.

O parâmetro LTEX também relaciona a experiência do time com relação à padronização do código fonte. Do ponto de vista da DO-178C, existem objetivos claros com relação ao *Code Standard* já na fase de planejamento, conforme ANEXO A objetivo 5.

Deste modo, sugere-se então na Tabela 7.16 a relação com a DO-178C de modo que quanto maior o nível do software, maior o esforço necessário relacionado com este parâmetro.

Tabela 7.16 – LTEX: Escala de Classificação adaptada

DO-178C	LTEX Very Low	LTEX Low	LTEX Nominal	LTEX High	LTEX Very High	LTEX Extra High
A	Very Low	Very Low	Nominal	High	High	n/a
B	Very Low	Very Low	Nominal	High	High	n/a
C	Very Low	Very Low	Nominal	High	High	n/a
D	Low	Nominal	High	Very High	Very High	n/a

Fonte: Adaptado de Boehm et al. (2000).

Para efeito didático, tem-se a seguir os passos sugeridos para identificação da classificação deste fator:

- Obtém-se a classificação original do parâmetro LTEX conforme sugerido pelo modelo COCOMO II na Tabela A.14;
- Relaciona-se então a classificação original com o nível do software segundo a DO-178C, conforme apresentado na Tabela 7.16;
- O resultado do cruzamento é então a classificação final que deve ser adotada para o fator LTEX.

A justificativa para tal relação se dá pelo fato que para softwares níveis A, B ou C há a necessidade de documentação do ambiente de desenvolvimento já na fase de planejamento, conforme apresentado pelo ANEXO A. Além disso, mesmo para software nível D há a necessidade de controlar tais ferramentas, conforme ANEXO H. Com relação a necessidade de padrões de código *(Code Standard)*, há exigência apenas para os níveis A, B e C, conforme ANEXO A e ANEXO E.

Observa-se então na Tabela 7.16 que para os níveis A, B e C foram excluídos as opções *Very High* e *Low*, de modo a deixar claro que para estes níveis de software a experiência entre 3 e 6 anos tem o mesmo efeito no esforço, bem como entre 2 e 6 meses (ver Tabela A.14 para melhor entendimento). Deste modo, a maior exigência de controle do ambiente e padronização de código fica mais evidente em tais níveis de software.

No caso do software nível D, em função do menor rigor da DO-178C e da não exigência de padrão de código, a classificação *Very High* neste caso se aplica a partir dos 3 anos de experiência. De maneira similar, a classificação *Very*

Low foi removida, de modo a refletir que mesmo com o time menos experiente, a própria exigência de documentação e controle de tais ferramentas e traz uma facilidade e agilidade na curva de aprendizado.

7.2.15 TOOL – Use of Software Tools

Este fator considera o uso de ferramentas sofisticadas no desenvolvimento do software. Tais ferramentas podem simplificar e contribuir na redução do esforço necessário. O nível “Very Low” considera o uso apenas de simples ferramentas para edição e escrita do código. A Tabela A.15 apresenta a classificação deste fator conforme recomenda o modelo COCOMO II.

A utilização de ferramentas e automatizações é extremamente necessária no processo de desenvolvimento de software que segue a DO-178C. A necessidade de rastreabilidade entre requisitos, *design* e código, o controle dos artefatos de verificação, *check-lists* e relatórios, bem como o próprio controle de configuração e revisão de todos os artefatos é praticamente impossível sem o uso de ferramentas modernas. No entanto, do ponto de vista da DO-178C, se uma ferramenta elimina, reduz ou automatiza processos estabelecidos pela norma e sua saída não é verificada conforme especificado, tal ferramenta deve ser “qualificadas” para o uso no projeto em questão. A qualificação de uma ferramenta para um determinado projeto deve seguir os processos estabelecidos pela DO-330 (RTCA INC, 2011b).

O processo de qualificação de ferramentas para projetos de desenvolvimento de software crítico pode ser objeto de estudo para um trabalho dedicado sobre este tema. Sendo assim, introduz-se nessa dissertação conceitos básicos para fundamentar o entendimento do leitor sobre o uso de ferramentas qualificadas no processo de desenvolvimento e seu efeito na estimativa de esforço do projeto.

Segundo a DO-178C, o impacto de uma ferramenta no processo deve ser analisado segundo os seguintes critérios:

- Critério 1: Uma ferramenta cuja saída é parte do software embarcado e, portanto, poderia inserir um erro.

- Critério 2: Uma ferramenta que automatiza os processos de verificação e, portanto, pode falhar na detecção de um erro e cuja saída é usada para justificar a eliminação ou redução de:
    - Processos de verificação que não sejam automatizados pela ferramenta, ou
    - Processos de desenvolvimento que podem ter impacto no software embarcado.
- Critério 3: Uma ferramenta que, dentro do escopo de seu uso pretendido, pode falhar na detecção de um erro.

Geradores automáticos de código fonte com base em modelo são bons exemplos de ferramentas que caem no Critério 1. Um analisador estático de código que, por exemplo, automatiza a análise de consumo de pilha, é um exemplo de ferramenta que atende ao Critério 3. Porém, caso o desenvolver deixe de implementar um mecanismo de detecção de possível *overflow* em tempo de execução, com base em tal ferramenta, então passa a atender o Critério 2.

Ferramentas que atendam ao Critério 1, tem um nível de qualificação mais rigoroso (nível 1), ao passo que as ferramentas que atendam ao Critério 3 tem o nível de qualificação mais simplificado (nível 5), conforme a DO-330. Tais níveis de qualificação estabelecidos pela DO-330 (1 ao 5) também variam em função do nível do software estabelecido pela DO-178C (A ao D). Os detalhes sobre cada nível de qualificação, bem como o processo e atividades necessárias para tal, estão dispostos na norma DO-330 e não é parte do escopo dessa dissertação.

Há de se destacar, no entanto, os critérios atendidos pela ferramenta a ser utilizada e seu efeito na estimativa de esforço. Deste modo, sugere-se então na Tabela 7.17 uma adaptação à Tabela A.15 original do COCOMO II, de modo a considerar o uso de ferramentas qualificadas no projeto de desenvolvimento de software seguindo a DO-178C.

Tabela 7.17 – TOOL: Descrição da escala adaptada

Fator	Classificação					
	Very Low	Low	Nominal	High	Very High	Extra High
TOOL	Ferramentas simples com poucos recursos, pouca integração e automatização	Sem uso de ferramentas qualificadas, porém atividades altamente automatizadas, como geradores de código, matriz de rastreabilidade, sistemas de gestão de requisitos.	Ferramentas que atendam o critério 3.	Ferramentas que atendam os critérios 1 OU 2.	Ferramentas que atendam os critérios 1 E (2 OU 3).	n/a
	1.17	1.09	1.00	0.90	0.78	n/a

Fonte: Adaptado de Boehm et al. (2000).

A Tabela 7.17 evidencia o fato do uso de ferramentas qualificadas contribuírem para a redução dos custos de prazos do projeto, uma vez que automatizam e substituem atividades que, do contrário, seriam executadas manualmente (PASTOR; CAULÍN; MARTÍNEZ, 2014).

A Tabela 7.18 apresenta o cruzamento da classificação obtida pela Tabela 7.17 com o nível do software conforme DO-178C, resultando então na nova classificação a ser adotada na estimativa pelo parâmetro TOOL.

Tabela 7.18 – TOOL: Escala de Classificação adaptada

DO-178C	TOOL Very Low	TOOL Low	TOOL Nominal	TOOL High	TOOL Very High	TOOL Extra High
A	Very Low	Low	Nominal	High	High	n/a
B	Very Low	Low	Nominal	High	High	n/a
C	Very Low	Low	Nominal	High	High	n/a
D	Low	Low	Nominal	High	Very High	n/a

Fonte: Adaptado de Boehm et al. (2000).

Para efeito didático, tem-se a seguir os passos sugeridos para identificação da classificação deste fator:

- Obtém-se a classificação do parâmetro TOOL conforme sugerido pela Tabela 7.17;
- Relaciona-se então a classificação com o nível do software segundo a DO-178C, conforme apresentado na Tabela 7.18;

- O resultado do cruzamento é então a classificação final que deve ser adotada para o fator TOOL.

A Tabela 7.18 apresenta que para os níveis A, B e C a classificação do parâmetro TOOL é em função do uso de ferramentas qualificadas para o projeto, variando de *High* para *Very Low*. Nota-se que a opção *Very-High* foi excluída da tabela de classificação, de modo a refletir na estimativa o próprio esforço para se qualificar uma ferramenta. Conforme mencionado, o processo de qualificação de tais ferramentas segue um ciclo de vida próprio e, via de consequência, demandam esforço da equipe para executar suas atividades. Com isso, mesmo contando com ferramentas que atendam aos critérios 1 ou 2, a maior redução na estimativa de esforço é representada pela opção *High.*

Para o nível D, por outro lado, foi excluída a classificação *Very Low*, uma vez que para tal nível de software há uma menor quantidade de objetivos a serem cumpridos, via de consequência a ausência de ferramentas avançadas também causa um impacto menos significativo. A possibilidade da seleção da classificação *Very-High* reflete na estimativa o fato do próprio processo de qualificação de ferramentas para software nível D ser mais simplificado.

7.2.16 SITE – Multisite Development

Segundo o COCOMO II, este fator deve ser analisado de duas formas: (i) co-localização do time, desde totalmente centralizado até internacionalmente distribuído, e (ii) suporte de comunicação, desde simples e-mail e telefone até ferramentas de interação multimídia. A Tabela A.16 apresenta a classificação deste fator conforme recomenda o modelo COCOMO II.

Do ponto de vista da DO-178C, não há uma preocupação específico com a localização física do time. Há, no entanto, a exigência da independência entre as atividades de verificação e desenvolvimento, tal independência é alcançada quando a atividade de verificação é realizada por uma pessoa diferente do desenvolvedor do item que está sendo verificado. Deste modo, a localização de times em diferentes pontos físicos contribui para facilitar a independência das atividades.

Deste modo, a Tabela 7.19 apresenta o cruzamento da classificação original sugerida pelo COCOMO II com o nível do software conforme DO-178C, resultando então na nova classificação a ser adotada na estimativa.

Tabela 7.19 – SITE: Escala de Classificação adaptada

DO-178C	SITE Very Low	SITE Low	SITE Nominal	SITE High	SITE Very High	SITE Extra High
A	Low	Low	Nominal	High	Very High	Very High
B	Low	Low	Nominal	High	Very High	Very High
C	Low	Low	Nominal	High	Very High	Extra High
D	Very Low	Low	Nominal	High	Very High	Extra High

Fonte: Adaptado de Boehm et al. (2000).

Para efeito didático, tem-se a seguir os passos sugeridos para identificação da classificação deste fator:

- Obtém-se a classificação original do parâmetro SITE conforme sugerido pelo modelo COCOMO II na Tabela A.16;
- Relaciona-se então a classificação original com o nível do software segundo a DO-178C, conforme apresentado na Tabela 7.19
- O resultado do cruzamento é então a classificação final que deve ser adotada para o fator SITE.

A justificativa para tal relação apresentada na Tabela 7.19 se dá pelo fato de que para o nível de software A, há a necessidade de independência em 30 dos 71 objetivos relacionados às atividades de verificação. Para o software nível B, a independência é exigida em 18 dos 69 objetivos. Deste modo, para tais níveis foi excluída a opção *Extra High*, de modo a refletir que a total centralização do time no mesmo local físico pode dificultar a independência das atividades, uma vez que facilita a interferência entre os times de desenvolvimento e verificação. No entanto, para software níveis C e D, a exigência de independência entre as atividades é reduzida, tornando a localização centralizada da equipe apenas um agente facilitador. Com isso, a opções *Extra High* está presente.

Pode-se observar também pela Tabela 7.19, que foi excluída a opção *Very Low* para os níveis A, B e C, de modo a refletir que tais níveis de software possuem boa documentação do processo, arquitetura e design do software, tornando

então a distância física menos relevante, uma vez todo o material necessário para entendimento do projeto deve estar disponível e controlado. Já para o software nível D, no entanto, em função da menor necessidade de documentação a maior distância física entre as equipes é um fator complicador, tornando então a opção *Very Low* disponível na tabela.

7.2.17 SCED – Required Development Schedule

Este fator considera as restrições de cronograma impostas sob o time de desenvolvimento de software. A avaliação deve ser feita sob o ponto de vista do percentual do prazo estipulado para o projeto quando comparado ao prazo nominal para um projeto que exige um determinado esforço. Cronogramas apertados exigem maior esforço, ao passo que cronogramas mais relaxados não aliviam a quantidade de esforço, uma vez que o ganho de times menores geralmente é contrabalanceado por necessidade de suporte e gerenciamento por um período de tempo mais longo. A Tabela A.17 apresenta a classificação deste fator conforme recomenda o modelo COCOMO II.

Similar aos Fatores de Escala da seção 7.1, na análise realizada por este autor, o parâmetro SCED não envolve questões técnicas, mas sim puramente aspectos da disciplina de Gerenciamento de Projetos. Em projetos de *software-crítico*, a segurança do produto deve sempre ser a prioridade, de modo que qualquer restrição orçamentária ou prazo afete a confiabilidade do software. Deste modo, sugere-se a aplicação deste parâmetro preferencialmente como “Nominal”, de modo a evitar que restrições de prazo afetem a qualidade do projeto.

7.3 Estimativa de Esforço

Uma vez identificado os Fatores de Escala e os Fatores Multiplicativos de Custo, a estimativa de esforço é obtida exatamente como sugerido pelo modelo COCOMO II. Para evitar repetição de conteúdo, identificam-se abaixo os passos para a estimativa de esforço referenciando as seções deste trabalho dais quais o modelo COCOMO II original foi previamente descrito.

Os passos para o cálculo da estimativa de esforço são:

1. Classificam-se os Fatores de Escala conforme seção 7.1;
2. Classificam-se os Fatores Multiplicativos de Custo conforme seção 7.2;
3. Estima-se o parâmetro SIZE, tamanho do software em KSLOC (número de linhas de código dividido por 1000);
4. Calcula-se o parâmetro PM_{NS} (*Person-months*) em sua versão nominal, conforme equação (2.3) da seção 2.3.2;

Para facilitar a leitura, a equação está repetida abaixo:

$$PM_{NS} = A * Size^{E} * \prod_{i=1}^{16} EM_i$$

O termo NS significa *Nominal-Schedule*, pois exclui o fator multiplicativo denominado SCED, assumindo-se que o projeto tem disponível 100% do tempo necessário para sua realização. Caso existam restrições impostas ao cronograma, o valor de PM não nominal deve ser calculado considerando o parâmetro SCED, conforme passo 5 a seguir:

5. Calcula-se o parâmetro PM (*Person-months*) em sua versão não-nominal, conforme equação (2.5) da seção 2.3.2;

Para facilitar a leitura, a equação está repetida abaixo:

$$PM \quad = A * Size^{E} * \prod_{i=1}^{17} EM_i$$

Após a aplicação destes passos, a estimativa de esforço é então obtida por meio do parâmetro PM (pessoa-mês). Segundo o modelo COCOMO II, uma pessoa-mês é a quantidade de tempo que uma pessoa gasta trabalhando no projeto de desenvolvimento de software por um mês. O modelo COCOMO II considera 1PM = 152 horas.

Cabe aqui ressaltar que uma boa estimativa do tamanho do software, executada no passo 3, é fundamental para um bom modelo de estimativa de esforço. No entanto, estimar o tamanho do software no início do projeto é uma tarefa desafiadora. Existem diversas maneiras para realizar tal estimativa, conforme recomenda o COCOMO II e diversos outros autores (JØRGENSEN;

SHEPPERD, 2007), o uso de dados históricos de outros projetos similares (ver seção 2.2.1) ou opinião especializada (ver seção 2.2.2) são excelentes fontes para fornecer um número provável.

Ainda com relação à estimativa do tamanho do software, há de se enfatizar que o número deve ser baseado em linhas de código executáveis (*logical source statement*), excluindo-se comentários; código fonte de bibliotecas compradas ou inserido pelo compilador (SEI, 1992).

