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

sid.inpe.br/mtc-m21c/2018/03.20.03.57-TDI

TÁTICAS DE REUSO DE SOFTWARE APLICÁVEIS A SISTEMAS ESPACIAIS DE BORDO

Demetryus Vitale Junqueira

Dissertação de Mestrado do Curso de Pós-Graduação em Engenharia e Tecnologia Espaciais/Engenharia e Gerenciamento de Sistemas Espaciais, orientada pelo Dr. Walter Abrahão dos Santos, aprovada em 21 de março de 2018.

URL do documento original:
<http://urlib.net/8JMKD3MGP3W34R/3QP3B4L>

INPE
São José dos Campos
2018

PUBLICADO POR:

Instituto Nacional de Pesquisas Espaciais - INPE
Gabinete do Diretor (GBDIR)
Serviço de Informação e Documentação (SESID)
Caixa Postal 515 - CEP 12.245-970
São José dos Campos - SP - Brasil
Tel.:(012) 3208-6923/6921
E-mail: pubtc@inpe.br

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

sid.inpe.br/mtc-m21c/2018/03.20.03.57-TDI

TÁTICAS DE REUSO DE SOFTWARE APLICÁVEIS A SISTEMAS ESPACIAIS DE BORDO

Demetryus Vitale Junqueira

Dissertação de Mestrado do Curso de Pós-Graduação em Engenharia e Tecnologia Espaciais/Engenharia e Gerenciamento de Sistemas Espaciais, orientada pelo Dr. Walter Abrahão dos Santos, aprovada em 21 de março de 2018.

URL do documento original:
<http://urlib.net/8JMKD3MGP3W34R/3QP3B4L>

INPE
São José dos Campos
2018

Dados Internacionais de Catalogação na Publicação (CIP)

Junqueira, Demetryus Vitale.

J968t Táticas de reuso de software aplicáveis a sistemas espaciais de bordo / Demetryus Vitale Junqueira. – São José dos Campos : INPE, 2018.

xxii + 133 p. ; (sid.inpe.br/mtc-m21c/2018/03.20.03.57-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. Walter Abrahão dos Santos.

1. Reusabilidade de software. 2. Software de bordo. 3. Nanosatélites. I.Título.

CDU 004.053:629.78

Aluno (a): *Demetryus Vitale Junqueira*
Título: "TÁTICAS DE REUSO DE SOFTWARE APLICÁVEIS A SISTEMAS ESPACIAIS DE BORDO".

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

Dra. Maria de Fátima Mattiello-Francisco

Presidente / INPE / São José dos Campos - SP

( ) Participação por Video - Conferência

Dr. Walter Abrahão dos Santos

Orientador(a) / INPE / São José dos Campos - SP

( ) Participação por Video - Conferência

Dr. Fabricio de Novaes Kucinskis

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

( ) Participação por Video - Conferência

Dr. Adilson Marques da Cunha

Convidado(a) / ITA / SJCampos - SP

( ) Participação por Video - Conferência

Este trabalho foi aprovado por:

(X) maioria simples

( ) unanimidade

São José dos Campos, 21 de março de 2018

AGRADECIMENTOS

Agradeço a DEUS e Familiares pela saúde e o apoio, em especial à minha esposa, Andréia Patrícia de Miranda Junqueira, que me acompanha desde o inicio.

Agradeço aos Professores e Instituições de Ensino pelas quais passei e me formei, em especial ao meu Orientador, Dr. Walter Abrahão dos Santos e ao INPE.

RESUMO

O processo de desenvolvimento de soluções de engenharia, é algo vivo, e está sempre em evolução. Problemas de recorrência e *time-to-market* por exemplo, podem afetar a produtividade dos engenheiros e merecem atenção. Uma técnica já usada em agências espaciais como a *European Space Agency* (ESA) e a *National Aeronautics and Space Administration* (NASA), além do *Department of Defense* (DoD) americano é o reuso do software, reconhecida por todos como uma boa prática. Especialistas em software embarcado, consideram que existe uma porcentagem relevante de semelhanças nos aplicativos embarcados com sistemas já desenvolvidos. Logo, com intuito de alavancar a produtividade, sem abrir mão da confiabilidade no desenvolvimento de soluções de engenharia, e colaborar com a redução dos custos, devido à redução da engenharia recorrente e da manutenção propõe-se, táticas para promover a reusabilidade de firmware e middleware, para os projetos de bordo de nanosatélites que poderão ser aplicados nos portfólios de produtos de instituições e companhias. A pesquisa traz como fundamentos para as táticas, o estado da arte das abordagens, técnicas e processos conhecidos, que alavancam o reuso de software. As agência espaciais possuem muitas lições aprendidas com os problemas que enfrentaram durante um longo período de experiência, e reconhecem tais fundamentos como melhores práticas, devido às falhas superadas no processo de implantação da reutilização. Esta pesquisa pretende, como forma de exemplificação à aplicação de tais táticas, apresentar um Estudo de Caso, que promova o reuso de uma funcionalidade da camada de *drivers* em dois nanosatélites, com a flexibilidade para ser empregado em outros, respeitando as condições deste Estudo. Quando se fala em reuso deve-se atentar aos seus custos, por isso serão apresentados, indicadores econômicos dentro de um modelo de análise de viabilidade, que poderão ser aplicados antes da decisão quanto ao desenvolvimento dos produtos de software reutilizáveis. Tais modelos serão aplicados em um Estudo de Caso, como forma de exemplificação de seu uso.

Palavras-chave: Reusabilidade de Software. Software de bordo. Nanosatélites. Sistemas de Software. Sistemas Embarcados. Reuso em Firmware. Reuso em Middleware.

SOFTWARE REUSE TACTICS APPLIED TO ON-BOARD SPACE SYSTEMS

ABSTRACT

The development process of engineering is alive and always evolving. Recurrence and time-to-market issues, for example, can affect the productivity of engineers and deserve attention. One technique already used in space agencies such as the European Space Agency (ESA) and the National Aeronautics and Space Administration (NASA), in addition to the US Department of Defense (DoD), is the Software Reuse and they recognize it as a good practice. Embedded Software experts considers that there is a relevant percentage of similarities in embedded applications with systems already developed. Therefore, in order to leverage productivity without relinquishing reliability in the development of engineering solutions and collaborate with cost reduction due to the reduction of recurrent engineering and maintenance, it is intended to present tactics to promote firmware and middleware reusability to nanosatellites on-board software projects, where It can be applied for Institutions and Companies products portfolios. The research provides as fundamentals for tactics, the state-of-the-art of known approaches, techniques and processes that leverage software reuse. The agencies has many lessons learned from the problems that faced over a long period of experience and recognize these fundamentals as best practices to failures overcome in the reuse deployment process.This research intends, as a way to exemplify the application of such tactics, to present a Case study, which promotes a driver layer functionality reuse into two nanosatellites, with the flexibility to be employed in others, respecting the conditions of this study. When it talks about reuse one must pay attention to its costs, so economic indicators will be presented, within feasibility analysis models, that can be applied before the decision, regarding the development of reusable software products. Such models will be applied in a Case study as a way of exemplifying their use.

Keywords: Software Reusability. On-Board Software. Nanosatellites. Software Systems. Embedded Systems. Firmware Reuse. Middleware Reuse.

LISTA DE FIGURAS

Pág.

Figura 2.1– Eixos da Engenharia de Linha de Produto - PLE. 19

Figura 2.2 – Processos de uma PLE. 21

Figura 2.3 – Esquema dos níveis de encapsulamento. 25

Figura 2.4 – Chamadas da classe "Efetuar LOG". 28

Figura 2.5 – Uso de aspectos no design dos Produtos-PLE. 29

Figura 2.6 – Encapsulamento de serviços considerando também os aspectos. 31

Figura 2.7 – Intent Specifications. 34

Figura 2.8 – Níveis de prontidão da NASA. 37

Figura 2.9 – Linha do Tempo de pesquisas sobre modelos de maturidade que incluem o reuso de software. 40

Figura 2.10 – Componentes do Modelo MPS.BR 2016. 42

Figura 2.11 – Modelo de Processos de acordo com GARCIA (2010). 44

Figura 2.12 – Exemplo de cenários inciais para reuso via PLEs. 48

Figura 2.13 – Desenvolvimento. 49

Figura 2.14 – Empresas que implantaram o CMMi. 52

Figura 2.15 – Desenvolvimento com Reuso Completo. 54

Figura 3.1 – Relacionamento entre o Projeto-Alvo e o Produto-PLE. 59

Figura 3.2 – Estágios do ciclo de vida da Biblioteca 60

Figura 3.3 – Camadas do software embarcado. 64

Figura 3.4 – Camada de Drivers. 65

Figura 3.5 – Periféricos internos de um Microcontrolador. 66

Figura 3.6 – BCI e seus constituintes. 68

Figura 3.7 – Intent Specifications Levels. 70
Figura 3.8 – Padrão Microkernel. 74
Figura 3.9 – Uso de Microkernel para subsistema AOCS-PLE. 75
Figura 3.10 – Dependências dos Produtos-PLE 77
Figura 3.11 – Exemplo de implantação dos Produtos-PLE 78
Figura 3.12 – Bibliotecas com SCA. 79
Figura 3.13 – Padrão de design da BCD 81
Figura 3.14 – Exemplo de satélite utilizando a Arquitetura Federada. 84
Figura 3.15 – Arquitetura Modular - IMA. 86
Figura 3.16 – MAPs de uma Arquitetura IMA. 87
Figura 3.17 – Nanosat híbrido com Arquiteturas Federada e IMA. 88
Figura 3.18 – Arquitetura SOIS 91
Figura 3.19 – Composição e agregação da Camera-PLE e SOIS-PLE. 93
Figura 3.20 – Uso de ferramentas de Ontologia na viabilidade 96
Figura 3.21 – Uso de ferramentas de Ontologia no índice de reusabilidade 99
Figura 4.1 – Módulo C&DH do Cubesat-1U Trailblazer (Arduino Based SDM-Lite - Single Board Computer Motherboard for Harsh Environments). 102
Figura 4.2 – Módulo C&DH do Cubesat-1U Trailblazer (PPM with Microchip® PIC24 for CubeSat Kit Motherboard). 102
Figura 4.3 – Funções para a BCI do MasterTimer-PLE. 104
Figura 4.4 – Exemplo de repositório BCD. 105
Figura 4.5 – PLE-Clock. 106
Figura 4.6 – Design da BCD com 2 Drivers e 2 Microcontroladores. 108

LISTA DE TABELAS

Pág.