Finalmente, assume-se no modelo proposto que a dificuldade de se implementar um software baseado em modelo é similar à de se programar o software manualmente (ESTRADA; SASAKI; DILLABER, 2013; PASTOR; CAULÍN; MARTÍNEZ, 2014; PAZ; EL BOUSSAIDI, 2016). Deste modo, código fonte gerado por ferramentas de MBD (*Model Based Development*), como por exemplo ANSI SCADE® e Matlab Simulink®, devem ser consideradas na estimativa do tamanho do software no passo em linhas de código. Neste caso, aspectos importantes com relação ao processo de desenvolvimento baseado em modelo devem também considerar a aplicação da DO-331 (RTCA INC, 2011c).

7.4 Estimativa de Prazo

Uma vez estimado o esforço por meio do parâmetro PM, a estimativa de prazo é obtida exatamente como sugerido pelo modelo COCOMO II, aplicando-se a equação (2.6) da seção 2.3.2 para obtenção do parâmetro $TDEV_{NS}$. Para facilitar a leitura, a fórmula é repetida abaixo:

$$TDEV_{NS} = C * (PM_{NS})^{F}$$

O termo NS significa *Nominal-Schedule*, pois exclui o fator multiplicativo denominado SCED, assumindo-se que o projeto tem disponível 100% do tempo necessário para sua realização. Caso existam restrições impostas ao cronograma, o cálculo do TDEV não nominal deve ser feito conforme equação (2.8) da seção 2.3.2. Para facilitar a leitura, a fórmula é repetida abaixo:

$$TDEV \ = TDEV_{NS} * \frac{SCED\%}{100}$$

O cálculo do prazo de desenvolvimento *TDEV* (*Time to Development)* é dados em meses.

7.5 Estimativa da Distribuição da Equipe

Uma vez calculados os parâmetros PM e TDEV, é possível então estimar o tamanho médio da equipe – FSP médio (*Full-time equivalent Software Personnel*) – por meio da equação (2.9) apresentada na seção 2.3.2. Para facilitar a leitura, a fórmula é repetida abaixo:

$$FSP_{\bar{x}} = \frac{PM}{TDEV}$$

Por meio da distribuição de *Rayleigh*, o próprio modelo COCOMO II sugere o cálculo da distribuição do time ao longo dos meses do projeto (TDEV), FSP aproximado, conforme apresentando na equação (2.10) apresentada na seção 2.3.2. Para facilitar a leitura, a fórmula é repetida abaixo:

$$FSP = PM\left(\frac{0.15 * TDEV + 0{,}7t}{0.25 * (TDEV)^2}\right) e^{\frac{(0.15*TDEV+0{,}7t)^2}{0{,}5*(TDEV)^2}}$$

Onde *t* é o tempo em meses ao longo do projeto, variando de 0 (zero) até TDEV.

O próximo capítulo apresenta a aplicação do modelo proposto em dois projetos distintos de *software* crítico, o primeiro em um produto do setor aeronáutico e outro do segmento espacial.

8 APLICAÇÃO DO MODELO PROPOSTO

Relata-se neste capítulo a aplicação do modelo proposto em dois projetos distintos de software crítico, o primeiro em um produto aeronáutico e outro espacial.

O primeiro projeto é o mesmo apresentado no Capítulo 5, no qual o modelo COCOMO II foi aplicado em sua forma original. Desta vez, será aplicado o modelo proposto por essa dissertação, seguindo a nova orientação em função do nível do software segundo a DO-178C. Deste modo, aplicando então os dois modelos no mesmo projeto, será possível a comparação da estimativa gerada por cada uma das abordagens.

O segundo projeto é referente ao satélite FireSat, apresentado pelo exemplo hipotético de desenvolvimento de uma missão espacial (WERTZ; LARSON, 1991).

A aplicação do modelo em ambos projetos segue os passos definidos conforme capítulo 7.

8.1 O Projeto do Software Embarcado da Empresa X

Esta seção apresenta os passos para aplicação do modelo de estimativa de esforço proposto por este trabalho no projeto do Software Embarcado da Empresa X.

8.1.1 Estimativa de Esforço

Passos 1 e 2:

Os passos 1 e 2 da Figura 7.1 não são escopo dessa dissertação por tratem de processos distintos, pois podem ser aplicados de diversas maneiras por diferentes empresas ou instituições. Assume-se então que os passos 1 e 2 já foram previamente executados e, conforme o escopo do projeto descrito na seção 5.4, é sabido que se trata de um projeto DO-178C nível A. Com isso, o nível do software já é conhecido.

Passo 3:

O passo 3 da Figura 7.1 é então a determinação dos fatores de escala. Conforme a seção 7.1, os fatores de escala no modelo proposto devem ser classificados exatamente como sugere o modelo COCOMO II. Deste modo, assume-se exatamente os fatores definidos na seção 5.5, Tabela 5.3. Sendo assim, herda-se também o parâmetro “E” conforme equação (2.4):

E = 0,91 + 0,01 * (3,72+5,07+ 2,83+3,29+1,56) = 1,0747.

Passo 4:

A determinação dos fatores multiplicativos de esforço é a grande contribuição deste trabalho. Desta forma, a classificação será analisada conforme o modelo proposto na seção 7.2. Deste modo, tem-se a seguinte classificação:

Tabela 8.1 – Fatores Multiplicativos de Custo

Fatores de Custo (EM – Effort Multipliers)	Seleção	Valor
Product Attributes		
Required Software Reliability (RELY)	Muito Alta	1,26
Database Size (DATA)	Baixa	0,9
Product Complexity (CPLX)	Extra Alta	1,74
Developed for Reusability (RUSE)	Baixa	0,95
Documentation Match to Life-Cycle Needs (DOCU)	Muito Alta	1,23
Computer Attributes		
Execution Time Constraint (TIME)	Muito Alta	1,29
Main Storage Constraint (STOR)	Alta	1,05
Platform Volatility (PVOL)	Alta	1,15
Personnel Attributes		
Analyst Capability (ACAP)	Baixa	1,19
Programmer Capability (PCAP)	Muito Alta	0,76
Personnel Continuity (PCON)	Muito Alta	0,81
Applications Experience (APEX)	Baixa	1,1
Platform Experience (PLEX)	Nominal	1
Language and Tool Experience (LTEX)	Nominal	1
Project Attributes		
Use of Software Tools (TOOL)	Alta	0,9
Multisite Development (SITE)	Baixa	1,09
Required Development Schedule (SCED)	Nominal	1

Fonte: Produção do autor.

A Tabela 8.1 apresenta então a nova classificação dos fatores multiplicativos de custo seguindo a proposta apresentada na seção 7.2. As linhas em destaque (cor mais escura), significam os parâmetros que foram classificados de maneira diferente de quando foram analisados seguindo o modelo COCOMO II original. Descreve-se abaixo a justificativa para a seleção destes fatores:

O parâmetro **RELY**, que mede o efeito de uma falha de software na função em que ele deve executar durante um período de tempo, foi selecionado como “muito alto”, por ser um software DO-178C nível A, conforme recomendado pela Tabela 7.1. Nota-se que neste item, não houve alteração com relação à seleção original realizada na seção 5.5.

O parâmetro **DATA** havia sido selecionado originalmente como “Nominal”, porém conforme Tabela 7.3, recomenda-se a seleção como “baixo” para projetos DO-178C nível A com o uso de ferramentas qualificadas.

O parâmetro **CPLX** havia sido selecionado originalmente como “Muito Alta”, porém conforme Tabela 7.4, recomenda-se a aplicação da classificação “Extra Alta” para projetos DO-178C nível A.

O parâmetro **RUSE** havia sido selecionado originalmente como “Nominal”, porém conforme Tabela 7.5, recomenda-se a aplicação da classificação “baixa” para projetos DO-178C nível A desenvolvidos sem a intenção de serem reutilizados (RSC - *Reusable Software Component*), o que é o caso deste projeto.

O parâmetro **DOCU**, que é avaliado em termos da necessidade de documentação ao longo do ciclo de vida do projeto em desenvolvimento, foi selecionado como muito alto, por ser um software DO-178C nível A, conforme recomendado pela Tabela 7.6. Nota-se que neste item, não houve alteração com relação à seleção original realizada na seção 5.5.

O parâmetro **TIME** havia sido selecionado originalmente como “alta”, porém conforme Tabela 7.7, recomenda-se a aplicação da classificação MUITO ALTA para projetos DO-178C nível A.

O parâmetro **STOR** havia sido selecionado originalmente como “Nominal”, porém conforme Tabela 7.8, recomenda-se a aplicação da classificação “alta” para projetos DO-178C nível A.

O parâmetro **PVOL** havia sido selecionado originalmente como “Nominal”, porém conforme nova escala de frequência da Tabela 7.9 e cruzamento com o nível A de software da Tabela 7.10, recomenda-se a aplicação da classificação “alta”.

O parâmetro **ACAP** foi mantido com a classificação “baixa”, conforme recomendado na Tabela 7.11.

O parâmetro **PCAP** foi mantido com a classificação “muito alta”, conforme recomendado na Tabela 7.12.

O parâmetro **PCON** foi mantido com a classificação “muito alta”, conforme recomendado na Tabela 7.13.

O parâmetro **APEX** foi mantido com a classificação “baixa”, conforme recomendado na Tabela 7.14.

O parâmetro **PLEX** foi mantido com a classificação “nominal”, conforme recomendado na Tabela 7.15.

O parâmetro **LTEX** foi mantido com a classificação “nominal”, conforme recomendado na Tabela 7.16.

O parâmetro **TOOL** havia sido selecionado originalmente como “Muito Alta”, considerando o critério original apresentado pelo COCOMO II. Porém, trata-se de um projeto com o uso de ferramentas qualificadas pela DO-330 (RTCA INC, 2011b), que atendem aos critérios 1 e 3, conforme descrito na seção 7.2.15. Com isso, dada a nova descrição apresentada na Tabela 7.17 e cruzamento com o nível A de software da Tabela 7.18, recomenda-se a aplicação da classificação “alta”.

O parâmetro **SITE** havia sido selecionado originalmente como “Muito Baixa”, porém conforme Tabela 7.19, recomenda-se a aplicação da classificação “baixo”.

Finalmente, o parâmetro **SCED** havia sido selecionado originalmente como “baixo”, porém foram adotados os valores nominais, conforme descrito na seção 5.5. Conforme recomendado pelo modelo proposto neste trabalho, na 7.2.17, o fator SCED deve ser classificado como “nominal” para projeto de desenvolvimento de software crítico.

Passo 5:

Calcula-se então neste momento a nova estimativa de esforço, o parâmetro PM nominal, conforme seção 7.3. Considera-se o valor nominal, pois o parâmetro SCED será incluído no produto dos fatores multiplicativos de esforço, uma vez que ele foi selecionado como 1 (um). Para o cálculo do PM, assume-se o parâmetro SIZE com o mesmo valor estimado originalmente, 140KSLOC, conforme descrito na seção 5.5. O cálculo do PM deve ser realizado conforme equação (2.5). Tem-se então:

$$\prod_{i=1}^{17} EM_i = 2{,}839$$

PM = 2,94 * ((SIZE)^E) * (Produto(EM_i))

O parâmetro “E” foi calculado no passo 3, deste modo tem-se:

PM = 2,94 * (140)^1,0747 * (2,839) = **1.690 pessoas/mês.**

Passo 6:

Uma vez estimado o esforço, estima-se então o prazo necessário para o desenvolvimento do projeto, conforme seção 7.4. Deste modo, calcula-se então o parâmetro TDEV nominal, conforme equação (2.6):

$$TDEV_{NS} = C * (PM_{NS})^F = 3{,}67 * (1690)^F$$

Onde, F = 0,28 + 0.2 * (1,0747 - 0,91) = 0,31294.

Com isso:

$TDEV_{NS}$ = 3,67 * (1690) ^{0,31294} = **37,5 meses.**

Passo 7:

Uma vez calculados os parâmetros PM e TDEV, é possível então calcular o tamanho médio da equipe conforme seção 7.5, considerando o cronograma nominal, seguindo equação (2.9):

$EQUIPE_{NS} = PM_{NS} / TDEV_{NS} = 1690 / 37,5 =$ **45 pessoas.**

A Tabela 8.1 sumariza então o resulta de todos os principais parâmetros estimados conforme o modelo proposto.

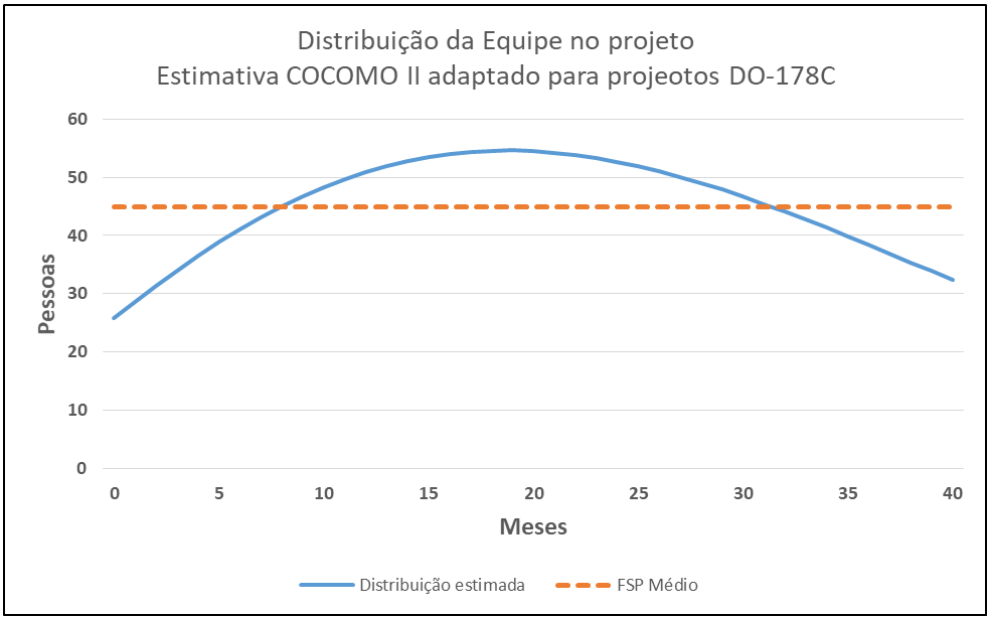
Tabela 8.2 – Valores Estimados pelo modelo proposto

Parâmetro	Esforço Estimado
KLOC	140
PM_{NS} (Pessoa-mês)	1690
$TDEV_{NS}$ (Tempo de desenvolvimento)	37,5 meses
Tamanho médio da Equipe	45 pessoas

Fonte: Produção do autor.

A Figura 8.1 apresenta a estimativa de distribuição da equipe ao longo dos 37,5 meses do projeto, calculada conforme equação (2.10). Os valores da distribuição calculada mês a mês estão disponíveis no APÊNDICE C.

Figura 8.1 – Distribuição estimada da equipe conforme modelo proposto



Fonte: Produção do autor.

A Figura 8.1 apresenta o tamanho médio de 45 pessoas, conforme calculado anteriormente, porém sugere um período de pico, chegando até 55 pessoas.

Apresentados então o esforço estimado e, utilizando o esforço real exigido pelo projeto conforme a seção 5.6, a seção seguinte apresenta uma discussão dos resultados obtidos com a aplicação do modelo proposto.

8.1.2 Avaliação da Estimativa de Esforço

Similar à seção 5.7, adota-se como medida para avaliar a precisão do modelo de estimativa em relação ao realizado a Magnitude do Erro Relativo (MRE - Magnitude of Relative Error), onde:

$$MRE = |esforço\ real - esforço\ estimado| / esforço\ real.$$

A Tabela 8.3 a seguir apresenta lado a lado os valores obtidos pela estimativa e os valores realmente demandados pelo projeto.

Tabela 8.3 – Comparação da estimativa x realizado

Parâmetro	Esforço Estimado	Esforço Real
KLOC	140	170
PM_{NS} (Pessoa-mês)	1690	1532
$TDEV_{NS}$ (Tempo de desenvolvimento)	37,5 meses	54 meses
Tamanho médio da Equipe	45 pessoas	28 pessoas

Fonte: Produção do autor.

Aplicando-se então o cálculo de MRE para análise da precisão do valor estimado de esforço (PM), resulta-se que:

MRE = | 1532 – 1690| / 1532 = 0,1033

O cálculo do MRE mostra que a estimativa apresentou um erro relativo de cerca de 10,33% em relação ao projeto de fato realizado. Tal valor, quando comparado com resultados obtidos pela estimativa realizada com o COCOMO II na seção 5.5, se mostra mais preciso e mais aderente à realidade do projeto. Comparando ainda com a literatura, alguns autores consideram estimativas de boas qualidade somente às que possuem um MRE abaixo de 25% (BASTEN; MELLIS, 2011).

A Tabela 8.4 a seguir apresenta os valores obtidos pela estimativa com o COCOMO II, os valores estimados pelo modelo proposto, bem como os valores reais utilizados pelo projeto, conforme seção 5.6.

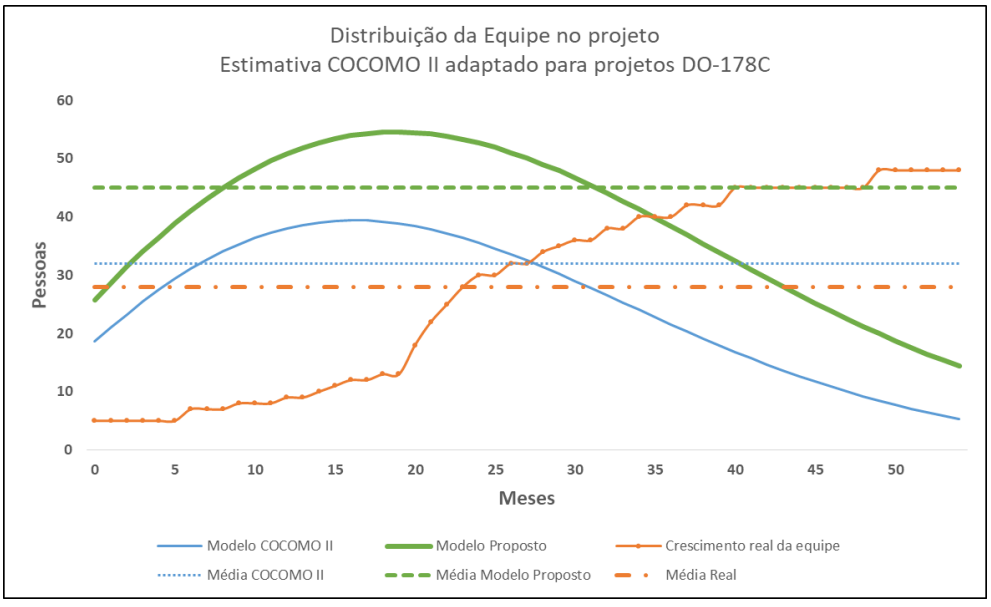
Tabela 8.4 – Comparação das estimativas

Parâmetro	Estimativa COCOMO II	Estimativa Modelo Proposto	Esforço Real
KLOC	140	140	170
Esforço em PM_{NS}	1052	1690	1532
Prazo em $TDEV_{NS}$	32,3 meses	37 meses	54 meses
Tamanho médio da Equipe	32 pessoas	45 pessoas	28 pessoas
MRE (Erro Relativo) Esforço	31,30%	10,33%	-

Fonte: Produção do autor.

Estendendo a distribuição de Rayleigh considerando o prazo real do projeto, isto é, de t=0 à t=54 na equação (2.10), temos então na Figura 8.2 a distribuição estimada pelo modelo COCOMO II original (em azul), a distribuição estimada pelo modelo proposto por esta dissertação (em verde), e a curva de crescimento real do time ao longo do projeto (em laranja), mês a mês, bem como as respectivas médias.

Figura 8.2 – Distribuições estimadas da equipe



Fonte: Produção do autor.

Nota-se pela Figura 8.2 que ambas estimativas acertam em determinado ponto. A distribuição obtida com o modelo COCOMO II cruzou com o crescimento real 28º mês do projeto, ao passo que o modelo proposto por este trabalho cruzou com o crescimento real da equipe no 36º mês do projeto. A justificativa para essa diferença se dá pelo fato do modelo proposto ter sido mais fiel a duração real de 54 meses do projeto, estimando o prazo necessário em 37 meses, contra 32,3 meses estimado pelo COCOMO II. Há de se considerar, no entanto, que as curvas estimadas são distribuições estatísticas e o crescimento real, por sua vez, ocorreu conforme demanda do projeto, disponibilidade de orçamento e mão de obra.

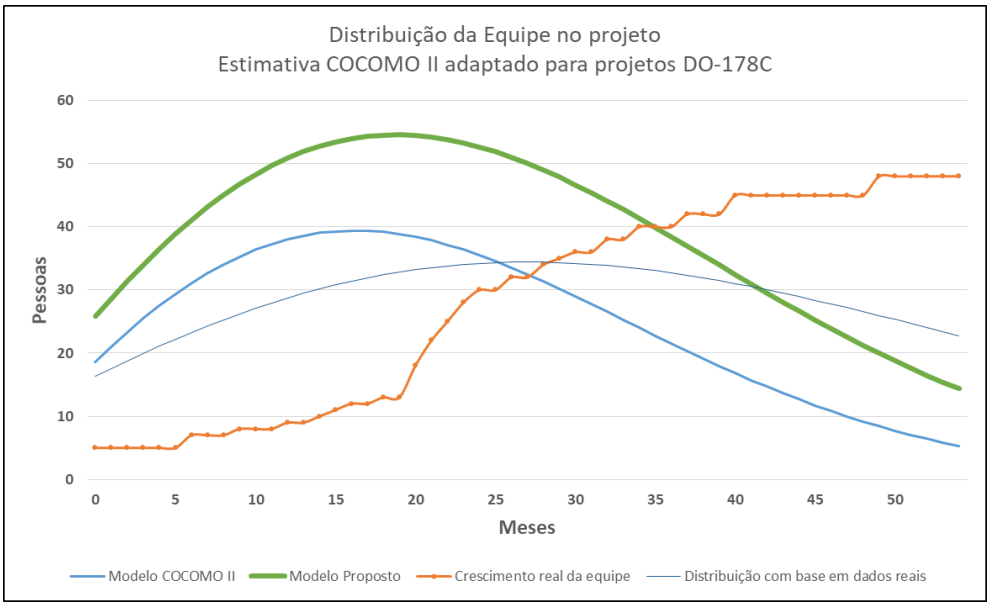
Observa-se ainda, pela Figura 8.2, que os pontos de maior erro entre o real e o estimado se dá no início e fim do projeto. A distribuição estimada pelos dois modelos já começa com valores elevados de quantidade de pessoas, com 18 e 28 pessoas respectivamente, logo no primeiro mês de projeto. Obviamente essa estimativa não se confirmou, pois, o início de um projeto complexo e uma grande empresa sofre com uma inércia para o engajamento de pessoal, em função de diversos fatores, como disponibilidade de recurso financeiro para contratação e mão de obra qualificada no mercado de trabalho. Já na fase final do projeto, a distribuição assume que, a partir de um determinado ponto, começará um desengajamento de pessoal em função da fase final do projeto. Porém, é característica de projetos de desenvolvimento de software crítico, que passam pelo processo de certificação, um aumento no engajamento de pessoal justamente na fase final, momento em que as autoridades de certificação fazem as últimas auditorias antes de emitir a homologação do produto.

Uma característica negativa de projetos deste tipo é que o engajamento máximo é justamente no final, e no dia seguinte à certificação do software já não existe uma grande demanda de mão de obra e, causando então imediato efeito ocioso no time de desenvolvimento. Deste modo, um ajuste mais fino da distribuição do time ao longo do projeto, ou mesmo a seleção de outra distribuição que não seja a distribuição de Rayleigh, pode ser objeto de pesquisa de um futuro trabalho.

Com relação ao tamanho médio da equipe, a estimativa apresentada pelo modelo COCOMO II original foi mais aderente ao que de fato de demandou do projeto. A justificativa para a média da equipe estimada pelo COCOMO II ser menor que a do modelo proposto é o próprio fato do esforço estimado ter sido também menor.

Para efeito comparativo, a Figura 8.3 apresenta as mesmas curvas da Figura 8.2, porém adicionando a distribuição de Rayleigh considerando o prazo e esforço real do projeto, em cinza. As médias foram removidas desta figura apenas para simplificar a leitura do gráfico. Nota-se pela Figura 8.3 que o modelo proposto (curva verde) apresenta uma discrepância maior no que diz respeito à quantidade de pessoas.

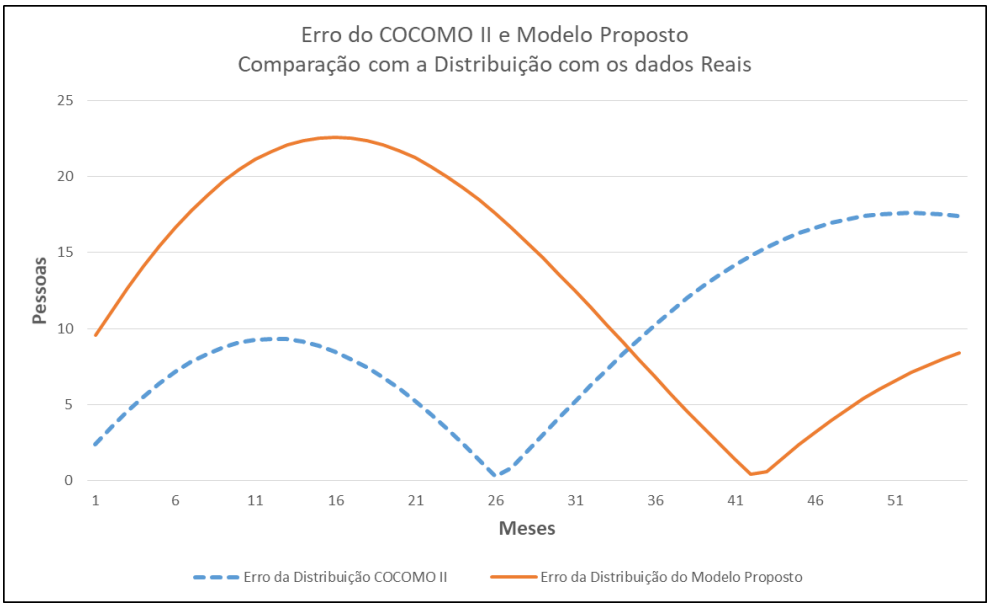
Figura 8.3 – Comparativo das distribuições estimadas da equipe



Fonte: Produção do autor.

Dado então as três distribuições, a Figura 8.4 apresenta o erro em número de pessoas das distribuições de Rayleigh geradas a partir das estimativas do COCOMO II (curva azul) e do modelo proposto (curva laranja). O erro é calculado ao se comparar tais distribuições com a distribuição de Rayleigh gerada com os dados reais do projeto (curva cinza) da Figura 8.3.

Figura 8.4 – Erro das distribuições do COCOMO II e Modelo Proposto



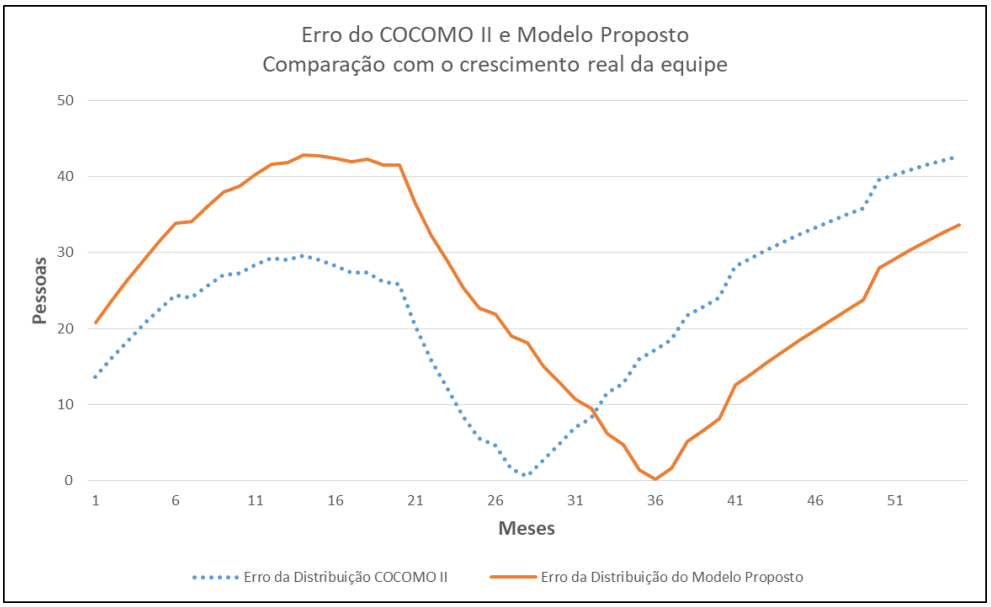
Fonte: Produção do autor.

Conforme mencionado nas análises anteriores, é possível observar pela Figura 8.4 que o modelo proposto apresenta um maior erro com relação ao número de pessoas, porém é mais aderente à distribuição real do meio do projeto em diante, ao passo que o modelo COCOMO II começa a aumentar o seu erro a partir deste mesmo período.

Quando comparadas as distribuições estimadas pelo modelo COCOMO II e pelo modelo proposto com relação ao crescimento real da equipe, curva laranja da Figura 8.2, obtém-se o tamanho do erro de cada distribuição conforme apresentado pela Figura 8.5. Nota-se o mesmo comportamento observado na Figura 8.4, porém com o erro agora indicando a diferença real entre os valores estimados e a quantidade de pessoas realmente engajadas no projeto em determinado mês.

A Figura 8.5 apresenta o erro das distribuições do COCOMO II e Modelo Proposto com relação ao crescimento real da equipe, deixando ainda mais evidente a necessidade de uma melhoria no modelo proposto no que diz respeito a estimativa do tamanho da equipe.

Figura 8.5 – Erro das distribuições do COCOMO II e Modelo Proposto pelo crescimento real da equipe



Fonte: Produção do autor.

Há de se ressaltar também, analisando a Tabela 8.4, que o prazo estimado por ambos modelos ficou bem distante do prazo real, apresentando erros relativos acima dos 30% para as duas estimativas. Deste modo, mesmo com um erro relativo de 10,33% na estimativa do esforço, uma melhor análise e cálculo da estimativa do prazo também pode ser considerado objeto de pesquisa de um futuro trabalho.

A próxima seção apresenta a aplicação do modelo proposto em um projeto de desenvolvimento de software da área espacial.

8.2 O Projeto do FireSat

O projeto FireSat baseia-se no livro SMAD – The Space Mission Analysis and Design Process (WERTZ; LARSON, 1991) e trata-se do exemplo ilustrativo de uma missão espacial, bem como do seu processo de desenvolvimento. Estima-se nesta seção o esforço necessário para o desenvolvimento do software de controle de atitude do FireSat.

Por ser um projeto hipotético, não existem dados reais de esforço e prazo para comparação da estimativa apresentada pelo modelo proposto, tampouco

informações sobre a equipe ou instituição responsável. Deste modo, algumas premissas foram assumidas para obtenção da estimativa, são elas:

- O desenvolvimento do software do FireSat será feito por uma organização experiente, com desenvolvimentos de sucesso realizados anteriormente em produtos lançados ou em operação;
- Por ser o desenvolvimento de um software crítico, o processo de desenvolvimento segue a norma DO-178C;
- A equipe de gerenciamento de projeto tem uma boa análise de risco, cobrindo pelo menos 90% dos riscos inerentes à projetos deste tipo;
- A equipe de desenvolvimento é experiente e atua de forma cooperativa;
- O processo de desenvolvimento é maduro, algo similar ao CMM nível 4 (SEI, 2010);

A estimativa de esforço para o desenvolvimento do software de voo do FireSat aplicando o processo proposto é apresentada então na próxima seção.

8.2.1 Estimativa de Esforço

Passos 1 e 2:

Os passos 1 e 2 da Figura 7.1 não são escopo dessa dissertação por tratem de processos distintos, pois podem ser aplicados de diversas maneiras por diferentes empresas ou instituições. Assume-se então que os passos 1 e 2 já foram previamente executados e, por se tratar do software de controle de atitude, crítico para a missão espacial, assume-se que se trata de um projeto DO-178C nível A. Com isso, o nível do software já é conhecido.

Passo 3:

O passo 3 da Figura 7.1 é então a determinação dos fatores de escala. Conforme a seção 7.1, os fatores de escala no modelo proposto devem ser classificados exatamente como sugere o modelo COCOMO II. Pela própria natureza do modelo, a seleção dos fatores de escala do projeto é algo subjetivo e foi realizada considerando as premissas adotadas na seção 8.2. Deste modo,

a Tabela 8.5 apresenta os fatores de escala selecionados para o projeto FireSat.