Tabela 2.1 - Aderência da pesquisa aos processos GARCIA+ECSS. 45

Tabela 4.1 – Funções e Drivers segundo a IE para System and Components. 103

Tabela 4.2 –UCP do conjunto de C&DH de nanosats de uma instituição 111

Tabela 4.3 – Ocupação do clock nos C&DH de cada nanosat..... 112

LISTA DE SIGLAS E ABREVIATURAS

ADC	*Analog Digital Converter*
AOCS	*Attitude and Orbit Control System*
API	*Application Programming Interface*
ARP	*Aerospace Recommended Practice*
ASIM	*Applique Sensor Interface Module*
BCD	Biblioteca da Camada de Drivers
BCI	Biblioteca da Camada Intermediaria
BSP	*Board Support Package*
C&DH	*Command and Data Handling*
CBD	*Component-based software*
CBSE	*Component-based software engineering*
CCSDS	*Consultative Committee for Space Data Systems*
cFS	*core Flight System*
CMMI	*Capability Maturity Model® Integration*
COTS	*Commercial off-the-shelf*
DoD	*Department of Defense*
ECSS	*European Cooperation for Space Standardisation*
ESA	*European Space Agency*
ESSR	*European Space Software Repository*
FAA	*Federal Aviation Administration*
FACE	*Future Air Capability Environment*
FPSS	*File and Packet Store Services*
FP	*Function points*
GC	Gestão da Configuração
GnER	Ganho não Econômico do Reuso
GSFC	*Goddard Space Flight Center*

HAL	*Hardware Abstract Layer*
IE	*Intent Specification*
IEC	*International Electrotechnical Commission*
IMA	*Integrated modular avionics*
INCOSE	*International Council on Systems Engineering*
INPE	Instituto Nacional de Pesquisas Espaciais
ISO	*International Organization for Standardization*
JSC	*Johnson Space Center*
MAP	*Mission Application Partition*
MTS	*Message Transfer Service*
MRU	*Manage Reuse Unit*
NASA	*National Aeronautics and Space Administration*
OO	Orientação a Objetos
OSI	*Open System Interconnection*
PLE	*Product Line Engineering*
POA	Programação orientada a aspecto
POO	Programação Orientada a Objetos
PR	Potencial de Reuso
RAM	*Reusable Artifacts Management*
REI	*Reuse Integration into Software Life Cycle*
RGD	*Reuse Goals Definitions*
RIF	*Reuse of Graphical User Interfaces*
ROS	*Reuse of Open Source Solutions*
RRL	*Reuse Readiness Levels*
RSC	*Reusable Software Components*
RTCA	*Radio Technical Commission For Aeronautics*
RTOS	*Real-Time Operational System*
SCA	*Service Component Architecture*
SLOC	*Source line of code*

SO	Sistema Operacional
SOA	*Service-oriented Architecture*
SOC	*Service-oriented Computing*
SOIS	*Spacecraft Onboard Interface Services*
SOSE	*Service-oriented software engineering*
SPA	*Space Plug-and-play Architecture*
SPAREv	*Software Process for Adaptability, safe REuse and Variability*
SPL	*Software Product Line*
TAS	*Time Access Service*
TC	*Telecommand*
TDS	*Time Distribution Service*
TM	*Telemetry*
TPRs	Trabalhos (estudos ou pesquisas), Padrões e Recomendações
TR	Taxas (ou índice) de Reusabilidade
TRA	*Technology Readiness Assessment*
TRL	*Technology readiness levels*
UCP	*Use case points*
UML	*Unified Modeling Language*
V&V	*Verification and validation*

SUMÁRIO

Pág.

1. INTRODUÇÃO AO REUSO DE SOFTWARE 1

1.1. MOTIVAÇÃO 1

1.2. JUSTIFICATIVAS PARA O TEMA 3

1.3. IDENTIFICAÇÃO DO PROBLEMA DO REUSO DE SOFTWARE 4

1.4. HIPÓTESES OU PRESSUPOSTOS 5

1.5. RELEVÂNCIA E EXPECTATIVAS PARA O TEMA 6

1.6. OBJETIVOS DESTA PESQUISA 6

2. REVISÃO DA LITERATURA 9

2.1. INICIATIVAS DE REUSO NO DOMÍNIO ESPACIAL E CORRELATOS 10

2.1.1. ESA 10

2.1.2. NASA 10

2.1.3. DoD 12

2.1.4. *Consultative Committee for Space Data Systems* (CCSDS, 2016) 13

2.1.5. Indústria 14

2.2. AS ABORDAGENS DE APLICAÇÕES DO REUSO 15

2.2.1. Reuso baseado em ontologia 16

2.2.2. Reuso baseado em modelos 17

2.2.3. Reuso baseado em linha de produto 18

2.2.4. Reuso baseado em serviços 24

2.3. TÉCNICAS APLICÁVEIS 33

2.3.1. V&V, Níveis de Prontidão e Certificação 35

2.4. PROCESSOS 38

2.4.1. Aderência da Proposta de Reuso aos processos da Engenharia de Software 39

2.4.2. Níveis de capacidades segundo GARCIA+ECSS 43
2.4.3. Organização das equipes: Projeto-alvo e Projeto-PLE 47
2.4.4. Diferenças dos processos - Produtos-PLE e Projetos-alvo 47
2.4.4.1. Simulação de um cenário de reuso 48
2.5. UMA ABORDAGEM SOBRE A ECONOMIA ATRAVÉS DO REUSO 53
3. DESCRIÇÃO DA PROPOSTA DE REUSO 59
3.1. INTRODUÇÃO À PROPOSTA 59
3.2. ORGANIZAÇÃO DOS REPOSITÓRIOS 64
3.2.1. Biblioteca da camada dos drivers (BCD) 65
3.2.2. Biblioteca da camada intermediária (BCI) 67
3.2.3. A camada de aplicação 68
3.3. O PROJETO DOS COMPONENTES REUTILIZÁVEIS (PRODUTOS-PLE) 69
3.3.1. Adoção das *Intent Specifications* aos Requerimentos dos Produtos-PLE 69
3.3.2. Adoção do padrão *Microkernel* à arquitetura dos Produtos-PLE 73
3.3.3. Adoção do padrão *Service Component Architecture* (SCA) à arquitetura dos Produtos-PLE 79
3.3.4. Adoção do padrão *abstract factory* ao Design dos Produtos-PLE da BCD 80
3.4. O USO DOS PRODUTOS-PLE NOS NANOSATÉLITES 82
3.4.1. Reuso em *Nanosats* com Sistema Operacional (SO) 83
3.4.2. Reuso em *Nanosats* com Arquitetura Federada 84
3.4.3. Reuso em *Nanosats* com a arquitetura *Integrated Modular Avionics* (IMA) 86
3.4.4. Reuso de software com CCSDS-SOIS 90
3.5. VIABILIDADE ECONÔMICA 93
3.5.1. Ferramentas ligadas à Ontologias 95
3.5.2. Modelo e métricas para a viabilidade econômica 97
3.6. ÍNDICE DE REUSABILIDADE DOS PRODUTOS-PLE 98
4. ESTUDOS DE CASOS 101

4.1. ESTUDO DE CASO PARA UM REUSO DE FIRMWARE, ATRAVÉS DO COMPONENTE "CLOCK" EM DOIS NANOSATÉLITES 101
4.1.1. Os requisitos, segundo a IE 102
4.1.2. Arquitetura do Produto-PLE 104
4.1.3. O *design* da BCD 105
4.1.4. A flexibilidade do *design* para mais *nanosats* 109
4.2. ESTUDO DE CASO PARA A APLICAÇÃO DA VIABILIDADE ECONÔMICA 109
4.2.1. Resultados relativos à viabilidade 110
5. CONCLUSÃO 115
5.1. CONTRIBUIÇÕES DESTA PESQUISA 115
5.1.1. Contribuições principais 115
5.1.2. Contribuições adicionais e complementares 117
5.2. LIMITAÇÕES DESTA PESQUISA 118
5.3. TRABALHOS FUTUROS 119
5.3.1. Recomendações 119
5.3.2. Sugestões 120
5.4. COMENTÁRIOS FINAIS 120
REFERÊNCIAS BIBLIOGRÁFICAS 121
GLOSSÁRIO 133

1. INTRODUÇÃO AO REUSO DE SOFTWARE

Este capítulo introdutório apresenta os seguintes tópicos: motivação do trabalho, justificativa, identificação do problema, hipóteses, relevância e expectativas e objetivos da pesquisa.

1.1. Motivação

Reuso é uma prática sistemática e efetiva de desenvolver software, a partir de um conjunto de "blocos de construção", de forma que as similaridades dos ativos gerados no processo de Engenharia de Software como requisitos, *templates*, processos, modelos, códigos, etc, entre as aplicações, possam ser exploradas, para que sejam alcançados benefícios substanciais na produtividade, qualidade e no desempenho do sistema (VARNELL-SARJEANT, 2015).

Logo, pode-se pensar que, usando esses ativos de software constrói-se novos, ou modifica-se outros, de acordo com a missão do sistema. Esses ativos podem ser aplicativos inteiros ou parciais, inclusive *Commercial Off-The–Shelf* (COTS). No entanto, padrões de qualidade como *Radio Technical Commission For Aeronautics* (RTCA, 2011), alertam quanto ao *deadcode* [1], além de prejudicar a confiabilidade, no caso do reuso de ativos inteiros de forma parcial.

Reuso é uma prática antiga, e seu conceito foi pensado em 1968, quando Mcllroy, numa importante conferência de Engenharia de Software escreveu sobre o reuso numa abordagem orientada a componentes, o que hoje chama-se de *Component-Based Software Engineering* (CBSE), e *Software Product Line* (SPL) (SERAFIM, 2016). Nesta conferência foram abordados problemas pelos quais a Engenharia de Software passava, principalmente ligados à produtividade e à confiabilidade dos produtos de software.

[1] Na programação de computadores, o *deadcode* é uma seção no código-fonte de um programa que é executado, mas cujo resultado nunca é usado em qualquer outro cálculo. A execução do código morto desperdiça o tempo de computação e a memória.

Este é um conceito que pouco mudou, porém, muito evoluiu com o propósito de concretizá-lo. Hoje, existem muitas formas de aplicar o reuso, assim como muitas lições aprendidas tanto na aplicação do reuso quanto em seu planejamento.

De fato os benefícios do reuso são muitos, como citados a seguir:

- Diminuição de custos com o desenvolvimento, pois o escopo de desenvolvimento é menor:
    - Redução do tempo do ciclo de vida - Processos, de acordo com a *International Organization for Standardization* (ISO) / *Internacional Eletrotechnical Commission* (IEC) 15288:2008, poderiam ser evitados ou diminuídos;
    - A taxa de reuso chegou a 70% na Hewlett-Packard, através da abordagem SPL (KLAUS, 2005); e
    - A CelsiusTech reutilizava 40% de seus códigos, com isso ela inverteu a relação de custos software / hardware de 65:35 para 20:80 (KLAUS, 2005).
- Aumento contínuo da produtividade - Sistemas são entregues cada vez mais rápidos pois, a base de reuso é cada vez maior:
    - A Cummin's, através do SPL, apresentou uma significativa redução dos esforços necessários para produzir o software dos seus novos motores a diesel (MCGregor, 2004); e
    - A Market Maker, teve o tempo de desenvolvimento reduzido na criação de novos produtos em mais de 50% através da aplicação da SPL(KLAUS, 2005).
- Melhoria na manutenabilidade - A prática do reuso favorece a diminuição de erros:

    - A LG Sistemas Industriais – LGIS, reduziu a complexidade do seu sistema de controle de elevadores em 50%, através do uso da SPL (KLAUS, 2005); e
    - A Siemens Medical Solutions, teve uma melhoria no reuso de casos de testes de 57%, através de método desenvolvido para testes de produtos baseados em SPL (KLAUS, 2005).

- Redução dos custos com treinamentos - Equipes poderiam já ter trabalhado com os ativos reutilizáveis;
- Aumento da confiabilidade - Os componentes já foram exercitados em outros sistemas, afinal, quanto mais se usa, menos *bugs* são encontrados (MAXIM, 2013);
- Redução nos riscos dos processos - Menores incertezas dos custos de desenvolvimento (MAXIM, 2013);
- Uso efetivo de especialistas - Favorecimento do reuso de componentes, ao invés de pessoas (MAXIM, 2013); e
- De acordo com padrões - Os componentes reusáveis, podem estar de acordo com normas, recomendações e práticas (MAXIM, 2013).

1.2. Justificativas para o tema

Atualmente, instituições na área espacial como a European Space Agency (ESA) e a National Aeronautics and Space Administration (NASA), além da àrea de defesa como no *Department of Defense* (DoD) americano praticam, de alguma forma, o reuso de software. Alguns dos programas que focam no reuso serão vistos no Cap. 2.1.

Segundo BAYER (1999), os métodos de reuso, ou não são flexíveis o suficiente para encontrar as necessidades de várias situações industriais, ou eles são muito vagos e não aplicáveis sem uma forte interpretação adicional e suporte. Um método flexível que pode ser adaptado à várias situações empresariais, fundado em processos e suas capacidades seriam necessários.