Tabela 8.5 – Fatores de Escala – FireSat

Fatores de Escala	Seleção	Valor
PREC (Precedentedness)	Muito Alto	1,24
FLEX (Development Flexibility)	Muito Baixo	5,07
RESL (Architecture / Risk Resolution)	Muito Alto	1,41
TEAM (Team Cohesion)	Alto	2,19
PMAT (Process Maturity)	Muito Alto	1,56

Fonte: Produção do autor.

O parâmetro **PREC**, que indica o nível de similaridade do projeto atual com projetos anteriores, foi selecionado como “muito alto”, dada a premissa de que o software do FireSat será feito por uma organização experiente, com desenvolvimentos de sucesso realizados anteriormente em produtos lançados ou em operação.

O parâmetro **FLEX**, que indica o nível de flexibilidade do software em função do processo de desenvolvimento, das restrições de interface, requisitos e prazo, foi selecionado como “muito baixo”, uma vez que há necessidade de cumprimento com a norma DO-178C ou similar, como por exemplo a NASA-STD-8719.13C (NASA, 2004b)

O parâmetro **RESL**, que reflete o resultado da análise de risco, foi selecionado como “muito alto”, dado a premissa de aproximadamente 90% dos riscos estão mapeados.

O parâmetro **TEAM**, que indica o nível de integração da equipe e fatores humanos, também foi selecionado como “alto”, dado a premissa de que a equipe de desenvolvimento é experiente e atua de forma cooperativa.

Finalmente, o parâmetro **PMAT**, que indica o nível de maturidade dos processos da empresa, foi selecionado como “muito alto” dada a premissa de que o processo de desenvolvimento é maduro, algo similar ao CMM nível 4.

Uma vez definidos os fatores de escala, foi possível então calcular o parâmetro “E” conforme equação (2.4):

*E = 0,91 + 0,01 * (1,24+5,07+ 1,41+ 2,19+1,56) = 1,0247.*

Observa-se então que o projeto apresenta E > 1,0. Conforme Tabela 2.5, tem-se então um projeto com aumento de custos conforme aumento de escala. Isto significa que se o tamanho do projeto dobrar, por exemplo, seus custos mais que dobram, algo também na estimativa do projeto realizado pela Empresa X.

Passo 4:

A determinação dos fatores multiplicativos de esforço também adota as mesmas premissas descritas anteriormente. A classificação será analisada conforme o modelo proposto na seção 7.2. Deste modo, tem-se a seguinte classificação:

Tabela 8.6 – Fatores Multiplicativos de Custo – Projeto FireSat

Fatores de Custo (EM – Effort Multipliers)	Seleção	Valor
Product Attributes		
Required Software Reliability (RELY)	Muito Alta	1,26
Database Size (DATA)	Baixa	0,9
Product Complexity (CPLX)	Extra Alta	1,74
Developed for Reusability (RUSE)	Baixa	0,95
Documentation Match to Life-Cycle Needs (DOCU)	Muito Alta	1,23
Computer Attributes		
Execution Time Constraint (TIME)	Alta	1,11
Main Storage Constraint (STOR)	Alta	1,05
Platform Volatility (PVOL)	Nominal	1
Personnel Attributes		
Analyst Capability (ACAP)	Nominal	1
Programmer Capability (PCAP)	Nominal	1
Personnel Continuity (PCON)	Nominal	1
Applications Experience (APEX)	Nominal	1
Platform Experience (PLEX)	Nominal	1
Language and Tool Experience (LTEX)	Nominal	1
Project Attributes		
Use of Software Tools (TOOL)	Alta	0,9
Multisite Development (SITE)	Baixa	1,09
Required Development Schedule (SCED)	Nominal	1

Fonte: Produção do autor.

O parâmetro **RELY** foi selecionado como “muito alto”, pois por ser um software de controle de voo de um veículo espacial, cujos requisitos de confiabilidade e

segurança do produto são dos mais elevados, uma vez que qualquer acidente pode levar risco para vidas humanas.

O parâmetro **DATA** foi selecionado como “baixo”, pois assume-se que um desenvolvimento de tamanha complexidade deve contar com o auxílio de ferramentas qualificadas.

O parâmetro **CPLX**, que mede a complexidade do produto; foi selecionado como “extra alta” por razões obvias.

O parâmetro **RUSE** foi selecionado como “baixo”, assume-se que não há intenção de se desenvolver tal software para ser reutilizado no futuro. Experiências passadas como no ocorrido com o ARIANE 5 mostram que o reuso de software deve ser feito com muita cautela (TERRY BAHILL; HENDERSON, 2005).

O parâmetro **DOCU**, que é avaliado em termos da necessidade de documentação ao longo do ciclo de vida do projeto em desenvolvimento, foi selecionado como “muito alto” em função da quantidade de documentos necessários ao longo do ciclo de vida de um projeto de desenvolvimento de software nível A, conforme a DO-178C (RTCA INC, 2011a).

O parâmetro **TIME**, que é a medida relacionada com a restrição para o tempo de execução do software, foi selecionado como “alta”, uma vez que para o software de controle de atitude do FireSat foi estimado a necessidade de um processador com capacidade de 356KIPS (WERTZ; EVERETT; PUSCHELL, 2011), o que era facilmente suportado por computadores disponíveis no mercado na época do desenvolvimento, conforme mostra ANEXO K.

O parâmetro **STOR**, que representa o grau de restrição do principal meio de armazenamento de dados do sistema, foi selecionado como “alto”, uma vez que para o software de controle de atitude do FireSat foi estimado a necessidade de um computador com capacidade de 44K *words* de dados (WERTZ; EVERETT; PUSCHELL, 2011), o que era facilmente suportado por computadores disponíveis no mercado na época do desenvolvimento, conforme mostra ANEXO K.

O parâmetro **PVOL**, que é medido sob o ponto de vista da quantidade de mudanças de plataforma, foi selecionado como “nominal”, uma vez que se assume não haver mudanças grandes de plataforma muito frequentes.

Os parâmetros **ACAP, PCAP, PCON, APEX, PLEX** e **LTEX**, que analisam atributos pessoais da equipe, foram todos selecionados como “nominal”, em função de premissa estabelecida de que a equipe de desenvolvimento é experiente e atua de forma cooperativa. No entanto, a escolha pela classificação nominal tem o objetivo de evitar que tal premissa afete de maneira muito significativa a estimativa de esforço final.

O parâmetro **TOOL** foi selecionado como “alto”, pois assume-se que um desenvolvimento de tamanha complexidade deve contar com o auxílio de ferramentas qualificadas.

O parâmetro **SITE** foi selecionado como “baixo”, pois se assume que desenvolvimentos de sistemas complexos, como um veículo espacial, envolva diversos fornecedores espalhados pelo mundo.

Finalmente, o parâmetro **SCED** foi selecionado como “nominal”, conforme recomendado pelo modelo proposto neste trabalho, na 7.2.17.

Passo 5:

Calcula-se então neste momento estimativa de esforço, o parâmetro PM, conforme seção 7.3. Para o cálculo do PM, utiliza-se o parâmetro SIZE igual a 26KSLOC, conforme estimado pela equipe do projeto FireSat (WERTZ; EVERETT; PUSCHELL, 2011). O cálculo do PM deve ser realizado conforme equação (2.5). Tem-se então:

$$\prod_{i=1}^{17} EM_i = 2{,}636163271$$

PM = 2,94 * ((SIZE)^E) * (Produto(EM_i))

O parâmetro “E” foi calculado no passo 3, deste modo tem-se:

PM = 2,94 * ((26)^ 1,0247) * (2,636163271) = **218 pessoas/mês.**

Passo 6:

Uma vez estimado o esforço, estima-se então o prazo necessário para o desenvolvimento do projeto, conforme seção 7.4. Deste modo, calcula-se então o parâmetro TDEV, conforme equação (2.6):

$TDEV_{NS} = C * (PM_{NS})^F = 3{,}67 * (218)^F$

Onde, F = 0,28 + 0.2 * (1,0247- 0,91) = 0,30294.

Com isso:

$TDEV_{NS} = 3{,}67 * (218)^{0{,}30294}$ = **18,76 meses.**

Passo 7:

Uma vez calculados os parâmetros PM e TDEV, é possível então calcular o tamanho médio da equipe conforme seção 7.5, considerando o cronograma nominal, seguindo equação (2.9):

$EQUIPE_{NS} = PM_{NS} / TDEV_{NS} = 218 / 18{,}76 =$ **12 pessoas.**

A Tabela 8.7 sumariza então o resulta de todos os principais parâmetros estimados conforme o modelo proposto.

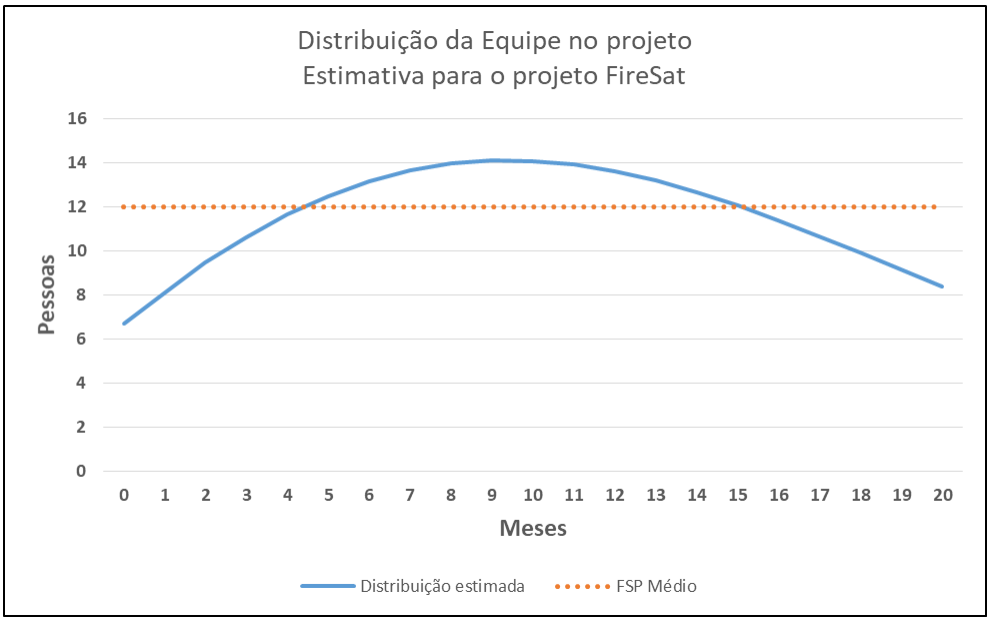
Tabela 8.7 – Valores Estimados para o projeto FireSat

Parâmetro	Esforço Estimado
KLOC	26
PM_{NS} (Pessoa-mês)	218
$TDEV_{NS}$ (Tempo de desenvolvimento)	18,7 meses
Tamanho médio da Equipe	12 pessoas

Fonte: Produção do autor.

A Figura 8.6 apresenta a estimativa de distribuição da equipe ao longo dos meses do projeto pode ser estimada conforme equação (2.10). Os valores da distribuição calculada mês a mês estão disponíveis no APÊNDICE D.

Figura 8.6 – Distribuição estimada da equipe conforme modelo proposto



Fonte: Produção do autor.

Observa-se pela Figura 8.6 o tamanho médio da equipe em 12 pessoas, conforme calculado anteriormente, com um período de pico chegando até 14 pessoas.

8.2.2 Avaliação da Estimativa de Esforço

Conforme mencionado anteriormente, o projeto FireSat é um exemplo hipotético adotado pelos autores do livro SMAD para ilustrar o processo de desenvolvimento de uma missão espacial, deste modo não existem dados reais de esforço e prazo para comparação da estimativa apresentada pelo modelo proposto. Com isso, não se faz possível o cálculo do MRE. No entanto, o SMAD adota o seu próprio modelo de estimativa de custo com base em custo por linhas de código. Deste modo, torna possível comparar a estimativa com as duas abordagens, (1) modelo de custo do SMAD, e (2) modelo de estimativa apresentado por essa dissertação. A Tabela 8.8 apresenta a custo do software estimado pelo modelo utilizado pelo SMAD.

Tabela 8.8 – Estimativa de custo pelo SMAD

Software	Custo Estimado
Software de Voo	USD 435K x KLOC
Software de Solo	USD 220K x KLOC

Fonte: Adaptado de Wertz et al. (2011)

Nota-se que o custo do software de voo é maior que o software de solo, uma vez que segundo os próprios autores há uma maior necessidade de testes em função da criticidade de tal software. Essa diferenciação pelo nível de criticidade do software vai ao encontro do modelo de estimativa de custo apresentado por essa dissertação, na qual introduz os níveis apresentados pela DO-178C para diferenciar a criticidade do software.

Adotando então o modelo de custo do SMAD, o software do FireSat pode ser estimado em USD 435K x 26KLOC = USD 11.310.000,00. No entanto, o SMAD sugere um fator 0.5 para o cálculo do custo deste projeto, em função da existência de projetos anteriores similares a este, o que possibilita realizar modificações no software de voo já existente. Com isso, o software do FireSat pode ser estimado em USD 435K x 26KLOC x 0.5 = **USD 5.655.000,00.**

A mesma premissa assumida pelo modelo do SMAD também é considerada na aplicação modelo da proposta deste trabalho por meio do fator de escala **PREC**, que foi selecionado como “muito alto”, conforme Tabela 8.5.

Segundo o modelo COCOMO II, há um risco em estimar os custos de mão-de-obra em dólares devido às grandes variações entre as organizações no que está incluído em tais custos, como custo de operação, aluguel, impostos etc. O cálculo do esforço em PM (pessoa-mês) se mostra mais estável do que os dólares, dadas as taxas de inflação atuais e as flutuações internacionais de câmbio, bem como facilita a adoção do custo por hora utilizado em cada organização. Sendo assim, de modo a viabilizar a comparação com o modelo do SMAD, se assume como referência 1PM = 152 horas e uma taxa hora de USD 200 dólares, valores estes utilizados por exemplo pela Empresa X. Deste modo, o custo do software segundo o modelo proposto por essa dissertação seria estimado em 218PM x 152 horas x USD 200,00 = **USD 6.639.210,00.**

A Tabela 8.9 apresenta o resultado final da estimativa de custo realizada pelas duas diferentes abordagens em valores financeiros.

Tabela 8.9 – Estimativa de custo para o FireSat em USD

Modelo de Estimativa	FireSat - Custo Estimado
SMAD	USD 5.655.000,00
COCOMO II adaptado DO-178C	USD 6.639.210,00

Fonte: Produção do autor.

Note-se que, dadas as premissas, o custo apresentado pelo modelo proposto por este trabalho é cerca de 20% maior que o custo apresentado pelo SMAD. Essa diferença poderia ser explicada pelo fato da utilização da DO-178C no processo e o rigor em documentação, verificação e teste. Porém, diversas outras hipóteses podem ser levantadas para justificar tal diferença, como por exemplo o uso de premissas não realistas, custo por hora muito elevado; estimativa do tamanho do software em 26KLOC muito otimista ou pessimista; etc. No entanto, pode-se concluir que para um mesmo projeto, assumindo as mesmas premissas, o modelo proposto por esse trabalho apresentou resultados coerentes com o outro modelo utilizado originalmente. Infelizmente, em função da ausência de dados reais do projeto, não há como se calcular o MRE para verificar qual das estimativas foi a mais aderente à realidade.

9 CONCLUSÃO

Essa dissertação propôs um modelo de estimativa de esforço para projetos de desenvolvimento de software críticos, baseado na adaptação do modelo COCOMO II de modo a considerar os aspectos de segurança definidos pela norma RTCA DO-178C. A proposta é que o modelo direcione a tomada de decisão dos gestores do projeto com relação ao tamanho da equipe e prazo para desenvolvimento do software, de modo a contribuir com o correto dimensionamento do esforço necessário, evitando assim que tal projeto seja parte das estatísticas negativas apresentadas pela literatura de projetos cancelados, atrasados ou acima do orçamento estimado.

O uso do modelo COCOMO II como base do modelo proposto ofereceu maior segurança para sua aplicação, pois parte de um modelo bem documentado, com vasto material disponível na literatura e com anos de aplicação pelas indústrias e academia.