Foram discutidos até o momento motivações e justificativas do por quê é importante a discussão deste tema. Foram mostrados benefícios do reuso e sua aplicação em segmentos importantes. Ou seja, além da prática do reuso trazer naturalmente benefícios, será mostrado, no Cap. 2.1, uma tendência na generalização do hardware e, consequentemente a elevação da quantidade dos produtos de software que também favorecem o reuso.

1.3. Identificação do problema do reuso de software

A forma de implantar o reuso numa instituição ou companhia, de acordo com os seus portfólios[2] de produtos, algumas vezes não é tão direta.

Este assunto continuará a se desenvolver como ocorre naturalmente em qualquer outro ramo da Engenharia. Entretanto, na atualidade, já nos traz muitos aprendizados, tanto teóricos quanto práticos. Nestes aprendizados tem-se erros e acertos, técnicas boas e ruins e outras específicas para um determinado portfólio que podem não servir ou devam ser adaptada para outros.

Trata-se de um assunto que foi por muito tempo praticado sem qualquer formalidade ou planejamento. Algumas vezes diretamente no código, muitas vezes sem gestão de configuração e do que foi reutilizado.

Desta forma, o reuso sem documentação ainda é algumas vezes praticado, onde semelhanças de produtos no nível do código são conhecidas e, então, o reuso de forma não planejada é usado. Esta é a pior forma de se beneficiar dele, na utilização não-consciente do reuso, principalmente, quando tudo funciona, apesar de ainda assim não haver o benefício propriamente, uma vez que a operacionalidade não é um deles. Muitas vezes, o código legado mereceria tamanha alteração, a ponto de realmente não compensar o reuso, mas por motivos de desconhecimento sobre o projeto e sua documentação,

[2] Termo de acordo com guia PMBOK 5a. Ed. Se a relação entre projetos for somente a de um cliente, vendedor, tecnologia ou recurso compartilhado, o esforço deve ser gerenciado como um portfólio de projetos e não como um programa.

não é possível fazer tais avaliações sem ter que se aprofundar no desenvolvimento e, então, fica tarde para desistir e recomeçar.

Alguns casos semelhantes de insucesso foram relatados. O Ariane 5, em 1996, sofreu uma explosão e isso acarretou na perda total do investimento. Outro acidente relevante neste contexto, foi o da sonda *Mars Climate Orbiter* da NASA, em 1999 (WEISS, 2004), e também o da máquina de radioterapia controlada por computador – Therac-25, envolvida em seis acidentes fatais entre 1985 e 1987, devido a overdoses massivas de radiação (LEVESON, 1993). Em todos os casos, foram atribuídas falhas na documentação, ou com o projeto de reuso do software.

Existe um forte desejo frequente de se reutilizar o código, e isto será comumente mais barato, se o código for desenvolvido para ser editado ou substituído. Por outro lado, um projeto desenvolvido sem os requisitos de manutenibilidade pode ser mais barato, mais eficiente e menos arriscado, se simplesmente descartar o software legado, e se iniciar o projeto do começo novamente (LARSON, 1999).

1.4. Hipóteses ou pressupostos

Boas especificações fazem parte do planejamento, e incluem a rastreabilidade dos requisitos e razões (*rationales*) de *design*. Eles são críticos para sistemas complexos como satélites, principalmente aqueles cujo software é intenso (alto escopo, equipes distribuídas e alta criticidade), e aqueles cujos componentes de software são projetados para serem reutilizados. As especificações devem ser entendidas e revisadas pelos especialistas do domínio, e pelos engenheiros-chave, incluindo os operadores. As informações necessárias ao reuso ou às mudanças dos componentes, bem como a operação segura de um satélite, por exemplo, deve ser facilmente encontrada quando necessário (WEISS, 2004).

Em JACOBSON (1996), explica-se que, para o reuso acontecer, deve-se projetar uma arquitetura de sistema e ter processos, além de haver uma harmonia na organização para torná-lo real.

Os benefícios do reuso vêm se concretizando e o mérito é, principalmente, devido à organização e aos processos exigidos pela Engenharia de Software e Engenharia de Sistemas, descritos hoje no INCOSE, ISO 15288, ISO 25010, DO-178, *European Cooperation for Space Standardisation* (ECSS), por exemplo. Somam-se também a evolução das abordagens da Engenharia da Confiabilidade e *Safety* descrito na *Aerospace Recommended Practice* - ARP-4761, por exemplo, além das contribuições de diversos autores especialistas nestes temas, que complementam os padrões oficiais. A ESA usa *Technology Readness Levels* (TRL) e a NASA usa *Reuse Readiness Levels* (RRL) para aferir o nível de “prontidão” do software ou de reuso conseguido ao aplicá-lo num projeto.

Reuso é o exemplo de conceito, que é aperfeiçoado com o tempo. Ele não nasce pronto de um único estudo, e nem é constituído de tecnologias próprias e específicas, ou seja, feitas exclusivamente para o reuso, mas sim da evolução natural do conhecimento sobre o desenvolvimento de software e de sistema. O reuso nem sempre funcionou e muito menos é simples de ser aplicado. Esta controvérsia é baseada não somente em utilizar códigos já concebidos num novo projeto, esta é a parte simples, mas no planejamento do reuso, pois está claro depois de casos de insucesso que o planejamento é necessário para que os benefícios sejam verificados.

1.5. Relevância e expectativas para o tema

O reuso de software, que pode ocorrer em qualquer estágio do ciclo de vida, permite oportunidades de benefício da redução do custo, tanto do desenvolvimento quanto da manutenção. Entretanto, o reuso também traz riscos em sua aplicação que podem resultar no sentido oposto. Logo, é importante que esta pesquisa forneça meios para reconhecê-los e formas de mitigá-los.

1.6. Objetivos desta pesquisa

A concretização do reuso, de forma a considerar sua viabilidade no desenvolvimento de produtos reutilizáveis, é um desafio a qualquer instituição

ou companhia que tenha a intenção de se beneficiar do mesmo. Todas as instituições citadas anteriormente passaram por este desafio. No Cap. 2.1, se mostra tanto ações concretas que favoreceram o reuso, quanto a preocupação do retorno sobre o investimento do mesmo. Hoje, servem de referência, porque muitos contratempos houveram, e vêm sendo superados, corrigidos e felizmente, alguns, divulgados.

Com base nestas ações e preocupações relatadas por instituições de renome, e nos conhecimentos teóricos relacionados ao reuso de software, atualmente disponíveis, esta pesquisa tem por objetivo propor táticas ou meios fundados no uso de técnicas, processos e abordagens, para favorecerem o reuso de planejado de software, no desenvolvimento de firmwares e middlewares embarcados a bordo de nanosatélites. Tal objetivo possui duas frentes: (1) apresentar as táticas que promovam o reuso planejado e (2) apresentar o emprego do seu resultado em projetos de nanosatélites.

Tendo o objetivo de exemplificação e contribuição com o entendimento desta proposta, será apresentado um Estudo de Caso de aplicação do reuso de uma mesma funcionalidade, que pertence à camada de firmware, em dois nanosatélites, que possuem componentes de processamento diferentes.

A viabilidade econômica, está ligada à tangência dos benefícios do Reuso, já explorados anteriormente. Uma vez que tais benefícios se concretizem, ficará demonstrado que, com a quantidade finita de recursos, foi feita a melhor escolha de alocação. A fim de mitigar o risco da má escolha de alocação dos recursos, métricas e medições devem ser aplicadas, antes da execução dos processos de desenvolvimento dos produtos de software reutilizáveis. Logo, o objetivo é estruturar um modelo de análise de viabilidade econômica, onde será possível conhecer de antemão as probabilidades de reuso destes produtos.

Através do modelo de viabilidade econômica, será apresentado um Estudo de Caso focado em sua aplicação, com o objetivo de exemplificação e contribuição com o seu entedimento. Logo, um cenário de aplicação será

construído, e nele implementado tal modelo, a fim de contribuir com a análise das probabilidades de reuso dos produtos de software reutilizáveis.

2. REVISÃO DA LITERATURA

O Reuso pode ocorrer em todos os processos do ciclo de vida dos projetos, seguindo as práticas da Engenharia de Software[3], ou seja, pode-se ter reuso de requisitos, funções, modelos, códigos, testes, etc.

Destaca-se como fundamentos para a implementação do Reuso, três grandes pilares, onde pode-se interagir, para que haja a compatibilidade na implantação da prática, de forma a minimizar os riscos. São eles: (1) As abordagens de reuso que descrevem como e onde o reuso poderá ser aplicado (VARNELL-SARJEANT, 2015); (2) As técnicas de implementação destas abordagens, que as aproximarão dos tipos de produtos, neste caso, os produtos espaciais de bordo, e (3) Os processos de desenvolvimento e de suporte[4], incluindo, modelos de capacidades para dar a flexibilidade de escolha às instituições, quanto ao nível de profundidade que desejam aplicar o Reuso, e por fim, modelos de organização das equipes, uma vez que o foco do desenvolvimento de produtos reutilizáveis não é necessariamente para um único projeto, e sim para um portfólio.

O retorno do Reuso, através de seus ganhos já apontados no Cap. 1.1, precisa ser quantificado e colocado como economicamente viável. Logo, serão apresentados conceitos ligados à economia do projeto, quando o reuso é praticado.

Mas antes de iniciar a revisão bibliográfica desses pilares, e da economia do reuso, é importante mostrar um histórico de iniciativas ligadas ao tema, tanto no domínio espacial, quanto da indústria em geral, a fim de mostrar como o assunto foi, e é explorado atualmente.

---

[3] Processos da ISO/IEC 12207:2008 .

[4] Processos de suporte são: V&V, Gerenciamento da Configuração, Garantia da Qualidade e de Certificação.

2.1. Iniciativas de reuso no domínio espacial e correlatos

2.1.1. ESA

A ESA, por exemplo, possui o *European Space Software Repository (ESSR),* que é um repositório de componentes para promover o reuso de software de código aberto, e permite que todos os parceiros envolvidos com desenvolvimento de software para a ESA, tenham acesso aos resultados de investimentos anteriores (LIVSCHITZ, 2014). A utilização destes componentes em projetos está disponível em *European Space Software Repository (*ESSR*)* (2017).

Outro programa de reuso que a ESA vem desenvolvendo é o *Onboard Software Reference Architecture*. Neste, a proposta é disponibilizar uma arquitetura única, dentro da abordagem CBSE, para uma solução comum dos sistemas de software de bordo e uma das aplicações, pode ser vista em CORDET-2 (2016).

2.1.2. NASA

A NASA tem uma excelente reputação para o reuso de software. Por exemplo, em 1979 um estudo reportou uma taxa de reuso de 32% em 25 sistemas de software, para suporte em solo de veículos não-gerenciados, onde se atingiu a marca de 3000 a 120000 linhas de código. Em outro estudo, indicou um Retorno sobre o Investimento de 4:1 em reuso, de acordo com o Centro de Pesquisas Ames/NASA (HOLMGREN, 1990).

A NASA investe em reuso desde o inicio dos anos 90, de acordo com Jeffrey Poulin, engenheiro chefe da *Lockheed Martin Distribution Technologies* e autor do livro "*Measuring Software Reuse: Principles, Practices and Economic Models"* (POULIN, 1996).

Segundo Poulin (1996), a NASA é uma boa candidata à prática, porque seus softwares são semelhantes e requerem pouca adaptação. A melhor estratégia da NASA seria investir nos softwares focados ao reuso desde o começo do

desenvolvimento, mesmo sendo mais caro inicialmente. De acordo com Poulin, enquanto estes softwares podem custar 50% a mais no desenvolvimento, considerando os componentes preparados para o reuso, isto irá refletir numa economia em torno de 80% nos custos de desenvolvimento, quando tais componentes forem reutilizados em outros software (FREDERICK, 2006).