O diferencial desta dissertação para com os trabalhos correlatos é a aplicação do modelo de estimativa de esforço com o foco em software crítico, mais especificamente para software que seguem os processos estabelecidos pela DO-178C. Os trabalhos correlatos podem ser utilizados como referência na realização da estimativa de esforço ou para aplicação da DO-178C, porém trabalhos que apresentassem a junção destes dois conceitos não foram encontrados por este autor, o que torna esta dissertação uma contribuição para profissionais da indústria e pesquisadores da academia interessados em estimativa de esforço em projetos de desenvolvimento de software crítico.

Nesta dissertação foi idealizado um modelo de estimativa de esforço para projetos de desenvolvimento de software crítico que seguem os processos estabelecidos pela DO-178C. Este modelo foi criado de modo a atender os projetos de software embarcado em sistemas cuja falha pode levar a efeitos desde catastróficos a menores, traduzindo-se em níveis de software A ao D, conforme a norma DO-178C.

O foco principal da dissertação foi desenvolver um guia para a classificação dos fatores multiplicativos de custo do COCOMO II conforme os processos de

desenvolvimento de software crítico apresentados pela DO-178C, considerando os objetivos a serem atingidos para cada um dos níveis de criticidade oferecidos pelo software.

Esta dissertação propõe, como referência, a aplicação do processo de Engenharia de Sistemas apresentado pela ARP-4754A (SAE, 2010) de modo a proporcionar a identificação do nível do software em questão, por meio da análise dos casos de falha em que o software em questão pode contribuir. O modelo proposto por esta dissertação apresenta ainda os conceitos do uso de ferramentas qualificadas no processo de desenvolvimento de software, conforme a DO-330 (RTCA INC, 2011b), influenciando no resultado da estimativa de esforço.

Esta dissertação apresentou um estudo de caso na seção 5 sobre a aplicação do COCOMO II em um projeto de desenvolvimento de software crítico realizado por uma empresa da indústria aeronáutica brasileira, cujo erro relativo entre o esforço estimado e o real foi de 31,3%. Este mesmo projeto foi então utilizado como referência para aplicação do modelo de estimativa de esforço proposto por este trabalho. O modelo proposto apresentou um erro relativo entre o esforço estimado e o real de 10,33%, um erro substancialmente menor que o apresentado pelo modelo COCOMO II. Com relação ao prazo de desenvolvimento, o modelo proposto também apresenta um valor estimado melhor que o apresentado pelo COCOMO II. Porém, com relação ao tamanho médio da equipe, o modelo COCOMO II apresentou um valor estimado mais próximo do real do que o estimado pelo modelo proposto por este trabalho.

Embora com um erro maior na estimativa de tamanho médio da equipe, conclui-se então que o modelo proposto apresentou uma estimativa melhor quando comparado ao COCOMO II para este projeto do estudo de caso, uma vez que os valores estimados de esforço e prazo foram consideravelmente melhores que o modelo COCOMO II original.

Por se tratar de um modelo de estimativa de esforço, as estimativas de prazo e tamanho médio da equipe podem ser consideradas secundárias por alguns leitores, porém são igualmente importantes para fornecer a melhor tomada de

decisão possível para os gestores do projeto. Sendo assim, por meio dos resultados estimados obtidos tanto pelo COCOMO II quanto pelo modelo proposto, fica claro a necessidade de um melhor ajuste na estimativa com relação a esses dois parâmetros.

Como um segundo estudo de caso de aplicação do modelo proposto, esta dissertação adotou o hipotético projeto de missão espacial denominado FireSat, apresentado pelo livro SMAD. Por ser um projeto para fins didático, não existem valores reais para a comparação e cálculo do MRE, no entanto esta dissertação apresentou a comparação entre a estimativa obtida pelo modelo proposto com a estimativa utilizada pelo modelo apresentado pelo livro SMAD. O custo final estimado pelo modelo proposto, dados algumas premissas, foi cerca de 20% superior ao custo estimado pelo SMAD. Não há como se concluir se o valor está melhor ou pior que o estimado originalmente, porém pode-se concluir que o modelo proposto apresentou estimativas coerentes quando se comparado ao modelo sugerido pelo livro SMAD.

Há de se ressaltar, no entanto, que o modelo proposto possui limitações que podem ser exploradas em trabalhos futuros. São elas:

- O modelo proposto é exclusivo para projetos que seguem o processo estabelecido pela DO-178C. Para projeto também *safety-critical,* porém que seguem alguma outra norma, uma adaptação dos fatores multiplicativos de esforço ou uma correlação entre tal norma e a DO-178C se faz necessária;
- Apesar da estimativa de esforço do modelo proposto ter apresentado um MRE menor que a comparada com o COCOMO II para o estudo de caso do projeto aeronáutico, a estimativa do tamanho médio da equipe se mostrou pior, com um número mais distante do real;
- Embora melhor que a estimativa apresentada pelo COCOMO II, o tempo de desenvolvimento estimado pelo modelo proposto ainda ficou distante do que de fato foi necessário, analisando o estudo de caso do projeto do setor aeronáutico;

- Pouco foi explorado nos benefícios ou desvantagens do processo de desenvolvimento baseado em modelos, conhecido como MBD – *Model Based Development;*
- O modelo proposto ainda é fortemente dependente da estimativa inicial do tamanho do software em linhas de código, ao passo que do ponto de vista da DO-178C, uma relação com o número de requisitos seria mais apropriada.
- O modelo proposto foi aplicado apenas em dois projetos, o da Empresa X e o FireSat, o que torna seu resultado sem embasamento estatístico.

Apesar das limitações mencionadas, pode-se concluir finalmente que esta dissertação apresenta importantes contribuições para a indústria e academia, destacando-se as seguintes:

- Estudo de caso de aplicação do COCOMO II em um projeto real da indústria aeronáutica;
- Desenvolvimento de um modelo de estimativa de esforço para projetos de software crítico baseado no consagrado COCOMO II, relacionando os fatores multiplicativos de esforço com os objetivos estabelecidos pela DO-178C;
- Estudo de caso de aplicação do modelo proposto no em um projeto real da indústria aeronáutica;
- Estudo de caso de aplicação do modelo proposto em um projeto didático do setor espacial;

9.1 Publicações

Como resultado da pesquisa que gerou esta dissertação, foram publicados, ou submetidos, os seguintes artigos científicos:

PEREIRA DOS SANTOS, L.; GONCALVES VIEIRA FERREIRA, M. Safety Critical Software Effort Estimation using COCOMO II: A Case Study in Aeronautical Industry. **IEEE Latin America Transactions**, v. 16, n. 7, p. 2069–2078, jul. 2018. ENG III Estrato Qualis B2.

PEREIRA DOS SANTOS, L.; GONCALVES VIEIRA FERREIRA, M. Applying COCOMO II for a DO-178C Safety-Critical Software Effort Estimation. **Journal of Aerospace Technology and Management**. Artigo aceito e com publicação prevista para a edição Janeiro/2019. ENG III Estrato Qualis B3.

PEREIRA DOS SANTOS, L.; GONCALVES VIEIRA FERREIRA, M. A Software Effort Estimation Model based on COCOMO II for DO-178C Projects. **Journal of Systems and Software**. Artigo submetido. ENG III Estrato Qualis A2.

PEREIRA DOS SANTOS, L.; GONCALVES VIEIRA FERREIRA, M. Applying a Software Effort Estimation Model for DO-178C Projects: A Case Study. **IEEE Systems Journal**. Artigo submetido. ENG III Estrato Qualis A2.

Como resultado da pesquisa que gerou esta dissertação, foi apresentado em conferência de engenharia e sistemas o seguinte artigo científico:

PEREIRA DOS SANTOS, L.; GONCALVES VIEIRA FERREIRA, M. **Applying Software Effort Estimation method for a Safety-Critical Software**. 2018 9º Workshop em Engenharia e Tecnologia Espaciais (WETE).

9.2 Trabalhos Futuros

Diante das limitações do modelo proposto apresentadas anteriormente, bem como da quantidade de aspectos que envolvem o desenvolvimento de software crítico e os processos estabelecidos pela DO-178C e seus suplementos, outros trabalhos podem ser elaborados para dar continuidade e aumentar a precisão do modelo proposto. Algumas das sugestões são:

- Adaptar o modelo para tornar possível a estimativa de software críticos que seguem não apenas a DO-178C, mas também normas estabelecidas para outros seguimentos, como a ISO/IEC62304 para o

setor médico e a NASA-STD-8719.13C, com o foco no setor espacial. Embora a DO-178C possa ser utilizada em tais setores, a adaptação do modelo para as normas específicas torna a aplicação mais conveniente para profissionais dessas áreas;

- Aplicar o modelo proposto no banco de dados de projetos utilizado pelo COCOMO II, de modo a obter uma comparação estatística entre eles;
- Ajustar o modelo para uma melhor estimativa do prazo e tamanho médio da equipe;
- Ajustar o modelo para ser menos dependente do tamanho do software em linhas de código (parâmetro SIZE), utilizando, porém, a quantidade de requisitos como métrica;
- Incluir no modelo de estimativa aspectos mais detalhados referentes ao processo de desenvolvimento de software baseado em modelos, cada vez mais comum e utilizado na indústria;

REFERÊNCIAS BIBLIOGRÁFICAS

ABBAS, S. A. et al. Cost estimation: a survey of well-known historic cost estimation techniques. **Journal of Emerging Trends in Computing and Information Sciences**, v. 3, n. 4, p. 612–636, 2012.

BASTEN, D.; MELLIS, W. **A current assessment of software development effort estimation**. In: INTERNATIONAL SYMPOSIUM ON EMPIRICAL SOFTWARE ENGINEERING AND MEASURMENT, 2011. **Proceedings...** IEEE, 2011

BIRD, C. et al. Does distributed development affect software quality? **Communications of the ACM**, v. 52, n. 8, p. 85, 2009.

BOEHM, B. et al. COCOMO suite methodology and evolution. **CrossTalk Magazine**, v. 18, n. 4, p. 20–25, 2005.

BOEHM, B.; ABTS, C.; CHULANI, S. Software development cost estimation approaches: a survey. **Annals of Software Engineering**, v. 10, n. 1/4, p. 177–205, 2000.

BOEHM, B. W. **Software engineering economics**. Upper Saddle River, NJ, EUA: Prentice-Hall, 1981. v. 1

BOEHM, B. W. et al. **Software cost estimation with Cocomo II**. Upper Saddle River, NJ, EUA: Prentice Hall, 2000.

BOEHM, B. W.; VALERDI, R. Achievements and challenges in Cocomo-based software resource estimation. **IEEE Software**, v. 25, n. 5, p. 74–83, 2008.

BRITTO, R. et al. Effort estimation in global software development: a systematic literature review. In: IEEE INTERNATIONAL CONFERENCE ON GLOBAL SOFTWARE ENGINEERING, 9., 2014. **Proceedings...** IEEE, 2014

BRITISH STANDARDS INSTITUTION - BSI. **IEC 62304:2006:** medical device software - software life-cycle processes. Brussels, Belgium: BSI, 2006.

CAPALDO, G. et al. The evaluation of innovation capabilities in small software firms: a methodological approach. **Small Business Economics**, v. 21, n. 4, p. 343–354, 2003.

CHANDRASEKARAN, R.; VENKATESH KUMAR, R. On the estimation of the software effort and schedule using Constructive Cost Model II and Functional Point Analysis. **International Journal of Computer Applications**, v. 44, n. 9, p. 38–44, 2012.

CHESBROUGH, H.; VANHAVERBEKE, W.; WEST, J. (Eds.). **Open innovation:** researching a new paradigm. New York, USA: Oxford University Press, 2006.

DE BARCELOS TRONTO, I. F.; SILVA, J. D. S.; SANT'ANNA, N. An investigation of artificial neural networks based prediction systems in software project management. **Journal of Systems and Software**, v. 81, n. 3, p. 356–367, 2008.

ESTRADA, R. G.; SASAKI, G.; DILLABER, E. Best practices for developing DO-178 compliant software using Model-Based Design. In: AIAA INFOTECH@AEROSPACE CONFERENCE, 2013. **Proceedings**...Reston, Virginia: American Institute of Aeronautics and Astronautics, 2013

FEDERAL AVIATION ADMINISTRATION. **ORDER 8110.49:** software approval guidelines. Washington, DC, EUA, 2003.

FEDERAL AVIATION ADMINISTRATION. **Job air conducting software reviews**. Washington, DC, EUA: FAA, 2004a.

FEDERAL AVIATION ADMINISTRATION. **AC 20-148:** reusable software components. Washington, DC, EUA, 2004b.

FEDERAL AVIATION ADMINISTRATION. **System safety analysis and assessment for part 23 airplanes**. Washington, DC, EUA: FAA, 2011.

FEDERAL AVIATION ADMINISTRATION. **AC 20-115C:** airborne software assurance. Washington, DC, EUA, 2013.

FINNIE, G. R.; WITTIG, G. E.; DESHARNAIS, J.-M. A comparison of software effort estimation techniques: using function points with neural networks, case-based reasoning and regression models. **Journal of Systems and Software**, v. 39, n. 3, p. 281–289, . 1997.

GOYAL, S.; PARASHAR, A. Machine learning application to improve COCOMO Model using neural networks. **International Journal of Information Technology and Computer Science**, v. 10, n. 3, p. 35–51, 2018.

GRIMSTAD, S. **Software effort estimation error**. [S.l.]: University of Oslo, 2006.

GRIMSTAD, S.; JØRGENSEN, M. A framework for the analysis of software cost estimation accuracy. In: ACM/IEEE INTERNATIONAL SYMPOSIUM ON EMPIRICAL SOFTWARE ENGINEERING, 2006. **Proceedings...** New York, USA: ACM Press, 2006.

GRIMSTAD, S.; JØRGENSEN, M. Inconsistency of expert judgment-based estimates of software development effort. **Journal of Systems and Software**, v. 80, n. 11, p. 1770–1777, 2007.

GRIMSTAD, S.; JØRGENSEN, M.; MOLØKKEN-ØSTVOLD, K. Software effort estimation terminology: the tower of Babel. **Information and Software Technology**, v. 48, n. 4, p. 302–310, 2006.

HALKJELSVIK, T.; JØRGENSEN, M. From origami to software development: a review of studies on judgment-based predictions of performance time. **Psychological Bulletin**, v. 138, n. 2, p. 238–271, 2012.

HUGHES, R. T.; CUNLIFFE, A.; YOUNG-MARTOS, F. Evaluating software development effort model-building techniques for application in a real-time telecommunications environment. **IEEE Proceedings Software**, v. 145, n. 1, p. 29, 1998.

IDRI, A.; ABRAN, A.; KJIRI, L. COCOMO cost model using fuzzy logic. In: INTERNATIONAL CONFERENCE ON FUZZY THEORY & TECHNIQUES, 7., 2000. **Proceedings...** Atlantic City, New Jersey, EUA, 2000.

IEEE COMPUTER SOCIETY. **Guide to the Software Engineering Body of Knowledge (SWEBOK guide)**. Los Alamitos, CA, EUA: IEEE Computer Society, 2004.

JØRGENSEN, M. A review of studies on expert estimation of software development effort. **Journal of Systems and Software**, v. 70, n. 1–2, p. 37–60, 2004.

JØRGENSEN, M. Forecasting of software development work effort: evidence on expert judgement and formal models. **International Journal of Forecasting**, v. 23, n. 3, p. 449–462, 2007.

JØRGENSEN, M. Selection of strategies in judgment-based effort estimation. **Journal of Systems and Software**, v. 83, n. 6, p. 1039–1050, 2010.

JØRGENSEN, M. Relative estimation of software development effort: it matters with what and how you compare. **IEEE Software**, v. 30, n. 2, p. 74–79, mar. 2013.

JØRGENSEN, M. What we do and don't know about software development effort estimation. **IEEE Software**, v. 31, n. 2, p. 37–40, 2014a.

JØRGENSEN, M. Communication of software cost estimates. In: INTERNATIONAL CONFERENCE ON EVALUATION AND ASSESSMENT IN SOFTWARE ENGINEERING, 18., 2014. **Proceedings…** New York, USA: ACM Press, 2014b.

JØRGENSEN, M. The effect of the time unit on software development effort estimates. In: INTERNATIONAL CONFERENCE ON SOFTWARE, KNOWLEDGE, INFORMATION MANAGEMENT AND APPLICATIONS (SKIMA), 9., 2015. **Proceedings...** IEEE, 2015.

JØRGENSEN, M. Unit effects in software project effort estimation: work-hours gives lower effort estimates than workdays. **Journal of Systems and Software**, v. 117, p. 274–281, 2016.

JØRGENSEN, M.; INDAHL, U.; SJØBERG, D. Software effort estimation by analogy and "regression toward the mean". **Journal of Systems and Software**, v. 68, n. 3, p. 253–262, 2003.

JØRGENSEN, M.; LØHRE, E. First impressions in software development effort estimation: easy to create and difficult to neutralize. In: INTERNATIONAL CONFERENCE ON EVALUATION & ASSESSMENT IN SOFTWARE ENGINEERING, 16., 2012. **Proceeding…** IET, 2012.

JORGENSEN, M.; SHEPPERD, M. A systematic review of software development cost estimation studies. **IEEE Transactions on Software Engineering**, v. 33, n. 1, p. 33–53, 2007.

JØRGENSEN, M.; SJØBERG, D. I. Impact of effort estimates on software project work. **Information and Software Technology**, v. 43, n. 15, p. 939–948, 2001.

JØRGENSEN, M.; SJOBERG, D. I. K. The importance of NOT learning from experience. In: EUROPEAN SOFTWARE PROCESS IMPROVEMENT, 2000. **Proceedings…** Copenhagen, Denmark, 2000.

KASHYAP, D.; MISRA, A. K. Software development cost estimation using similarity difference between software attributes. In: INTERNATIONAL CONFERENCE ON INFORMATION SYSTEMS AND DESIGN OF COMMUNICATION, 2013. **Proceedings…** New York, USA: ACM Press, 2013.

KAUSHIK, A.; SONI, A. K.; SONI, R. An adaptive learning approach to software cost estimation. In: NATIONAL CONFERENCE ON COMPUTING AND COMMUNICATION SYSTEMS, 2012. **Proceedings…** IEEE, 2012.

KHATIBI, V.; JAWAWI, D. N. A. Software cost estimation methods : a review. **Journal of Emerging Trends in Computing and Information Sciences**, v. 2, n. 1, p. 21–29, 2011.

KUZNECOVA, K. Mrj first delivery expected to be delayed. **50SKYSHADES.COM**, p.1, Jan. 2017.

LEE, S.-Y.; WONG, W. E.; GAO, R. Software safety standards: evolution and lessons learned. In: INTERNATIONAL CONFERENCE ON TRUSTWORTHY SYSTEMS AND THEIR APPLICATIONS, 2014. **Proceedings…** IEEE, 2014.

LUM, K.; MENZIES, T.; BAKER, D. 2CEE: a twenty first century effort estimation methodology. In: ISPA / SCEA JOINT INTERNATIONAL CONFERENCE, 2008. **Proceedings…** Huis ter Duin, Noordwijk, The Netherlands,: ISPA / SCEA, 2008.

MACDONELL, S. G.; SHEPPERD, M. J. Combining techniques to optimize effort predictions in software project management. **Journal of Systems and Software**, v. 66, n. 2, p. 91–98, 2003.

MCCORMICK, J. W.; SINGHOFF, F.; HUGUES, J. 'OME. **Building parallel, embedded, and real-time applications with Ada**. New York, USA: Cambridge University Press, 2011.

MIRANDA, Z. **O Vôo da Embraer:** a competitividade brasileira na indústria de alta tecnologia. São Paulo, SP: Papagaio, 2007.

MOLOKKEN-OSTVOLD, K. et al. A survey on software estimation in the Norwegian industry. In: INTERNATIONAL SYMPOSIUM ON SOFTWARE METRICS, 2004. **Proceedings**... IEEE, 2004.

MOLOKKEN, K.; JORGENSEN, M. A review of software surveys on software effort estimation. In: INTERNATIONAL SYMPOSIUM ON EMPIRICAL SOFTWARE ENGINEERING, 2003. **Proceedings**... IEEE, 2003.

NATIONAL AERONAUTICS AND SPACE ADMINISTRATION. **NASA-GB-8719.13:** software safety guidebook. Washington, D.C., EUA, 2004a.

NATIONAL AERONAUTICS AND SPACE ADMINISTRATION. **NASA-GB-8719.13B:** software safety guidebook. Washington, D.C., EUA, 2004b.

NATIONAL AERONAUTICS AND SPACE ADMINISTRATION. **NASA-STD-8719.13C:** software safety standard. Washington, D.C., EUA, 2013.

NEWHOUSE, J. **Boeing versus Airbus:** the inside story of the greatest international competition in business. New York, USA: Vintage Books, 2008.

NGUYEN, Q. et al. A high performance command and data handling system for NASA's lunar reconnaissance orbiter. In: AIAA SPACE CONFERENCE AND EXPOSITION, 2008. **Proceedings…** .Reston, Virigina: American Institute of Aeronautics and Astronautics, 2008.

PASTOR, I. I. L.; CAULÍN, C. C. T.; MARTÍNEZ, C. F. C. Model-based development and verification for a safety-critical application: from DO-178B/ED-12B to DO-178C/ED-12C compliance. In: ERTS, 2014. **Proceedings…** Madrid, Spain. 2014.

PAZ, A.; EL BOUSSAIDI, G. On the exploration of model-based support for DO-178C-compliant avionics software development and certification. In: IEEE INTERNATIONAL SYMPOSIUM ON SOFTWARE RELIABILITY ENGINEERING WORKSHOPS (ISSREW), 2016. **Proceedings…** IEEE, 2016.

PRESSMAN, R. S. **Software engineering:** a practitioner's approach. 7.ed. New York, USA: McGraw-Hill, 2009.

PRIES, K. H.; QUIGLEY, J. M. **Testing complex and embedded systems**. Boca Raton, EUA: CRC Press, 2011.

PROJECT MANAGEMENT INSTITUTE. **A guide to the project management body of knowledge**. 4.ed. Newtown Square, Pennsylvania, EUA: Project Management Institute, 2008.

PUTNAM, L. H. A general empirical solution to the macro software sizing and estimating problem. **IEEE Transactions on Software Engineering**, v. SE-4, n. 4, p. 345–361, 1978.

RIERSON, L. **Developing safety-critical software:** a practical guide for aviation software and DO-178C compliance. Boca Raton, EUA: CRC Press, 2013.

RIJWANI, P.; JAIN, S.; SANTANI, D. Software effort estimation : a comparison based perspective. **International Journal of Application or Innovation in Engineering & Management**, v. 3, n. 12, p. 18–29, 2014.

RTCA INC. **DO-178:** software considerations in airborne systems and equipment certification. Washington, EUA, 1980.

RTCA INC. **DO-178A:** software considerations in airborne systems and equipment certification. Washington, EUA, 1985.

RTCA INC. **DO-178B:** software considerations in airborne systems and equipment certification. Washington, EUA, 1992.

RTCA INC. **DO-178C:** software considerations in airborne systems and equipment certification. Washington, EUA, 2011a.

RTCA INC. **DO-330:** software tool qualification considerations. Washington, EUA, 2011b.

RTCA INC. **DO-331:** model-based development and verification supplement to DO-178C and DO-278A. Washington, EUA, 2011c.

RTCA INC. **DO-332:** object-oriented technology and related techniques supplement to ed-12c and ed-109a. Washington, EUA, 2011d.

RTCA INC. **DO-333:** formal methods supplement to DO-178C and DO-278A. Washington, EUA, 2011e.

RTCA INC. **DO-248C:** supporting information for DO-178C and DO-278A. Washington, EUA, 2011f.

SAE INTERNATIONAL. **ARP4754A:** guidelines for development of civil aircraft and systems. Warrendale, EUA, 2010.

SALJOUGHINEJAD. R.; KHATIBI, V. A new optimized hybrid model based on COCOMO to increase the accuracy of software cost estimation. **Journal of Advances in Computer Engineering and Technology**, v. 4, n. 1, p. 27–40, 2018.

SARNO, R.; SIDABUTAR, J.; SARWOSRI. Improving the accuracy of COCOMO's effort estimation based on neural networks and fuzzy logic model. In: INTERNATIONAL CONFERENCE ON INFORMATION AND COMMUNICATION TECHNOLOGY AND SYSTEMS (ICTS), 2015. **Proceedings…** IEEE, 2015.

SCARAMUZZA, D. et al. Vision-controlled micro flying robots: from system design to autonomous navigation and mapping in GPS-denied environments. **IEEE Robotics & Automation Magazine**, v. 21, n. 3, p. 26–40, 2014.

SHARMA, N.; BAJPAI, A.; LITORIYA, R. A comparison of software cost estimation methods : a survey. **The International Journal of Computer Science & Applications**, v. 1, n. 3, p. 121–127, 2012.

SHEPPERD, M.; SCHOFIELD, C. Estimating software project effort using analogies. **IEEE Transactions on Software Engineering**, v. 23, n. 11, p. 736–743, 1997.

SHETA, A. F. Estimation of the COCOMO model parameters using genetic algorithms for NASA software projects. **Journal of Computer Science**, v. 2, n. 2, p. 118–123, 2006.

SHETA, A. F.; ALJAHDALI, S. Software effort estimation inspired by COCOMO and FP models: a fuzzy logic approach. **International Journal of Advanced Computer Science and Applications**, v. 4, n. 11, 2013.

SOFTWARE ENGINEERING INSTITUTE. **Software measurement for DoD systems:** recommendations for initial core measures. Pittsburgh, EUA: SEI, 1992.

SOFTWARE ENGINEERING INSTITUTE. **CMMI® for development, version 1.3**. Hanscom, MA, EUA: SEI, 2010.

SOMMERVILLE, I. **Software engineering**. 8.ed. Harlow, England: Pearson Education, 2007.

TERRY BAHILL, A.; HENDERSON, S. J. Requirements development, verification, and validation exhibited in famous failures. **Systems Engineering**, v. 8, n. 1, p. 1–14, 2005.

THE STANDISH GROUP. **The standish group report:** chaos. New York, USA: Project Smart, 2014.

THE STANDISH GROUP. **CHAOS manifesto 2012:** the year of the executive sponsor Chaos manifesto. New York, USA: [S.n.], 2012.

TRONTO, I. B. Uma Investigação de modelos de estimativas de esforço em gerenciamento de projeto de software. In: SIMPÓSIO BRASILEIRO DE ENGENHARIA DE SOFTWARE, 20., 2006. **Anais...** Florianópolis, , Brasil, 2006.

VELARDE, H. et al. Software development effort estimation based-on multiple classifier system and lines of code. **IEEE Latin America Transactions**, v. 14, n. 8, p. 3907–3913, 2016.

WERTZ, J. R.; EVERETT, D. F.; PUSCHELL, J. J. **Space mission engineering:** the new SMAD. Portland, EUA: Microcosm Press, 2011.

WERTZ, J. R.; LARSON, W. **Space mission analysis and design**. 3ed. El Segundo, EUA: Space Technology Library, 1991.

WORLD INTELLECTUAL PROPERY ORGANIZATION. **Who filed the most Pct patent applications in 2015?** Genebra, Suíça: WIPO, 2015.

YAN, F. Comparison of means of compliance for onboard software certification. In: INTERNATIONAL CONFERENCE ON COMPUTER SCIENCE AND EDUCATION, 4., 2009. **Proceedings…** IEEE, 2009.

YELISETTY, S. M. H.; MARQUES, J.; TASINAFFO, P. M. A set of metrics to assess and monitor compliance with RTCA DO-178C. In: DIGITAL AVIONICS SYSTEMS CONFERENCE (DASC), 34., 2015. **Proceedings…** IEEE, 2015.

APÊNDICE A – APLICAÇÃO DOS FATORES MULTIPLICATIVOS DE CUSTO CONFORME MODELO COCOMO II

PARÂMETRO RELY

Este fator mede o efeito de uma falha de software na função em que ele deve executar durante um período de tempo. Para cada efeito da falha um nível é atribuído.

Tabela A.1 – RELY: COCOMO II Escala de Classificação

Fator	Classificação					
	Very Low	Low	Nominal	High	Very High	Extra High
RELY	Inconveniente	Baixo, com perdas facilmente recuperadas	Moderado, com perdas facilmente recuperadas	Alto risco financeiro	Alto risco para vidas humanas	n/a
	0.82	0.92	1.00	1.10	1.26	n/a

Fonte: Adaptado de Boehm et al. (2000).

PARÂMETRO DATA

Este fator relaciona o efeito de testes de grandes quantidades de dados no desenvolvimento do produto. O nível é atribuído com base na taxa de D/P, que significa a taxa de Bytes no banco de dados de teste (D) pela quantidade de linhas de código do software (P). Em outras palavras, DATA está capturando o esforço necessário para montar e manter os dados necessários para concluir o teste do software até o início da operação.

Tabela A.2 – DATA: COCOMO II Escala de Classificação

Fator	Classificação					
	Very Low	Low	Nominal	High	Very High	Extra High
DATA	n/a	D/P < 10	10 ≤ D/P < 100	100 ≤ D/P < 1000	D/P ≥ 1000	n/a
	n/a	0.90	1.00	1.14	1.28	n/a

Fonte: Adaptado de Boehm et al. (2000).

PARÂMETRO CPLX

A complexidade do produto é dividida em cinco áreas pelo modelo COCOMO II: controle de operação, operações computacionais, operações com dispositivos, gerenciamento de dados, operações de interface com usuário. A combinação dessas áreas deve ser utilizada para chegar ao nível do CPLX.

Tabela A.3 – CPLX: COCOMO II Escala de Classificação

CPLX	Área de Complexidade				
	Controle de Operação	Operação computacional	Operação com dispositivo	Gerenciamento de dados	Interface com usuário
Very Low 0.73	Lógicas simples, com apenas chamadas de função e sem laços ou recursão	Expressões simples, como: A = B+C*(D-E);	Simples leitura e escrita em formatos simples	Simples arrays na memória ou bancos de dados COTS	Simples formulários de entrada e geradores de relatórios
Low 0.87	Lógicas simples, com algumas lógicas estruturadas, como laços e IFs	Expressões moderadas, como: D=SQRT(B**2 – 4*A*C)	Dispositivo de I/O pouco conhecido, porém acessado por simples GET/PUT.	Simples escritas em um arquivo por vez, sem edição ou arquivos intermediários	Uso de simples geradores de interface – GUI Builders
Nominal 1.00	Basicamente estruturas simples, com tabelas de decisão e passagem de argumentos	Uso de biblioteca matemática padrão, com operações e vetores ou matrizes	Processamento de I/O com controle de estado e tratamento de erro	Vários arquivos de entrada, mas apenas um de saída.	Interface com simples uso de ferramentas
High 1.17	Operações com alto nível de alinhamento. Controle de pilha e operações de soft real-time	Análises numérica simples: interpolações, equações diferenciais	Operações direta no dispositivo de I/O, com leitura, escrita, transações etc	Simples gatilhos ativados por transmissão de dados	Ferramentas e extensões desenvolvidas, com medias e reconhecimento de voz
Very High 1.34	Código recursivo e com reentrada. Controle de prioridade e interrupções. Sincronização de tarefas e operações de hard real-time, com um único processador	Análises numérica difíceis: equações com matrizes, equações diferenciais parciais	Rotinas para diagnóstico de interrupções, barramento de dados, sistemas embarcados intensivos	Bancos de dados distribuídos, gatilhos complexos e necessidade de otimização de buscas	Complexidade moderada com gráficos 2D/3D e interface multimídia
Extra High 1.74	Múltiplas tarefas escalonadas, com prioridades dinâmicas. Processamento distribuído de hard real-time	Análises numérica complexas: dados estocásticos, analise de ruídos e dados muito precisos	Dispositivo com processamento dedicado. Sistemas embarcados críticos	Altamente acoplado, relacionamento dinâmico.	Interface de linguagem natural, multimídias complexas e realidade virtual

Fonte: Adaptado de Boehm et al. (2000).

PARÂMETRO RUSE

Este fator relaciona o esforço adicional que será necessário para desenvolver software com intenção de ser reutilizado por projetos futuros.

Tabela A.4 – RUSE: COCOMO II Escala de Classificação

Fator	Classificação					
	Very Low	Low	Nominal	High	Very High	Extra High
RUSE	n/a	Nenhum, reuso	Reuso dentro do projeto	Reuso em vários projetos	Reuso em vários produtos	Reuso em várias áreas organizacionais
	n/a	0.95	1.00	1.07	1.15	1.24

Fonte: Adaptado de Boehm et al. (2000).