Assim como na ESA, a NASA também possui um repositório de componentes reutilizáveis, o *core Flight System (cFS)*. Esta suíte de aplicações é composta por 12 *cores* individuais de *Command and Data Handling (C&DH)*, que juntos criam uma biblioteca de funções C&DH comuns. Ele é mantido pela Divisão de Engenharia de Software da *Goddard Space Flight Center*(GSFC).

A suíte de aplicações *cFS* permite aos desenvolvedores rapidamente configurar e implantar uma quantidade significativa de software para o subsistema C&DH em novas missões, plataformas de testes e protótipos, resultando na redução dos prazos e custos (LEONARDI, 2015).

Projetos que aplicaram o *cFS* podem ser vistos em PROKOP (2017), cujo destaque é o Morpheus Lander do *Johnson Space Center* (JSC). O projeto Morpheus é um dos 20 projetos de sistemas avançados na direção de missões de exploração e operações humanas da NASA desenvolvida no JSC.

O veículo autônomo Morpheus é um *robot test-bed*, que inclui quatro tanques de combustível / oxidante, em um único propulsor principal, e vários jatos do Sistema de Controle de Reação. Seu principal objetivo é testar novos sistemas de propulsão de combustível verde e tecnologia de detecção de perigo. No final de 2011, o projeto *Autonomous Landing Hazard Avoidance Technology* uniu forças com Morpheus para perseguir esses objetivos comuns. Estas tecnologias visam ser incorporadas em um veículo de descida de pouso usado na Lua, em um asteróide ou em Marte.

Trata-se de um exemplo de como uma demonstração de tecnologia com uma programação desafiadora poderia ser atendida, usando a arquitetura *cFS*. O GSFC-*Applied Engineering and Technology Directorate* recebeu um prêmio

Morpheus do projeto da JSC pelo benefício e suporte recebido devido ao uso do cFS. A placa do prêmio declara: "*Não poderíamos ter feito o rápido progresso que temos sem a qualidade, o profissionalismo e os conhecimentos fornecidos pela Divisão de Engenharia de Software da GSFC, e pela Direção de Engenharia e Tecnologia Aplicada*" (cFS, 2017).

2.1.3. DoD

O Departamento de Defesa dos EUA (DoD) também faz investimentos massivos no reuso de software: em 1992 o DoD publicou o *"DoD software reuse initiative: vision and strategy"*, uma espécie de inicialização oficial ao assunto, ou uma abertura à discussão do tema. O governo investiu pesado em reuso como no *Control Channel Toolkit* em 1997, que reforçou a abordagem do reuso de *Software Produt Line* (SPL). Em seguida, com o *Global Broadcasting Service* em 1998 o qual, naquela ocasião, mostrava índices de reuso. O governo solicitava que as propostas tivessem requisitos da quantificação esperada em economia devido ao reuso, já indicando a importância da viabilidade econômica do reuso.

Um programa vigente do DoD é o *Future Air Capability Environment* (FACE), que representa uma arquitetura padrão, com o foco, entre outras coisas, de reduzir o custo do ciclo de vida, gerenciar a obsolescência e aumentar a produtividade, acelerando a velocidade da integração de novas capacidades, etc. As características chave desta arquitetura são padronizar as interfaces, fornecer suporte aos princípios de camadas, facilitar a abstração e conter atributos de segurança (*security and safety*).

Esta arquitetura possui pacotes de serviços de transporte, componentes portáteis e serviços específicos de plataformas (de aviões, por exemplo). Tais pacotes passam por um processo de conformidade que inclui a verificação, a certificação e o registro, ficando assim disponíveis em bibliotecas para serem utilizados em projetos (STEVENS, 2016).

Um exemplo de uso do FACE ocorreu no *AV-8B Harrier II,* onde novas capacidades foram construídas, adicionadas e então compartilhadas entre o *T-45 Goshawk e o F/A-18 SuperHornet*. Em outras palavras, constrói-se uma vez e se usa em três plataformas de aeronaves diferentes, de acordo com Zaffanella - vice-presidente da *General Dynamics Mission Systems* (BELLAMY ,2016).

2.1.4. *Consultative Committee for Space Data Systems* (CCSDS, 2016)

Voltando ao contexto espacial, em 1982 foi fundado pela maioria das agências espaciais do mundo um fórum multinacional, voltado ao desenvolvimento das comunicações e a padronização dos sistemas de dados para equipamentos espaciais. Este fórum, conhecido como *Consultative Committee for Space Data Systems* (CCSDS, 2016), cujo objetivo é aprimorar a interoperabilidade comercial e governamental, além de favorecer o suporte, também reduz os riscos, o tempo de desenvolvimento e os custos dos projetos entre as nações envolvidas.

Isso é possível devido ao interesse das próprias nações em estabelecer uma base comum de informações, mas tecnicamente pela característica das semelhanças de funcionalidades no âmbito sistêmico. Uma das áreas de concentração do CCSDS é o *Spacecraft Onboard Interface Services* (SOIS), cujo foco, inclui a padronização das funções de um satélite genérico (SCHNURR, 2006).

Uma vez que os serviços SOIS semelhantes de um portfólio de nanosalitélites foram mapeados, bem como a interface de rede e a padronização dos controladores de hardware e periféricos, a probabilidade destes serviços estarem presentes em todos os *nanosats* é alta, logo, os serviços SOIS podem ser compartilhados com outros *nanosats*, e a possibilidade de reuso em larga escala é evidente.

É importante ressaltar no SOIS, uma possível implementação quanto ao protocolo de mensagens, baseado em pacotes trocados entre, a recepção de

telecomandos e o envio de telemetrias. O SOIS, faz referência a ECSS-E-ST-70-41C (2016) (*Space Engineering—Ground Systems and Operations—Telemetry and Telecommand Packet Utilization*), cujo propósito é padronizar tais comunicações, uma vez que já era conhecido o alto grau de similaridades desta função sistêmica entre os diversos tipos de missão que um equipamento pode apresentar.

2.1.5. Indústria

Saindo do mundo espacial e da defesa, e expandindo às indústrias do software crítico embarcado e de tempo-real DOUGLASS (2002) da IBM comenta que neste tipo de aplicação 60% a 90%, são muito similares à aplicações anteriores. De fato numa organização isso ocorre, pois os produtos que requerem este tipo de software servem a um mesmo portfólio[5], pois estão dentro de um determinado segmento, e compartilham de elementos (ativos) comuns, como por exemplo:

- Gerenciamento na criação e destruição dos objetos;
- Tratamento das interrupções;
- Execução das máquinas de estado; e etc.

Uma realidade dos produtos embarcados é o aumento das plataformas de hardware genéricas ou COTS programáveis, e consequentemente, o crescimento da quantidade de software na alocação funcional do sistema, onde 95% dos componentes dos sistemas embarcados são reutilizados, e 90% desses sistemas são compostos por software, conforme já observava SPECHT (2007).

Um exemplo disto, é o Rádio Definido por Software, onde funcionalidades antes restritas ao hardware são configuráveis por software. Outro são os

[5] Termo de acordo com guia PMBOK 5a. Ed. Se a relação entre projetos for somente a de um cliente, vendedor, tecnologia ou recurso compartilhado, o esforço deve ser gerenciado como um portfólio de projetos e não como um programa.

módulos tipo COTS, programáveis para a indústria espacial como por exemplo, no conceito da arquitetura *Plug-and-play (Space Plug-and-play Architecture* - SPA). Esta arquitetura contém uma série de módulos de interface padrão, *Applique Sensor Interface Module (ASIM*), que permitem a construção de um nanosatélite inteiro, em termos do hardware e processamento para o software, pois permite, que o desenvolvedor construa suas camadas funcionais, e a de aplicação para o projeto. Logo, o emprego de plataformas de hardware programáveis, indiscutivelmente, colabora e reforça a importância da prática do reuso (LYKE, 2010).

Foram discutidos os benefícios do reuso e sua aplicação em segmentos importantes, além de mostrá-los como forma de prática do reuso, incluindo a utilização, sempre embasada numa análise de estimativa para a validação posterior.

Foi mostrado o benefício na generalização do hardware, e consequentemente, a elevação da quantidade dos produtos de software favorecem a prática do reuso.

A partir dos relatos das agências espaciais, e do DoD dos EUA, percebe-se uma tendência no uso de arquiteturas flexíveis, e de bibliotecas de reuso como forma de sua implementação, com o objetivo de serem aplicadas em contextos ou missões diferentes.

2.2. As abordagens de aplicações do reuso

O reuso envolve o entendimento atualizado das abordagens e técnicas que podem ser usadas na implantação desta prática. Pode-se destacar atualmente, quatro grandes abordagens para tratar o reuso: (1) Baseado na Ontologia[6]; (2) Baseado em Modelos; (3) Baseado em Linha de Produto; e (4) Baseado em Serviços.

[6] Ontologia possui um significado muito maior, entretanto nesta pesquisa, seu uso restringe-se a identificação de ativos semelhantes dentro do domínio do Reuso.

Em seguida analisá-se cada uma dessas abordagens e seus propósitos.

2.2.1. Reuso baseado em ontologia

Ontologia é o termo usado para se referir à entendimentos compartilhados em um domínio de interesse. Trata-se do principal recurso atualmente usado em sistemas de busca da internet, por exemplo. É possível conhecer mais sobre a Ontologia em GRUBER (2009).

Segundo este conceito (VARNELL-SARJEANT, 2015), é possível ganhar produtividade, pois os dados são disponibilizados para a análise do desenvolvedor do projeto-alvo[7], pois um repositório único, com os dados de vários projetos, pode, através da ontologia serem entendidos, terem seus dados identificados e trazidos para um projeto-alvo[7].

Esta abordagem trabalha com vários tipos de ativos como: requisitos, funcionalidades, links para os modelos, testes dos produtos, códigos, além de informações em geral de documentações (VARNELL-SARJEANT, 2015).

Nesta pesquisa, o reuso baseado em ontologia aplica-se durante o desenvolvimento da base de reuso, com o objetivo de filtrar as similaridades dos ativos dos projetos, e também durante o seu processo de desenvolvimento, onde filtram-se os ativos de reuso que já pertencem a uma base comum de interesse. Entretanto, não contribui com o desenvolvimento dos ativos [8] reutilizáveis, no sentido de garantir que os ativos no repositório, de fato estão preparados para serem reaproveitados. Logo, esta abordagem é útil em determinados momentos do ciclo de vida do projeto.

[7] Projeto-alvo é o projeto onde o reuso será aplicado.

[8] Os ativos de interesse desta pesquisa são os requisitos e os componentes de software reutilizáveis, bem como as documentações envolvidas. O foco nestes ativos se dá devido à deficiência no planejamento, critério identificado como causa de problemas e acidentes na aplicação do Reuso.

Este assunto será visto com maior profundidade no momento da exploração da proposta de reuso, como um dos marcos deste trabalho, mais especificamente, no estágio da conceituação do ciclo de vida do produto, no Cap. 3.1.

2.2.2. Reuso baseado em modelos

Na Engenharia Baseada em Modelos (*Model-Based Engineering*), o uso de ferramentas e linguagens de modelagem, como por exemplo a IBM *Rational Rhapsody* e a *Unified Modeling Language* (UML) / *Systems Modeling Language*, são comuns e atualmente amplamente aplicados em indústrias. Isto porque dão suporte através da simulação, e padronizam o desenho do projeto, colaborando diretamente com a comunicação entre os profissionais da área de forma global, contribuindo também com a demonstração e divulgação de boas práticas, como os *design patterns,* por exemplo.

Os modelos em um contexto de '*model-based*' ou '*model-driven*', assumem um papel central e governam especificações, *designs*, integrações, validações e operações de sistemas (VARNELL-SARJEANT, 2015), pois usam descrições abstratas de software, como forma primária de expressão (DOS SANTOS, 2008). Com isto, uma vez assumindo o reuso como intenção, seria possível aproveitar esses modelos, a fim de contribuir com o projeto-alvo. Tais modelos, reutilizáveis, oferecem maior simplicidade na integração com o projeto-alvo, e consequentemente, ocasionam ganhos de produtividade. É comum nesta abordagem haver reuso de modelos de arquitetura, funcionalidades, modelos de desempenho e simulações.

Essa abordagem oferece um meio de realizar o reuso. É claro que os ativos deverão ser, desde o inicio, pensados em prol do reuso, afetando intrinsicamente seu comportamento e estrutura. É uma excelente forma de construir o reuso através dos modelos, por questões de padronização e abrangência na comunicação. Logo, esta abordagem será útil na construção dos ativos reutilizáveis, principalmente, quanto às questões de arquitetura e de padrões de projeto.

2.2.3. Reuso baseado em linha de produto

Esta abordagem é a mais direta e objetiva relacionada ao tema desta pesquisa, pois de fato colabora com a construção dos ativos em prol da reusabilidade. A *Product Line Engineering* (PLE[9]), considera o portfólio como a única entidade a ser gerenciada, diferente da visão de gerenciar os produtos de forma individual. Isto traz aos sistemas e à Engenharia de Software uma perspectiva de fábrica, entregando melhorias na produtividade (SEI, 2013).

Por isso, a PLE se adequa às instituições, e companhias que atuam em determinados segmentos, sendo capaz de fornecer um ganho ainda maior àquelas cujo segmento é de alguma forma normatizado, como o Espacial por exemplo, pois a padronização sugerida pelas normas, bem como as práticas também sugeridas pelo CCSDS, servem de garantias que a PLE precisa para ser bem aproveitada, porque esta abordagem opera sobre as similaridades dos produtos.

A construção da PLE envolve a construção de produtos de software reutilizáveis (Produtos-PLE), e é dividida em três áreas de concentração. De acordo com a Figura 2.1: (1) O gerenciamento do ciclo de vida de cada produto (eixo Y ou vertical), (2) A evolução do produto no tempo (eixo X ou horizontal) e, (3) O gerenciamento da pluralidade dos produtos (eixo Z ou ortogonal a X e Y).

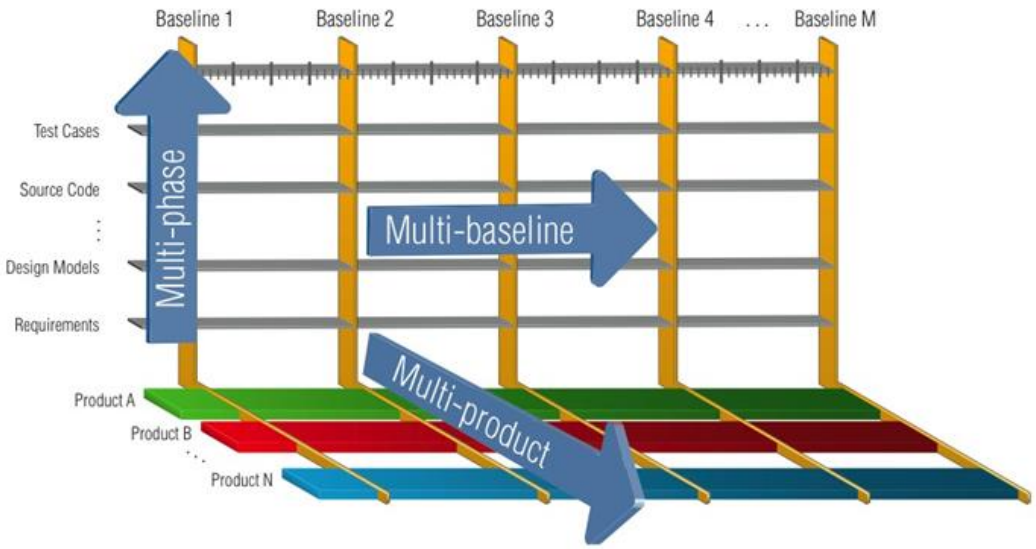
No eixo *Multi-phase,* existe a preocupação do ciclo de vida de engenharia, como: requisitos, arquiteturas, *designs*, implementações, testes e etc, além da rastreabilidade para manter a interdependência entre as fases.

Uma *feature* nasce da visão das similaridades do segmento, de um determinado portfólio de produtos dentro de uma instituição ou companhia.

[9] Baseado na segunda geração da Engenharia de Linha de Produto (2GPLE).

A PLE funciona como um repositório reutilizável de *features*, que contém um conjunto de ativos com suas versões associadas. Assim, uma *feature* possui seus requisitos, modelos, códigos, testes e registros de conformidade, por exemplo. As features podem possuir dependências de outras features, tanto em caráter de agregação (não obrigatório), quanto de composição (obrigatório), e isto deve ser considerado ao aplicar a feature no projeto-alvo. É esperado na PLE que as features devam passar por algum tipo de customização, a fim de se adequar às variações que cada projeto possa exigir.

Figura 2.1– Eixos da Engenharia de Linha de Produto - PLE.



Fonte: SEI1 (2013).

O eixo *Multi-baseline* traz as preocupações temporais habituais da Engenharia de Produtos, como: versão, configuração e gerenciamento de mudanças. A característica que mantêm a gestão de configuração (GC) dos ativos (e não dos produtos) da PLE, suporta a evolução de todos os ativos, sem ter que gerenciar de forma independente, a evolução de cada um dos produtos em uma linha de produtos.

Um conjunto de ativos da PLE disponíveis é que são gerenciados, e não os produtos ou sistemas individuais. Uma nova versão de um produto não é derivada de uma versão anterior do mesmo produto, mas a partir do conjunto compartilhado dos próprios ativos da PLE.

Pode-se entender que uma *feature* é composta de um conjunto de ativos, e existe uma GC para cada *feature*, logo, uma nova versão do produto pode surgir devido ao rearranjo destas *features,* através de mudanças nas interdependências, ou de mudanças dos próprios ativos de um determinado conjunto.

Sob o paradigma da fábrica de PLE, todos os defeitos são corrigidos nos ativos, não nos produtos. Os produtos afetados serão então regerados.

O eixo *Multi-product,* indica os Produtos-PLE construídos através de uma *Baseline* e seus ativos.

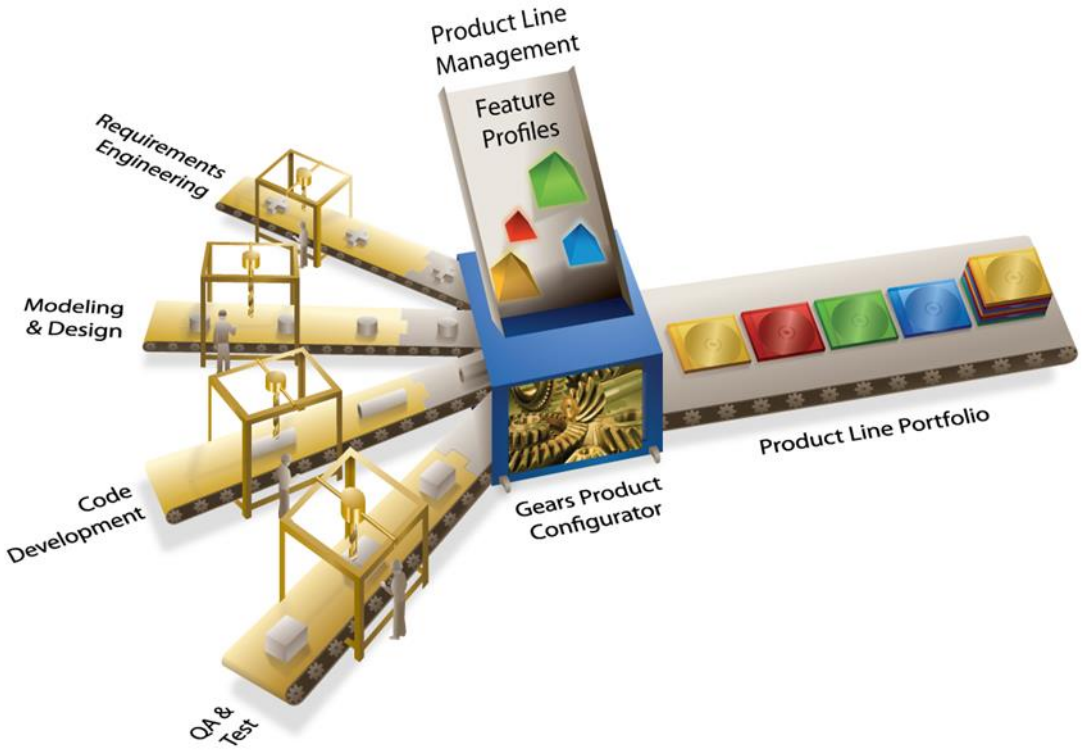
A Figura 2.2 mostra o processo de desenvolvimento de uma PLE. Este processo implica em três atividades essenciais, como:

(1) Desenvolvimento das *features*;

(2) Desenvolvimento dos ativos; e

(3) Configuração do produto.

Figura 2.2 – Processos de uma PLE.



Fonte: SEI2 (2013).

O desenvolvimento das *features* se trata de um nível de abstração dos ativos. Uma vez que as *features* são conjuntos de ativos, é necessário de forma macro entender o que cada *feature* representa, o seu propósito e suas possíveis variações.

Por exemplo, supondo que o subsistema *Attitude and Orbit Control system* (AOCS) em um satélite seja uma *feature*-PLE, então a intenção é tornar o subsistema AOCS um produto de linha de fabricação. Para isto, faz-se necessário conhecer: as órbitas de operação, os meios de estabilização adotados, os apontamentos previstos, entre outros requisitos. Trata-se de uma especificação do ponto de vista da Engenharia de Sistemas, segundo os processos da ISO/IEC 15288:2008.

Nesta especificação existirão pontos de variação, ou seja, atributos do susbsistema que devem ser ajustados para cada produto do tipo AOCS, por exemplo: órbita de operação, meios de estabilização, estratégias de

apontamento, níveis de saúde, etc. Tais atributos podem ou não serem comuns a todos os subsistemas, entretanto, é altamente recomendável que existam em todos os eles, a fim de colaborar com a eficiência da *feature*-PLE, não desperdiçando recursos parcialmente necessários, e não deixando recursos "mortos" em determinados produtos, além de contribuir com a confiabilidade dos mesmos, evitando mais pontos de falhas e manutenções. Logo, os pontos de variações deverão ficar explícitos na especificação dos perfis da *feature*-PLE.

Uma vez definidas as *features*, passa-se para o desenvolvimento dos ativos que compõem este conjunto. Neste momento, se está em um nível de abstração menor, pois agora olha-se para cada ativo, e deve-se, através dos processos do ciclo de vida da Engenharia de Software[10], desenvolver os ativos dentro do escopo definido pela *feature*. Assim, as áreas de requisitos, modelagens, implementações, testes, qualidade e etc, participam (Figura 2.2).

O configurador do produto é aquele que conhece a *feature*, os seus perfis, os seus ativos e o produto que se deseja produzir. O objetivo do configurador é adequar a *feature* para o Produto-alvo, através da valoração de cada atributo que foi definido para os perfis.

Retomando o subsistema AOCS, como uma *feature*, o configurador para um determinado satélite, configuraria diferentes estratégias de aquisição em função da órbita de operação *(Geostationary Orbit*, *Low Earth Orbit*, etc.), diferentes estratégias de estabilização, em função dos meios de estabilização escolhidos (dual spin, 3 eixos, etc.) e diferentes estratégias de apontamento (Nadir, Sol, etc.) por exemplo.