PARÂMETRO DOCU

Este fator é avaliado em termos da necessidade de documentação ao longo do ciclo de vida do projeto em desenvolvimento.

Tabela A.5 – DOCU: COCOMO II Escala de Classificação

Fator	Classificação					
	Very Low	Low	Nominal	High	Very High	Extra High
DOCU	Pouca necessidade ao longo do ciclo de vida	Alguma necessidade	Necessidade de tamanho ideal para o ciclo de vida	Excessiva necessidade ao longo do ciclo de vida	Muito excessiva necessidade ao longo do ciclo de vida	n/a
	0.81	0.91	1.00	1.11	1.23	n/a

Fonte: Adaptado de Boehm et al. (2000).

PARÂMETRO TIME

Esta é a medida relacionada com a restrição para o tempo de execução do software. A medida é dada em função do percentual do tempo disponível do processador que se espera que seja consumido pelo software em questão.

Tabela A.6 – TIME: COCOMO II Escala de Classificação

Fator	Classificação					
	Very Low	Low	Nominal	High	Very High	Extra High
TIME	n/a	n/a	≤ 50% do tempo de execução disponível	70% do tempo de execução disponível	85% do tempo de execução disponível	95% do tempo de execução disponível
	n/a	n/a	1.00	1.11	1.29	1.63

Fonte: Adaptado de Boehm et al. (2000).

PARÂMETRO STOR

Esta é a medida representa o grau de restrição do principal meio de armazenamento de dados do sistema.

Tabela A.7 – STOR: COCOMO II Escala de Classificação

Fator	Classificação					
	Very Low	Low	Nominal	High	Very High	Extra High
STOR	n/a	n/a	≤ 50% da capacidade disponível	70% da capacidade disponível	85% da capacidade disponível	95% da capacidade disponível
	n/a	n/a	1.00	1.05	1.17	1.46

Fonte: Adaptado de Boehm et al. (2000).

PARÂMETRO PVOL

O termo “plataforma” é utilizado para traduzir a complexidade do hardware e software que o software em questão utiliza para executar suas tarefas, como por exemplo o sistema operacional em que ele está instalado, o banco de dados, software de baixo nível, interface com processador, compiladores etc. A análise é feita sob o ponto de vista da quantidade de mudanças de tal plataforma.

Tabela A.8 – PVOL: COCOMO II Escala de Classificação

Fator	Classificação					
	Very Low	**Low**	**Nominal**	**High**	**Very High**	**Extra High**
PVOL	n/a	Mudanças grandes a cada 12 meses. Pequenas todo mês	Mudanças grandes a cada 6 meses. Pequenas a cada 2 semanas	Mudanças grandes a cada 2 meses. Pequenas toda semana	Mudanças grandes a cada 2 semanas. Pequenas a cada 2 dias	n/a
	n/a	0.87	1.00	1.15	1.30	n/a

Fonte: Adaptado de Boehm et al. (2000).

PARÂMETRO ACAP

Considera-se “analista” neste fator as pessoas que trabalham nos requisitos, arquitetura e detalhamento. Os principais atributos que devem ser considerados na análise deste fator são: habilidade do analista, eficiência e rigor, e habilidade de comunicação e cooperação com os demais membros do time. Esta análise não deve considerar o nível de experiência do analista, pois para isto existem os fatores APEX, LTEX e PLEX. Equipes de analistas que estejam dentro do 15º percentil, devem ser classificadas como “*very low”*.

Tabela A.9 – ACAP: COCOMO II Escala de Classificação

Fator	Classificação					
	Very Low	Low	Nominal	High	Very High	Extra High
ACAP	15º percentil	35º percentil	55º percentil	75º percentil	90º percentil	n/a
	1.42	1.19	1.00	0.85	0.71	n/a

Fonte: Adaptado de Boehm et al. (2000).

PARÂMETRO PCAP

Neste fator a figura do programador deve ser analisada, porém com o ponto de vista no time e não especificamente em indivíduos. Assim como no ACAP, os principais atributos que devem ser considerados na análise deste fator são: habilidade do analista, eficiência e rigor, e habilidade de comunicação e cooperação com os demais membros do time. Novamente, a experiência do programador não deve ser considerada.

Tabela A.10 – PCAP: COCOMO II Escala de Classificação

Fator	Classificação					
	Very Low	Low	Nominal	High	Very High	Extra High
PCAP	15º percentil	35º percentil	55º percentil	75º percentil	90º percentil	n/a
	1.34	1.15	1.00	0.88	0.76	n/a

Fonte: Adaptado de Boehm et al. (2000).

PARÂMETRO PCON

A seleção do nível PCON é feita em termos do turnover anual da equipe do projeto, ou seja, o % de pessoas que deixam o time ao longo do desenvolvimento. Deste modo, objetiva-se avaliar então o nível de continuidade da equipe.

Tabela A.11 – PCON: COCOMO II Escala de Classificação

Fator	Classificação					
	Very Low	Low	Nominal	High	Very High	Extra High
PCON	48% por ano	24% por ano	12% por ano	6% por ano	3% por ano	n/a
	1.29	1.12	1.00	0.90	0.81	n/a

Fonte: Adaptado de Boehm et al. (2000).

PARÂMETRO APEX

Este fator depende do nível de experiência na aplicação que possui o time de desenvolvimento do software em questão, medido em tempo. A análise deve ser feita sob o ponto de vista do time de desenvolvimento e não de uma pessoa específica.

Tabela A.12 – APEX: COCOMO II Escala de Classificação

Fator	Classificação					
	Very Low	Low	Nominal	High	Very High	Extra High
APEX	≤ 2 meses	6 meses	1 ano	3 anos	6 anos	n/a
	1.22	1.10	1.00	0.88	0.81	n/a

Fonte: Adaptado de Boehm et al. (2000).

PARÂMETRO PLEX

O modelo Post-Architecture considera a influencia na produtividade do nível de experiência do time de desenvolvimento na plataforma em questão.

Tabela A.13 – PLEX: COCOMO II Escala de Classificação

Fator	Classificação					
	Very Low	**Low**	**Nominal**	**High**	**Very High**	**Extra High**
PLEX	≤ 2 meses	6 meses	1 ano	3 anos	6 anos	n/a
	1.19	1.09	1.00	0.91	0.85	n/a

Fonte: Adaptado de Boehm et al. (2000).

PARÂMETRO LTEX

Este fator, por sua vez, considera o nível de experiência do time de desenvolvimento na linguagem de programação adotada e nas ferramentas de software utilizadas. Os fatores considerados aqui devem ser: ferramentas de criação de diagramas para representação da arquitetura, ferramentas de controle de configuração, gerenciamento de bibliotecas, padrão de código etc.

Tabela A.14 – LTEX: COCOMO II Escala de Classificação

Fator	Classificação					
	Very Low	Low	Nominal	High	Very High	Extra High
LTEX	≤ 2 meses	6 meses	1 ano	3 anos	6 anos	n/a
	1.20	1.09	1.00	0.91	0.84	n/a

Fonte: Adaptado de Boehm et al. (2000).

PARÂMETRO TOOL

Este fator considera o uso de ferramentas sofisticadas no desenvolvimento do software. Tais ferramentas podem simplificar e contribuir na redução do esforço necessário. O nível “very low” considera o uso apenas de simples ferramentas para edição e escrita do código.

Tabela A.15 – TOOL: COCOMO II Escala de Classificação

Fator	Classificação					
	Very Low	Low	Nominal	High	Very High	Extra High
TOOL	Escrita, edição e debug do código	Ferramentas simples de frontend, backend CASE com pouca integração	Ferramentas básicas no ciclo de vida com algum nível de integração	Ferramentas maduras e com muita integração	Ferramentas maduras, altamente integradas com os processos e métodos	n/a
	1.17	1.09	1.00	0.90	0.78	n/a

Fonte: Adaptado de Boehm et al. (2000).

PARÂMETRO SITE

Este fator deve ser analisado de duas formas: (i) co-localização do time, desde totalmente centralizado até internacionalmente distribuído, e (ii) suporte de comunicação, desde simples e-mail e telefone até ferramentas de interação multimídia.

Tabela A.16 – SITE: COCOMO II Escala de Classificação

SITE	**Classificação**					
	Very Low	Low	Nominal	High	Very High	Extra High
Localização	Internacional	Várias cidades E várias empresas	Várias cidades OU várias empresas	Mesma cidade ou região	Mesmo prédio ou complexo	Totalmente centralizada
Comunicação	Telefone, e-mail	Ramal dedicado	Telefone, e-mail mensagem instantânea	Comunicação totalmente eletrônica	Comunicação totalmente eletrônica e vídeo conferência	Media interativa
SITE Level	1.22	1.09	1.00	0.93	0.86	0.80

Fonte: Adaptado de Boehm et al. (2000).

PARÂMETRO SCED

Este fator considera as restrições de cronograma impostas sob o time de desenvolvimento de software. A avaliação deve ser feita sob o ponto de vista do percentual do prazo estipulado para o projeto quando comparado ao prazo nominal para um projeto que exige um determinado esforço. Cronogramas apertados exigem maior esforço, ao passo que cronogramas mais relaxados não aliviam a quantidade de esforço, uma vez que o ganho de times menores geralmente é contrabalanceado por necessidade de suporte e gerenciamento por um período de tempo mais longo.

Tabela A.17 – SCED: COCOMO II Escala de Classificação

Fator	**Classificação**					
	Very Low	Low	Nominal	High	Very High	Extra High
SCED	75% do nominal	85% do nominal	100% do nominal	130% do nominal	160% do nominal	n/a
	1.43	1.14	1.00	1.00	1.00	n/a

Fonte: Adaptado de Boehm et al. (2000).

APÊNDICE B – FONTE DE DADOS DA Figura 5.3

Tabela A.18 – FSP: Fonte de Dados da Figura 5.3

FSP	t
18,63656678	0
21,02723184	1
23,30476661	2
25,45828843	3
27,47801266	4
29,35532395	5
31,0828332	6
32,65441985	7
34,06525919	8
35,31183466	9
36,3919354	10
37,30463951	11
38,05028354	12
38,63041905	13
39,04775732	14
39,30610315	15
39,41027912	16
39,36604153	17
39,17998956	18
38,85946895	19
38,41247172	20
37,8475334	21
37,17362921	22
36,40007048	23
35,5364027	24
34,59230641	25
33,57750193	26
32,50165907	27
31,37431263	28
30,2047843	29
29,00211179	30
27,77498544	31
26,5316927	32
25,28007069	33
24,02746689	34
22,78070779	35

Fonte: Produção do autor.

APÊNDICE C – FONTE DE DADOS DA Figura 8.1

Tabela A.19 – FSP: Fonte Dados da Figura 8.1

FSP	t
25,80764858	0
28,67093252	1
31,41936437	2
34,04309029	3
36,53308023	4
38,88117814	5
41,08014415	6
43,12368855	7
45,00649729	8
46,72424898	9
48,27362329	10
49,65230106	11
50,858956	12
51,89323861	13
52,7557524	14
53,44802294	15
53,97246028	16
54,33231524	17
54,53163018	18
54,57518494	19
54,46843854	20
54,21746747	21
53,82890113	22
53,30985533	23
52,66786434	24
51,91081236	25
51,04686507	26
50,08440175	27
49,03194886	28
47,89811535	29
46,69153044	30
45,4207843	31
44,09437193	32
42,72064081	33
41,30774237	34

Continua

Tabela A.19 – Conclusão

39,86358784	**35**
38,39580831	36
36,91171947	37
35,41829084	38
33,92211966	39
32,42940939	40

Fonte: Produção do autor.

APÊNDICE D – FONTE DE DADOS DA Figura 8.6

Tabela A.20 – FSP: Fonte Dados da Figura 8.6

FSP	t
6,67627993	0
8,129532503	1
9,453649743	2
10,63081897	3
11,64702003	4
12,49227184	5
13,1607379	6
13,6506926	7
13,96435591	8
14,10760963	9
14,08961243	10
13,92233452	11
13,6200346	12
13,19870299	13
12,67549401	14
12,06816982	15
11,39457506	16
10,67215898	17
9,917557923	18
9,146247294	19
8,372268304	20

Fonte: Produção do autor.

ANEXO A – DO-178C TABELA A-1

Table A-1 Software Planning Process

	Objective		Activity	Applicability by Software Level				Output		Control Category by Software Level			
	Description	Ref	Ref	A	B	C	D	Data Item	Ref	A	B	C	D
1	The activities of the software life cycle processes are defined.	4.1.a	4.2.a 4.2.c 4.2.d 4.2.e 4.2.g 4.2.i 4.2.l 4.3.c	O	O	O	O	PSAC SDP SVP SCM Plan SQA Plan	11.1 11.2 11.3 11.4 11.5	① ① ② ② ②	① ① ② ② ②	① ② ② ② ②	① ② ② ② ②
2	The software life cycle(s), including the inter-relationships between the processes, their sequencing, feedback mechanisms, and transition criteria, is defined.	4.1.b	4.2i 4.3.b	O	O	O		PSAC SDP SVP SCM Plan SQA Plan	11.1 11.2 11.3 11.4 11.5	① ① ① ① ①	① ① ① ① ①	① ② ② ② ②	
3	Software life cycle environment is selected and defined.	4.1.c	4.4.1 4.4.2.a 4.4.2.b 4.4.2.c 4.4.3	O	O	O		PSAC SDP SVP SCM Plan SQA Plan	11.1 11.2 11.3 11.4 11.5	① ① ① ① ①	① ① ① ① ①	① ② ② ② ②	
4	Additional considerations are addressed.	4.1.d	4.2.f 4.2.h 4.2.i 4.2.j 4.2.k	O	O	O	O	PSAC SDP SVP SCM Plan SQA Plan	11.1 11.2 11.3 11.4 11.5	① ① ① ① ①	① ① ① ① ①	① ② ② ② ②	① ② ② ② ②
5	Software development standards are defined.	4.1.e	4.2.b 4.2.g 4.5	O	O	O		SW Requirements Standards SW Design Standards SW Code Standards	11.6 11.7 11.8	① ① ①	① ① ①	② ② ②	
6	Software plans comply with this document.	4.1.f	4.3.a 4.6	O	O	O		Software Verification Results	11.14	②	②	②	
7	Development and revision of software plans are coordinated.	4.1.g	4.2.g 4.6	O	O	O		Software Verification Results	11.14	②	②	②	

Fonte: RTCA (2011a).

ANEXO B – DO-178C TABELA A-2

Table A-2 Software Development Processes

	Objective		Activity	Applicability by Software Level				Output		Control Category by Software Level			
	Description	Ref	Ref	A	B	C	D	Data Item	Ref	A	B	C	D
1	High-level requirements are developed.	5.1.1.a	5.1.2.a 5.1.2.b 5.1.2.c 5.1.2.d 5.1.2.e 5.1.2.f 5.1.2.g 5.1.2.j 5.5.a	O	O	O	O	Software Requirements Data Trace Data	11.9 11.21	① ①	① ①	① ①	① ①
2	Derived high-level requirements are defined and provided to the system processes, including the system safety assessment process.	5.1.1.b	5.1.2.h 5.1.2.i	O	O	O	O	Software Requirements Data	11.9	①	①	①	①
3	Software architecture is developed.	5.2.1.a	5.2.2.a 5.2.2.d	O	O	O	O	Design Description	11.10	①	①	①	②
4	Low-level requirements are developed.	5.2.1.a	5.2.2.a 5.2.2.e 5.2.2.f 5.2.2.g 5.2.3.a 5.2.3.b 5.2.4.a 5.2.4.b 5.2.4.c 5.5.b	O	O	O		Design Description Trace Data	11.10 11.21	① ①	① ①	① ①	
5	Derived low-level requirements are defined and provided to the system processes, including the system safety assessment process.	5.2.1.b	5.2.2.b 5.2.2.c	O	O	O		Design Description	11.10	①	①	①	
6	Source Code is developed.	5.3.1.a	5.3.2.a 5.3.2.b 5.3.2.c 5.3.2.d 5.5.c	O	O	O		Source Code Trace Data	11.11 11.21	① ①	① ①	① ①	
7	Executable Object Code and Parameter Data Item Files, if any, are produced and loaded in the target computer.	5.4.1.a	5.4.2.a 5.4.2.b 5.4.2.c 5.4.2.d 5.4.2.e 5.4.2.f	O	O	O	O	Executable Object Code Parameter Data Item File	11.12 11.22	① ①	① ①	① ①	① ①

Fonte: RTCA (2011a).