Para atingir tais aspectos a PLE, por si só, não é suficiente, basicamente porquê trata-se de uma tecnologia, um conceito, e a preocupação está em demonstrá-la. Por outro lado, as preocupações com a sua implementação, considerando as interfaces dos produtos, os princípios, e as boas práticas da

[10] Segundo os processos da ISO/IEC 12207:2008.

Engenharia de Software, como por exemplo: a coesão e o baixo acoplamento, que interferem em pontos estruturais dos produtos, ocasiona a necessidade de uma outra abordagem que a implemente.

A PLE apresenta uma forma contínua de fabricação de ativos reusáveis, entretanto, é parcial em alguns aspectos de qualidade dos produtos de acordo com a IEC 25010 (2011) pois, apesar de haver preocupações com a construção dos produtos, onde é possível considerar aspectos funcionais, de desempenho, de confiabilidade, usabilidade, e segurança, por exemplo. Não há de forma direta outros pontos quanto à compatibilidade, à portabilidade e à cobertura de contexto[11], preocupações intrínsecas do Reuso.

A pressão por redução de custos e prazos, o atendimento pronto aos requisitos não funcionais, *Quality of Service*, além do respeito ao corpo técnico da instituição com relação ao escopo, leva à adoção de mais software de terceiros, e software livres, afinal, podem existir produtos-alvo em que se demandem o desenvolvimento de partes fora do *core* da companhia, ou até de sua missão. A expectativa é que, adotando softwares de terceiros, a qualidade do software e o *time-to-market* podem ser melhorados.

Infelizmente, software de terceiros, geralmente, não se integra à Tecnologia de linha de Produtos proprietária, pois a tecnologia de Linha de Produtos, por si só, caso não adotadas outras medidas, pode exigir componentes especialmente preparados, e de controle fino sobre os ativos de software, inviabilizando interfaces com produtos de terceiros. Uma vez que a Tecnologia de Linha de Produtos é fechada, no sentido exposto, os softwares de terceiros poderão não se encaixar, e como resultado, a utilização de software de terceiros é preferida à Tecnologia da Linha de Produtos ou, vice-versa (DE JONGE, 2009).

[11] Para o reuso é importante que exista uma coesão entre os requisitos das estruturas geradas a partir da PLE.

2.2.4. Reuso baseado em serviços

O modelo *Service-Oriented Computing* (SOC) foi definido em GEORGAKOPOULOS (2009) como um paradigma que utiliza serviços como elementos fundamentais do desenvolvimento de aplicações distribuídas. Em uma implementação do SOC, a *Service-oriented Architecture* (SOA), possui as seguintes características:

(1) Baixo acoplamento – mínima quantidade de interdependências entre os serviços;

(2) Serviço contratado – a interação entre serviços é baseada na comunicação e, em um documento acordado;

(3) Abstração – serviços escondem sua lógica interna dos ambientes externos, exceto aquelas partes que estão descritas no contrato do serviço;

(4) Reusabilidade – divisão da lógica em serviços para promover o reuso;

(5) Composabilidade – Serviços podem ser montados de outros serviços;

(6) Habilidade para descobrir – existência de mecanismos que façam com que os serviços possam ser descobertos por seus usuários.

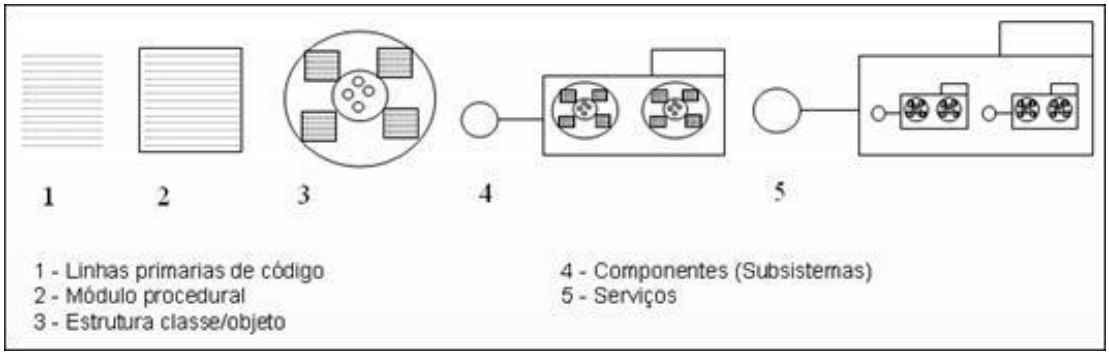
Soluções como *Service-Oriented Software Engineering (*SOSE), se destinam, na maioria das vezes, a superar limitações humanas, como lidar com grande quantidade de informações complexas. Para lidar com tal complexidade, a abstração é uma ferramenta chave, e está intimamente relacionada com o conceito de encapsulamento.

Focando nesse aspecto essencial, SERAFIM (2016), descreve os níveis de encapsulamento, conforme a Figura 2.3.

Baseando-se nesses níveis, é possível partir de simples linhas de código, e chegar ao SOSE, passando por *Component-Based Development* (CBD) (KOSKELA, 2007), e Orientação a Objetos (OO).

Mesmo considerando que essas abordagens foram concebidas em contextos de softwares diferentes dos de bordo, pode-se trazer tais conceitos para o tipo de produto embarcado, e com níveis de criticidade mais estreitos, fazendo uso de analogias e de técnicas conforme mostrado no Cap. 2.3.

Figura 2.3 – Esquema dos níveis de encapsulamento.



Fonte: SERAFIM (2016).

Na Figura 2.3, marcado como “4”, o componente encapsula objetos, protegendo-os com uma interface, de maneira contratual. O uso de contrato (*Design by Contract*), implica que cada uma das operações do componente é definida em termos de sua assinatura (seus tipos de argumentos de entrada e saída), e de suas pré-condições e pós-condições. Um componente oferece funcionalidades, por meio de interfaces bem definidas para o meio externo. Componentes de negócio são subsistemas projetados para serem reutilizáveis corporativamente, mas a reutilização não é a única motivação, a necessidade de lidar com modificações de maneira rápida e controlada, tornou-se essencial atualmente (SERAFIM, 2016).

Esta é a abordagem CBD, e deve ser vista como sistemas desenvolvidos pela integração de componentes (coleção de objetos), conforme um modelo de *design* CBD, que seguem os princípios*:*

(1) Componentes são independentes e não interferem uns com os outros;

(2) As implementações dos componentes são ocultas ou encapsuladas;

(3) A comunicação entre componentes se dá através de interfaces bem-definidas; e

(4) Componentes podem ser compostos de outros.

Diferente de objetos da Programação Orientada a Objetos (POO), um componente é um composto de objetos da POO, ou seja, é como um objeto voltado à especificação e não à implementação.

Vale reparar que uma integração de componentes não significa uma soma direta de funcionalidades, como $fC1+fC2+...+fCn$, pois, de acordo com os princípios da Engenharia de Sistemas, os componentes poderão ser combinados numa arquitetura de tal forma que, resulte também em funcionalidades que só podem existir devido à integração dos componentes, logo, a soma das funções sistêmicas de software poderá e, é recomendável, ser maior que a soma das funcionalidades individuais de cada componente.

Os serviços denotados em “5” na Figura 2.3, são parecidos com componentes em muitos aspectos. Contudo, ele é a representação lógica de uma tarefa de negócio (domínio) repetitiva e, que tem um resultado específico (por exemplo, verificação de crédito dos clientes, fornecer dados meteorológicos, e etc.). Uma abordagem promissora mostra, serviços sendo encontrados diretamente em processos de negócio. *Service-Oriented Architecture* (SOA) é um estilo arquitetônico, que suporta a orientação à serviços, com foco nos processos de negócios. Uma definição formal pode ser vista em *THE OPEN GROUP* (2017). O desenvolvimento baseado em componente, fornece uma base experimentada e testada para a implementação de SOA. Pode-se considerar que os serviços, no caso do SOA, encapsulam componentes (SERAFIM, 2016).

Vê-se, que os serviços são ou representam a evolução do pensamento, na tentativa de dividir um problema complexo e, ao mesmo tempo, focar na semântica do problema ou do negócio, fazendo com que os serviços resolvam os negócios e, mantenham seu código oculto para seus usuários. Uma forma usualmente praticada na implementação dos serviços, apesar de não ser

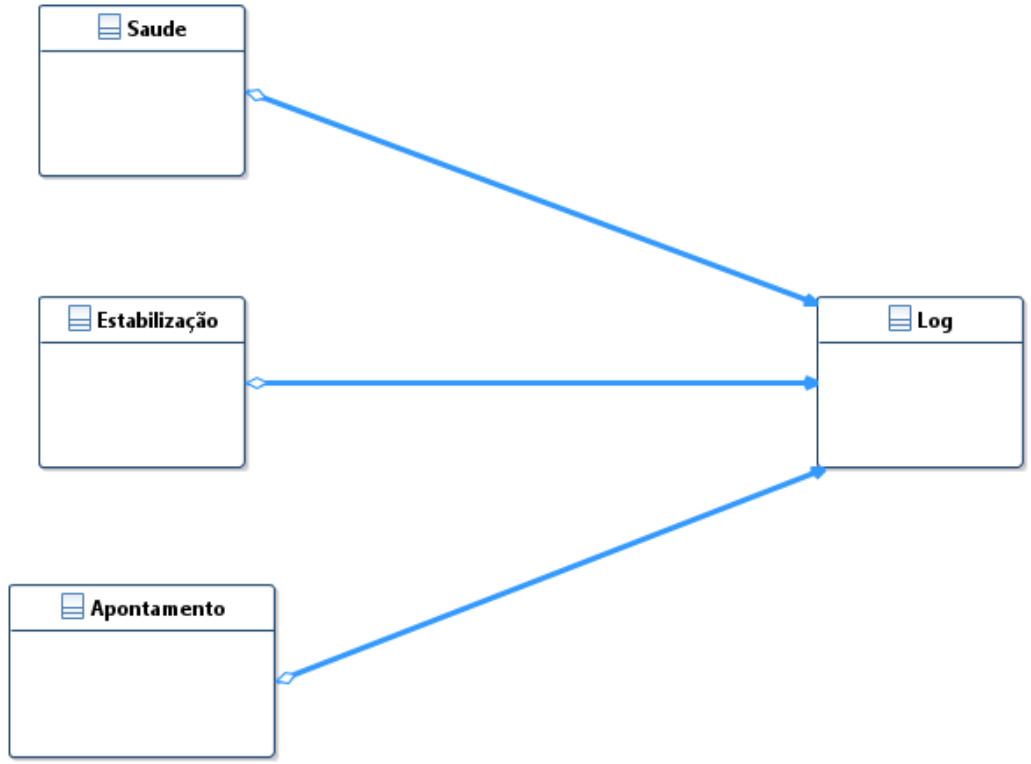
unanime entre os autores, e que fornece uma conotação diferenciada no uso do CDB, é o fato dos serviços, serem normalmente implementados como unidades de software fechadas, analogamente a um processo de um Sistema Operacional (SO), logo, quanto menor for a sua dependência com os outros serviços e, com o compartilhamento dos recursos computacionais como: memória, entradas / saídas e etc., melhor será o serviço. Pode-se idealizar os serviços, como aplicações que rodam em partições.

Muitas vezes esses serviços, dispensam o uso de aplicações como camada de controle, requerendo, quando necessário um arquivo de configurações. Os serviços podem ser autônomos, e instanciados no próprio *startup* do sistema operacional.

Neste contexto pode-se destacar outra evolução do pensamento que também aumenta o reuso, a Programação Orientada a Aspecto (POA). O paradigma POO, implementa a separação do código através de entidades únicas, por exemplo, a funcionalidade de *log* de dados, numa linguagem OO, é implementada em uma única classe, que é referenciada em todos os pontos onde é necessário fazer *logs* de dados. Como praticamente todo método necessita que alguns dados sejam registrados em *logs*, as chamadas à esta classe, são espalhadas por toda a aplicação. A Figura 2.4, mostra uma exemplificação.

Tipicamente, uma implementação de POA busca encapsular essas chamadas através de uma nova construção, chamada de "aspecto”.

Figura 2.4 – Chamadas da classe "Efetuar *LOG*".



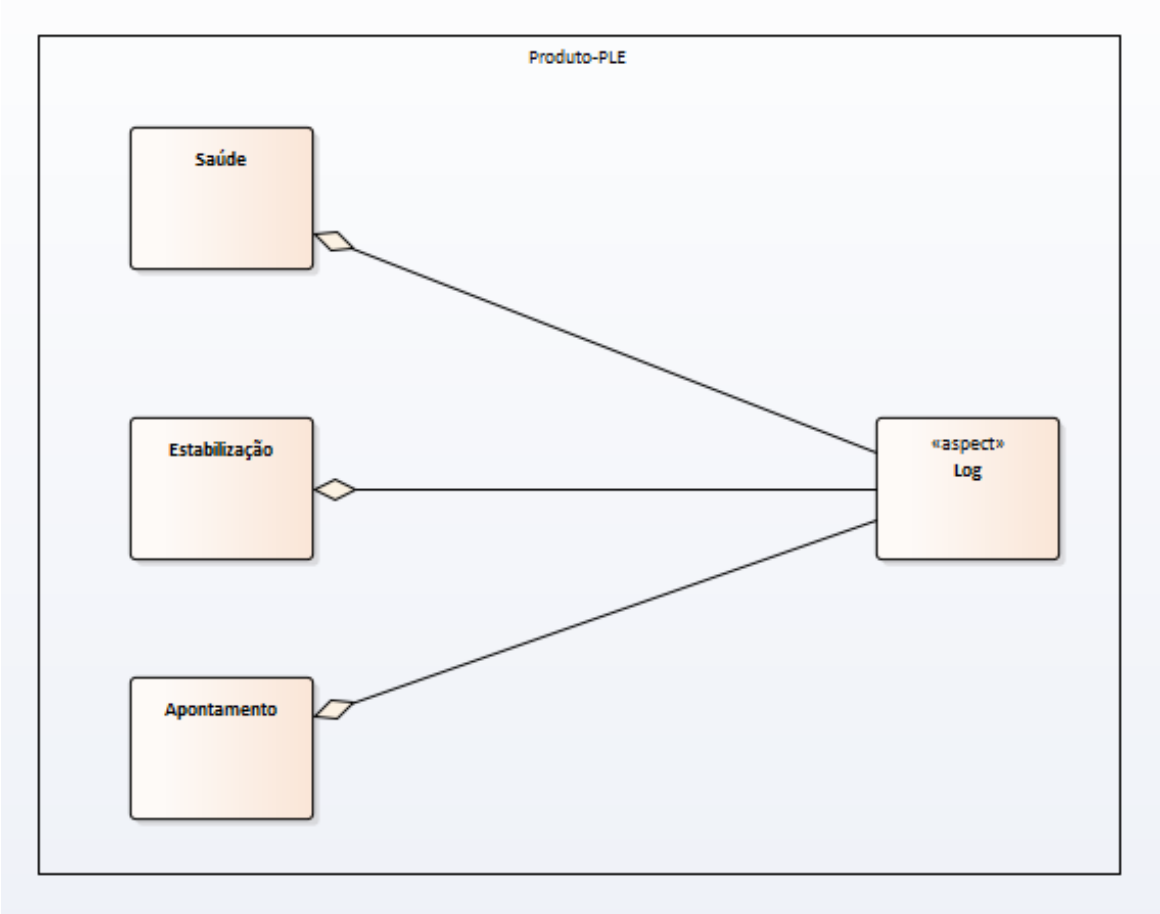
Fonte: Produção do autor.

De acordo com KOSKELA (2007), a abordagem POA afeta a modularidade OO, pois permite que métodos comuns, como por exemplo: tratativas de gerenciamento de falhas, *logs,* encriptar/decriptar informação, validar *checksum* e etc., fiquem espalhados de forma redundante em muitas classes, fazendo com que elas percam a sua coesão, ou até mesmo fiquem em classes dedicadas, e aumente o acoplamento com as classes que provêm estas funcionalidades, devido ao grande número de chamadas. Logo, para aqueles interesses (métodos) que não podem ser modularizados em classes dá-se o nome de interesses transversais.

Esta é considerada uma das principais fontes de complexidade em sistemas críticos de software. A POA, representa um paradigma de programação, e lida diretamente com aspectos de preocupações transversais, ao invés de módulos de código de software. O objetivo da POA, é remover aquilo que foge do negócio de uma classe, tornando possível expressar em aspectos, as

preocupações transversais e, em seguida, combinar estes aspectos uns com os outros e com o código executável, usando ferramentas automatizadas. Também, em POA a abstração e a decomposição, são usadas para quebrar um problema complexo, no entanto, em vez da decomposição funcional, a POA é baseada na decomposição aspectual (KOSKELA, 2007).

Figura 2.5 – Uso de aspectos no *design* dos Produtos-PLE[12].



Fonte: Produção do autor.

Usando a notação UML no *profile* segundo ALDAWUD (2005), para aspectos na Figura 2.5, o estereótipo *"<<aspect>>"* na classe "*log*", informa uma implementação transversal da mesma, evitando a diminuição da coesão das demais classes.

[12] Produtos-PLE são artefatos de software, onde foi empregado a técnica PLE e portanto possuem a característica de reutilização planejada.

Com isso conclui-se que a POA contribui com o reuso, tanto na forma da sua implementação quanto na separação das preocupações tranversais dentro dos aspectos. Pode-se entender que houve um aumento do reuso na separação dos interesses transversais, antes dentro de classes, mas agora em aspectos, onde poderão ser reaproveitados, se encapsulados ,e ao mesmo tempo sem causar danos à coesão e ao acoplamento entre as entidades do restante do sistema.

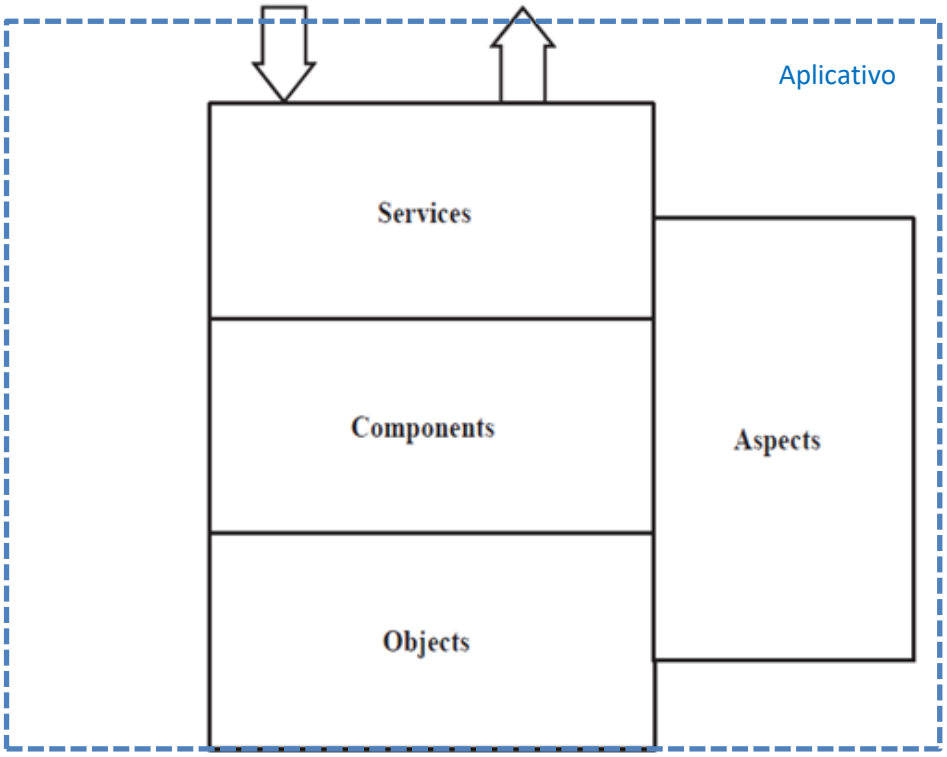
A Figura 2.6 eleva a abordagem de encapsulamento de serviços considerando também os aspectos. Desta forma pode-se entender que aplicativos como caixa-branca, são compostos por tais níveis de encapsulamentos, bem como, das preocupações transversais que podem afetar qualquer uma destas camadas.

Para exemplifica, pode-se pensar que o subsistema AOCS do satélite, possui um aplicativo com o serviço de apontamento[13], ou seja, algo genérico que poderia ser aplicado a todos os subsistemas AOCS. Este serviço tem a responsabilidade de prover o apontamento para um dado alvo. As Interfaces públicas para a recepção da informação de configuração do alvo, e a interface de saída das resultantes do apontamento, são algumas que o serviço tem.

Internamente os componentes deste aplicativo, têm a responsabilidade de resolver as diversas funcionalidades necessárias para prover tal serviço, como: Adquirir os dados dos sensores de atitude, processar os dados para construir uma estimativa da atitude corrente do satélite, computar o desvio da atitude corrente da nominal, e etc.

[13] Dependendo da arquitetura de software adotada, pode-se ter um aplicativo com mais de um serviço, mas, para exemplificação foi adotado deste modo.

Figura 2.6 – Encapsulamento de serviços considerando também os aspectos.



Fonte: Produção do autor.

Existem componentes que resolvem a camada de *drivers*[14], a fim de permitir que o sinal chegue até o componente eletrônico de processamento, responsável pelo controle do periférico, que de fato, efetua a aquisição dos dados dos sensores.

Nota-se que existem duas classificações de componentes de software: aqueles responsáveis pelas funcionalidades sistêmicas do serviço, e aqueles responsáveis pela interface com o hardware do produto.

Para fins de planejamento do reuso, considera-se que existem duas oportunidades de reuso.

[14] A camada de *Driver* ou *HAL* (*hardware abstract layer*) representa outros aplicativos que estão mais próximos do hardware e, portanto é responsável pela entrada e saída de sinais com componente eletrônico de processamento. Entretanto neste caso dependendo da arquitetura de projeto escolhida o sinal poderá ser encaminhado para uma API do sistema operacional e este se relacionará com o *HAL*.

Os componentes responsáveis pelas funcionalidades sistêmicas do serviço, ou de apoio à missão, que são comumente empregados em todos os subsistemas do mesmo tipo, neste caso, tais componentes poderiam ser empregados em todos os subsistemas AOCS, uma vez que, todos possuem as mesmas responsabilidades, e claro, para cada satélite, os atributos que definem tais funcionalidades, são configurados de acordo com sua missão específica.

A outra classe de componentes é a que está próxima do hardware, e é reutilizada quando a tecnologia de hardware adotada pelo sistema é comum, ou seja, o satélite adota controladores e periféricos comuns para seus subsistemas, por exemplo, todos os subsistemas AOCS de todos os satélites da mesma classe, possuem a mesma família de controladores, sensores e atuadores. A padronização e o desenvolvimento de fornecedores são comuns, mas neste caso eles são reforçados, pois contribuem também com o reuso.

Quando se desenvolve um aplicativo, ou um conjunto deles através de uma *feature*-PLE, é importante que se faça, dentro do contexto de serviços (SOSE). Assim, os envolvidos poderão se relacionar, usando as interfaces públicas e, se for necessário, adotar um produto de terceiros, os *stakeholders* do projeto-alvo deverão se relacionar através das interfaces que o fabricante disponibilizou. Desta forma, o projeto-alvo ficará livre para adotar produtos de terceiros ou proprietários com a *feature*-PLE.