ANEXO C – DO-178C TABELA A-3

Table A-3 Verification of Outputs of Software Requirements Process

	Objective		Activity	Applicability by Software Level				Output		Control Category by Software Level			
	Description	Ref	Ref	A	B	C	D	Data Item	Ref	A	B	C	D
1	High-level requirements comply with system requirements.	6.3.1.a	6.3.1	●	●	○	○	Software Verification Results	11.14	②	②	②	②
2	High-level requirements are accurate and consistent.	6.3.1.b	6.3.1	●	●	○	○	Software Verification Results	11.14	②	②	②	②
3	High-level requirements are compatible with target computer.	6.3.1.c	6.3.1	○	○			Software Verification Results	11.14	②	②		
4	High-level requirements are verifiable.	6.3.1.d	6.3.1	○	○	○		Software Verification Results	11.14	②	②	②	
5	High-level requirements conform to standards.	6.3.1.e	6.3.1	○	○	○		Software Verification Results	11.14	②	②	②	
6	High-level requirements are traceable to system requirements.	6.3.1.f	6.3.1	○	○	○	○	Software Verification Results	11.14	②	②	②	②
7	Algorithms are accurate.	6.3.1.g	6.3.1	●	●	○		Software Verification Results	11.14	②	②	②	

Fonte: RTCA (2011a).

ANEXO D – DO-178C TABELA A-4

Table A-4 Verification of Outputs of Software Design Process

	Objective		Activity	Applicability by Software Level				Output		Control Category by Software Level			
	Description	Ref	Ref	A	B	C	D	Data Item	Ref	A	B	C	D
1	Low-level requirements comply with high-level requirements.	6.3.2.a	6.3.2	●	●	○		Software Verification Results	11.14	②	②	②	
2	Low-level requirements are accurate and consistent.	6.3.2.b	6.3.2	●	●	○		Software Verification Results	11.14	②	②	②	
3	Low-level requirements are compatible with target computer.	6.3.2.c	6.3.2	○	○			Software Verification Results	11.14	②	②		
4	Low-level requirements are verifiable.	6.3.2.d	6.3.2	○	○			Software Verification Results	11.14	②	②		
5	Low-level requirements conform to standards.	6.3.2.e	6.3.2	○	○	○		Software Verification Results	11.14	②	②	②	
6	Low-level requirements are traceable to high-level requirements.	6.3.2.f	6.3.2	○	○	○		Software Verification Results	11.14	②	②	②	
7	Algorithms are accurate.	6.3.2.g	6.3.2	●	●	○		Software Verification Results	11.14	②	②	②	
8	Software architecture is compatible with high-level requirements.	6.3.3.a	6.3.3	●	○	○		Software Verification Results	11.14	②	②	②	
9	Software architecture is consistent.	6.3.3.b	6.3.3	●	○	○		Software Verification Results	11.14	②	②	②	
10	Software architecture is compatible with target computer.	6.3.3.c	6.3.3	○	○			Software Verification Results	11.14	②	②		
11	Software architecture is verifiable.	6.3.3.d	6.3.3	○	○			Software Verification Results	11.14	②	②		
12	Software architecture conforms to standards.	6.3.3.e	6.3.3	○	○	○		Software Verification Results	11.14	②	②	②	
13	Software partitioning integrity is confirmed.	6.3.3.f	6.3.3	●	○	○	○	Software Verification Results	11.14	②	②	②	②

Fonte: RTCA (2011a).

ANEXO E – DO-178C TABELA A-5

Table A-5 Verification of Outputs of Software Coding & Integration Processes

	Objective		Activity	Applicability by Software Level				Output		Control Category by Software Level			
	Description	Ref	Ref	A	B	C	D	Data Item	Ref	A	B	C	D
1	Source Code complies with low-level requirements.	6.3.4.a	6.3.4	●	●	○		Software Verification Results	11.14	②	②	②	
2	Source Code complies with software architecture.	6.3.4.b	6.3.4	●	○	○		Software Verification Results	11.14	②	②	②	
3	Source Code is verifiable.	6.3.4.c	6.3.4	○	○			Software Verification Results	11.14	②	②		
4	Source Code conforms to standards.	6.3.4.d	6.3.4	○	○	○		Software Verification Results	11.14	②	②	②	
5	Source Code is traceable to low-level requirements.	6.3.4.e	6.3.4	○	○	○		Software Verification Results	11.14	②	②	②	
6	Source Code is accurate and consistent.	6.3.4.f	6.3.4	●	○	○		Software Verification Results	11.14	②	②	②	
7	Output of software integration process is complete and correct.	6.3.5.a	6.3.5	○	○	○		Software Verification Results	11.14	②	②	②	
8	Parameter Data Item File is correct and complete	6.6.a	6.6	●	●	○	○	Software Verification Cases and Procedures Software Verification Results	11.13 11.14	① ②	① ②	② ②	② ②
9	Verification of Parameter Data Item File is achieved.	6.6.b	6.6	●	●	○		Software Verification Results	11.14	②	②	②	

Fonte: RTCA (2011a).

ANEXO F – DO-178C TABELA A-6

Table A-6 Testing of Outputs of Integration Process

	Objective		Activity	Applicability by Software Level				Output		Control Category by Software Level			
	Description	Ref	Ref	A	B	C	D	Data Item	Ref	A	B	C	D
1	Executable Object Code complies with high-level requirements.	6.4.a	6.4.2 6.4.2.1 6.4.3 6.5	○	○	○	○	Software Verification Cases and Procedures Software Verification Results Trace Data	11.13 11.14 11.21	① ② ①	① ② ①	② ② ②	② ② ②
2	Executable Object Code is robust with high-level requirements.	6.4.b	6.4.2 6.4.2.2 6.4.3 6.5	○	○	○	○	Software Verification Cases and Procedures Software Verification Results Trace Data	11.13 11.14 11.21	① ② ①	① ② ①	② ② ②	② ② ②
3	Executable Object Code complies with low-level requirements.	6.4.c	6.4.2 6.4.2.1 6.4.3 6.5	●	●	○		Software Verification Cases and Procedures Software Verification Results Trace Data	11.13 11.14 11.21	① ② ①	① ② ①	② ② ②	
4	Executable Object Code is robust with low-level requirements.	6.4.d	6.4.2 6.4.2.2 6.4.3 6.5	●	○	○		Software Verification Cases and Procedures Software Verification Results Trace Data	11.13 11.14 11.21	① ② ①	① ② ①	② ② ②	
5	Executable Object Code is compatible with target computer.	6.4.e	6.4.1.a 6.4.3.a	○	○	○	○	Software Verification Cases and Procedures Software Verification Results	11.13 11.14	① ②	① ②	② ②	② ②

Fonte: RTCA (2011a).

ANEXO G – DO-178C TABELA A-7

Table A-7 Verification of Verification Process Results

	Objective		Activity	Applicability by Software Level				Output		Control Category by Software Level			
	Description	Ref	Ref	A	B	C	D	Data Item	Ref	A	B	C	D
1	Test procedures are correct.	6.4.5.b	6.4.5	●	○	○		Software Verification Results	11.14	②	②	②	
2	Test results are correct and discrepancies explained.	6.4.5.c	6.4.5	●	○	○		Software Verification Results	11.14	②	②	②	
3	Test coverage of high-level requirements is achieved.	6.4.4.a	6.4.4.1	●	○	○	○	Software Verification Results	11.14	②	②	②	②
4	Test coverage of low-level requirements is achieved.	6.4.4.b	6.4.4.1	●	○	○		Software Verification Results	11.14	②	②	②	
5	Test coverage of software structure (modified condition/decision coverage) is achieved.	6.4.4.c	6.4.4.2.a 6.4.4.2.b 6.4.4.2.d 6.4.4.3	●				Software Verification Results	11.14	②			
6	Test coverage of software structure (decision coverage) is achieved.	6.4.4.c	6.4.4.2.a 6.4.4.2.b 6.4.4.2.d 6.4.4.3	●	●			Software Verification Results	11.14	②	②		
7	Test coverage of software structure (statement coverage) is achieved.	6.4.4.c	6.4.4.2.a 6.4.4.2.b 6.4.4.2.d 6.4.4.3	●	●	○		Software Verification Results	11.14	②	②	②	
8	Test coverage of software structure (data coupling and control coupling) is achieved.	6.4.4.d	6.4.4.2.c 6.4.4.2.d 6.4.4.3	●	●	○		Software Verification Results	11.14	②	②	②	
9	Verification of additional code, that cannot be traced to Source Code, is achieved.	6.4.4.c	6.4.4.2.b	●				Software Verification Results	11.14	②			

Fonte: RTCA (2011a).

ANEXO H – DO-178C TABELA A-8

Table A-8 Software Configuration Management Process

	Objective		Activity	Applicability by Software Level				Output		Control Category by Software Level			
	Description	Ref	Ref	A	B	C	D	Data Item	Ref	A	B	C	D
1	Configuration items are identified.	7.1.a	7.2.1	O	O	O	O	SCM Records	11.18	②	②	②	②
2	Baselines and traceability are established.	7.1.b	7.2.2	O	O	O	O	Software Configuration Index SCM Records	11.16 11.18	① ②	① ②	① ②	① ②
3	Problem reporting, change control, change review, and configuration status accounting are established.	7.1.c 7.1.d 7.1.e 7.1.f	7.2.3 7.2.4 7.2.5 7.2.6	O	O	O	O	Problem Reports SCM Records	11.17 11.18	② ②	② ②	② ②	② ②
4	Archive, retrieval, and release are established.	7.1.g	7.2.7	O	O	O	O	SCM Records	11.18	②	②	②	②
5	Software load control is established.	7.1.h	7.4	O	O	O	O	SCM Records	11.18	②	②	②	②
6	Software life cycle environment control is established.	7.1.i	7.5	O	O	O	O	Software Life Cycle Environment Configuration Index SCM Records	11.15 11.18	① ②	① ②	① ②	② ②

Fonte: RTCA (2011a).

ANEXO I – DO-178C TABELA A-9

Table A-9 Software Quality Assurance Process

	Objective		Activity	Applicability by Software Level				Output		Control Category by Software Level			
	Description	Ref	Ref	A	B	C	D	Data Item	Ref	A	B	C	D
1	Assurance is obtained that software plans and standards are developed and reviewed for compliance with this document and for consistency.	8.1.a	8.2.b 8.2.h 8.2.i	●	●	●		SQA Records	11.19	②	②	②	
2	Assurance is obtained that software life cycle processes comply with approved software plans.	8.1.b	8.2.a 8.2.c 8.2.d 8.2.f 8.2.h 8.2.i	●	●	●	●	SQA Records	11.19	②	②	②	②
3	Assurance is obtained that software life cycle processes comply with approved software standards.	8.1.b	8.2.a 8.2.c 8.2.d 8.2.f 8.2.h 8.2.i	●	●	●		SQA Records	11.19	②	②	②	
4	Assurance is obtained that transition criteria for the software life cycle processes are satisfied.	8.1.c	8.2.e 8.2.h 8.2.i	●	●	●		SQA Records	11.19	②	②	②	
5	Assurance is obtained that software conformity review is conducted.	8.1.d	8.2.g 8.2.h 8.3	●	●	●	●	SQA Records	11.19	②	②	②	②

Fonte: RTCA (2011a).

ANEXO J – DO-178C TABELA A-10

Table A-10 Certification Liaison Process

	Objective		Activity	Applicability by Software Level				Output		Control Category by Software Level			
	Description	Ref	Ref	A	B	C	D	Data Item	Ref	A	B	C	D
1	Communication and understanding between the applicant and the certification authority is established.	9.a	9.1.b 9.1.c	O	O	O	O	Plan for Software Aspects of Certification	11.1	①	①	①	①
2	The means of compliance is proposed and agreement with the Plan for Software Aspects of Certification is obtained.	9.b	9.1.a 9.1.b 9.1.c	O	O	O	O	Plan for Software Aspects of Certification	11.1	①	①	①	①
3	Compliance substantiation is provided.	9.c	9.2.a 9.2.b 9.2.c	O	O	O	O	Software Accomplishment Summary Software Configuration Index	11.20 11.16	① ①	① ①	① ①	① ①

Fonte: RTCA (2011a).

ANEXO K – COMPUTADORES DO SEGMENTO ESPACIAL DISPONÍVEIS PARA O PROJETO FIRESAT

TABLE 16-17. Commercially Available Space Computers. These computers have been developed for use in a variety of general purpose space applications.

Supplier and Computer	ISA	Word Length (bits)	Memory (RAM + EEPROM)	Performance (MIPS)	Radiation Hardness	Connectivity	Heritage
Honeywell GVSC	1750A	16	16 MB*	1 to 3	1 MRad	1553B RS-232 RS-422	NEAR, ChinaStar, Clementine
Honeywell RH32	R 3000	32	4 GB	10 to 20	1 MRad	1553B RS-232 RS-422	GPS II SBIRS High
Honeywell RHPPC	603E	32	4 GB	20	100 KRad	RS-232 RS-422	VIIRS
L-M GVSC	1750A	16	16 MB*	1 to 2	1 MRad	1553B RS-232 IEEE-488	Cassini, Rapid I
L-M RAD 3000	R 3000	32	16 MB	10	Rad Hard	1553B RS-422	LM-900
L-M RAD 6000	RS 6000	32	16 GB	10 to 20	100 KRad	PCI Firewire HSS	Mars Pathfinder, Globalstar, Space Station, SBIRS Low, Mars 98
TRW	RS-3000	32	16 MB	10	Rad Hard	1553B RS-422	SSTI, T200b, Step-E
SWRI SC-2A	80C186	16	768 KB	0.3	10 KRad	Parallel RS-422	MSTI-2
SWRI SC-5	80C386	32	320 KB	0.6	10 KRad	Parallel RS-422	RADARSAT, SNOE
SWRI SC-7	TI320C30	32	640 KW	12	100 KRad	1553B	MSTI-3
SWRI SC-1750A	1750A	16	512 KB	1	10 KRad	RS-422 1553B	MSTI-1,2,3 New Millennium DS-1
SWRI SC-9	RS 6000	32	128 MB	20	30 KRad	RS-232	Space Station
SWRI MOPS	R 6000	32	128 MB	25	30 KRad	RS-422	Gravity Probe B, International Space Station Alpha (ISSA)
Sanders STAR-RH	R 3000	32	4 MB	10	50 KRad	1553B	CRSS
GDAIS ISE	603E	32	2 GB	25	Rad Hard	1553B	HEAO, AFAX
Acer Sertek	80186	16	512 KB	0.5	Rad Hard	1553B RS-422	ROCSAT

* Address Space L-M: Lockheed Martin SWRI: Southwest Research Institute

Fonte: Wertz et al. (2011, p. 669).

PUBLICAÇÕES TÉCNICO-CIENTÍFICAS EDITADAS PELO INPE

Teses e Dissertações (TDI)

Teses e Dissertações apresentadas nos Cursos de Pós-Graduação do INPE.

Manuais Técnicos (MAN)

São publicações de caráter técnico que incluem normas, procedimentos, instruções e orientações.

Notas Técnico-Científicas (NTC)

Incluem resultados preliminares de pesquisa, descrição de equipamentos, descrição e ou documentação de programa de computador, descrição de sistemas e experimentos, apresentação de testes, dados, atlas, e documentação de projetos de engenharia.

Relatórios de Pesquisa (RPQ)

Reportam resultados ou progressos de pesquisas tanto de natureza técnica quanto científica, cujo nível seja compatível com o de uma publicação em periódico nacional ou internacional.

Propostas e Relatórios de Projetos (PRP)

São propostas de projetos técnico-científicos e relatórios de acompanhamento de projetos, atividades e convênios.

Publicações Didáticas (PUD)

Incluem apostilas, notas de aula e manuais didáticos.

Publicações Seriadas

São os seriados técnico-científicos: boletins, periódicos, anuários e anais de eventos (simpósios e congressos). Constam destas publicações o Internacional Standard Serial Number (ISSN), que é um código único e definitivo para identificação de títulos de seriados.

Programas de Computador (PDC)

São as sequências de instruções ou códigos, expressos em uma linguagem de programação compilada ou interpretada, a ser executada por um computador para alcançar um determinado objetivo. São aceitos tanto programas fonte quanto executáveis.

Pré-publicações (PRE)

Todos os artigos publicados em periódicos, anais e como capítulos de livros.