A abordagem de serviços SOSE realizada com componentes CBD, favorece o reuso dos sistemas de software, pois reconhece que as mesmas capacidades do sistema são requeridas em diversas situações. Não existe valor no desenvolvimento das mesmas capacidades e, de fato, todo desenvolvedor em um processo padrão de desenvolvimento, quando deparado, em ter que fazer alguma rotina, busca em sua própria base de dados, na de um colega, ou ainda em um software livre por algum trabalho semelhante, ao invés, de seguir rigorosamente o processo.

Os serviços formalizam a abordagem para criar componentes de prateleira (*off-the-shelf*), para servir a uma arquitetura, e então, permitir que os engenheiros

desenvolvam sistemas complexos, incorporando componentes já desenvolvidos ou de mercado. Isto leva supostamente, a uma grande redução do desenvolvimento, do tempo de testes e dos custos, além da redução dos riscos, pois, uma vez validado e aplicado de forma equivalente no produto-alvo, os componentes deveriam se comportar de forma já conhecida e esperada. E ao longo da vida destes componentes, é esperada uma redução dos custos de manutenção, associados às atualizações dos sistemas complexos (VARNELL-SARJEANT, 2015).

2.3. Técnicas aplicáveis

Além das abordagens descritas anteriormente, é importante notar uma série de técnicas que podem ser aplicadas, devido a se aproximarem das abordagens de reuso e dos produtos de sistemas de software de bordo.

Trata-se de técnicas voltadas ao planejamento de software, já apontado nesse trabalho como o maior desafio para a prática do reuso de software, além de já serem amplamente utilizadas em sistemas que necessitam de tal confiança. Tais técnicas, refletem ferramentas adotadas atualmente nos diversos processos do ciclo de vida do software, onde, uma vez que a prática de reuso seja adotada, se deveria atentar com o propósito de mitigar os riscos que o reuso pode implicar.

Nos requisitos, pode-se citar:

- *Intent Specifications (IE*) (LEVESON, 2000);

  Trata-se, de uma abordagem de especificação *Human-Centered* (Figura 2.7), tanto na resolução de problemas quanto nas tarefas de evolução e desenvolvimento de software.

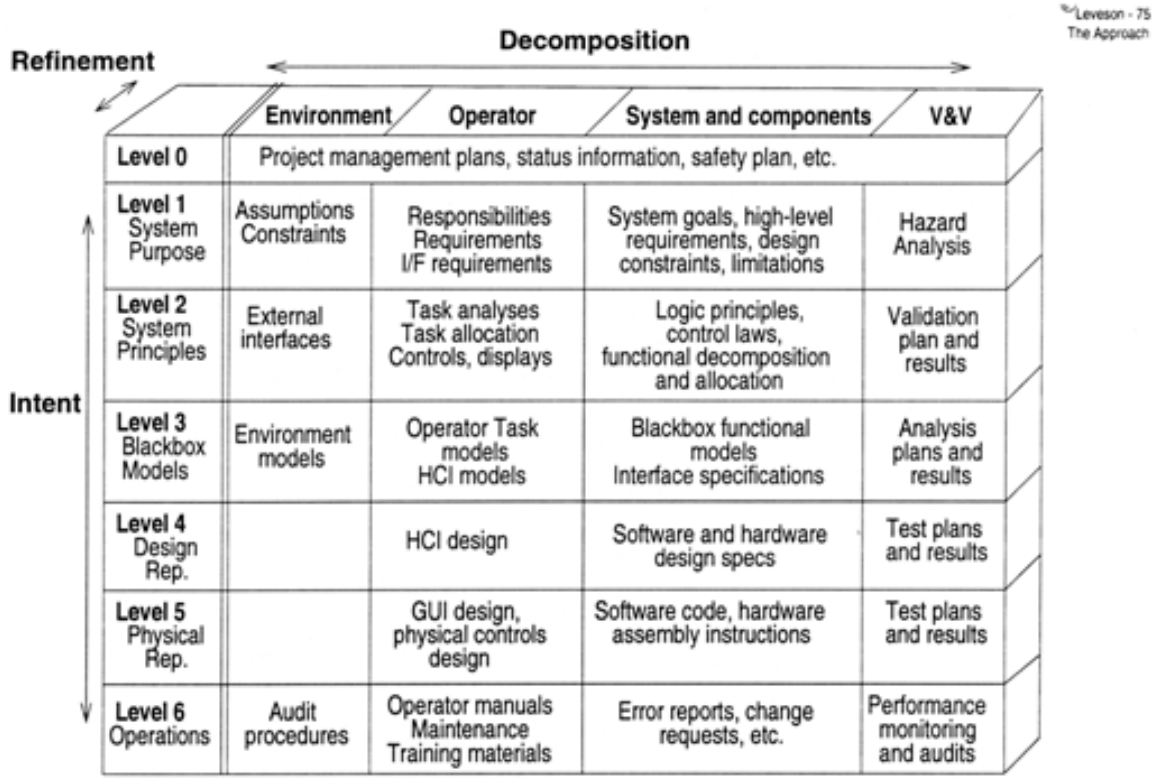
Na arquitetura, pode-se citar:

- *Service Component Architecture (SCA)* (BRUGALI, 2011);

  Um conceito arquitetônico para a criação de aplicações, que são construídas, através da montagem de componentes interoperáveis, e

acoplados livremente, cujas interações, são definidas em termos de ligações entre os serviços fornecidos e os serviços necessários.

Figura 2.7 – *Intent Specifications.*



Fonte: WEISS (2004).

- *Federated* (TAGAWA, 2011);

  As arquiteturas Federadas, são baseadas em unidades de processamento distribuídos através do sistema, onde uma rede de comunicação comum, serve de interface entre tais unidades.

- *Integrated Modular Avionics* (IMA) (TAGAWA, 2011);

  Uma unidade de processamento, hospeda inúmeras aplicações de diferentes níveis de criticidade, como em uma arquitetura centralizada, porém, cada aplicação é isolada por um robusto mecanismo de particionamento dado pelo Sistema Operacional.

- *Design Patterns* (FREEMAN, 2009);

Modelos eficazes para a resolução de problemas comuns em software.

- Programação Orientada a Aspecto (KOSKELA, 2007);

No suporte, pode-se citar:

- *Verification and validation* (V&V);

  Testes de caixa branca e de caixa preta (MYERS, 2004).

- A certificação e o registro, normalmente são realizados por autoridades e órgãos competentes como, a *Federal Aviation Administration* (FAA) nos EUA; e no Brasil, projetos aviônicos civis são realizados, pela Agência Nacional de Aviação Civil; já, os projetos militares são realizados pelo Instituto de Fomento e Coordenação Industrial, e o Departamento de Controle do Espaço Aéreo.

- Níveis de qualidade de acordo com TRL/RRL (DOWNS, 2010); e

  São ferramentas que ajudam a minimizar os problemas potenciais, enfrentados na ausência de um instrumento, para medir a maturidade de reutilização de software na ESA e NASA.

- Gestão da Configuração (RTCA, 2011);

  Garantir as *baselines* entre os artefatos gerados, a rastreabilidade dos *problem reports*, e as *change requests*.

2.3.1. V&V, Níveis de Prontidão e Certificação

Esta pesquisa não pretende explorar o tema de V&V aplicada na questão dos componentes reutilizáveis, pois entende-se que este tema, de fato é uma oportunidade, porém, está fora do escopo deste trabalho, no qual aborda questões relacionadas com o seu desenvolvimento, e principalmente focada aos requisitos, arquitetura e *design*.

Contudo, se reconhece que o tema de V&V deveria ser melhor explorado, pois sabe –se que, com o aumento do reuso de um mesmo produto reutilizável em

contextos diversos, e somado às suas manutenções, o produto tende a aumentar sua confiabilidade, porém, quando um determinado teste de integração ou de regressão poderia ser dispensado?

Sabe-se que com as interfaces bem definidas (*Design by Contract*) isso age positivamente na diminuição dos testes de integração e regressão, uma vez que o mapa de variações (valoração dos atributos X caminhos críticos) diminuem, contudo, é um tema que merece maiores detalhamentos, estando fora do escopo desta pesquisa, e será deixado como recomendações para trabalhos futuros.

Neste momento não há outra alternativa senão, manter a tradicional preocupação com V&V e testes para os produtos reutilizáveis, ou seja, deve-se seguir as práticas conhecidas de V&V com seus métodos. Isto inclui análises de caixa-preta e branca (MYERS, 2004), dentro de um processo de V&V adotado pela companhia.

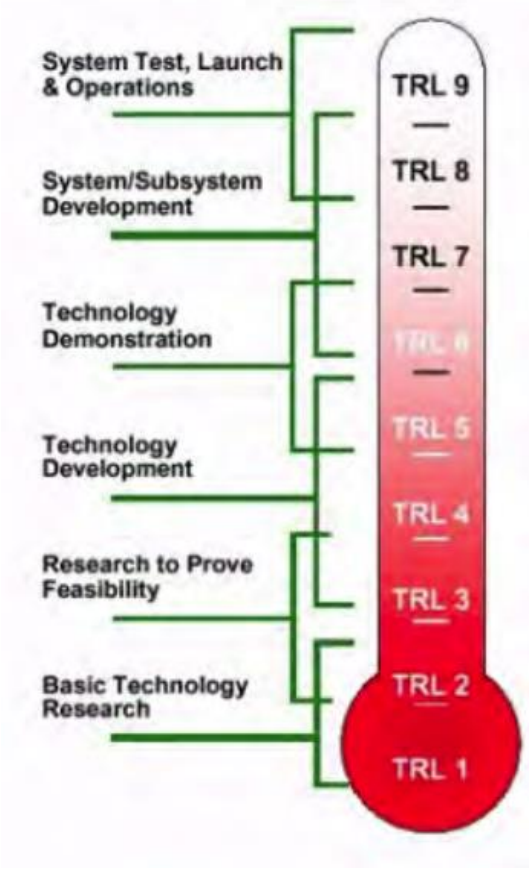
Além do V&V em conformidade com as agências internacionais de referência, bem como com o DoD dos EUA o fizeram, a adoção de níveis de prontidão (ou de maturidade) das tecnologias aplicadas nos produtos, poderia ser analogamente traduzida aos Produtos-PLE.

Logo TRL, RRL (DOWNS, 2010) ou *Tecnology Readiness Assessment* (TRA) (DRD, 2009), poderiam classificar, de forma análoga, cada Produto-PLE do repositório, e indicar, desde os princípios básicos observados e relatados nos níveis mais baixos de prontidão tecnológica, como uma pesquisa científica, que começou a ser traduzida em pesquisa e desenvolvimento aplicado. Até o nível de maturidade maior, onde o sistema real foi comprovado através de operações de missão bem sucedidas, e que houve a aplicação real da tecnologia em sua forma final e em condições de missão, como as encontradas nos testes operacionais e na avaliação (V&V).

A informação dos diversos cenários operacionais já implantados, é considerada de grande valia para o Reuso e aplicável aos Produtos-PLE, pois a análise do

comportamento, nos diversos cenários operacionais é altamente recomendado, segundo LEVESON (1993).

Figura 2.8 – Níveis de prontidão da NASA.



Fonte: NASA (2014).

A fim de trazer um melhor aproveitamento dos componentes reutilizáveis, deve-se focar na certificação de tais componentes, e isto envolve melhorias na economia e na qualidade do reuso. Atualmente, existem conceitos que mostram diretivas para que um determinado componente de software, seja certificado, como o *Reusable Software Components* (RSC) (KHANNA, 2005) da FAA (2004), e em utilizações subsequentes, dispense verificações e testes sobre o seu funcionamento interno, isto inclui também, a possibilidade de uma certificação incremental em arquiteturas IMA, conforme mostra TAGAWA (2011), dentro de certas condições.

Com isso, produtos reutilizáveis, devem ser tratados como componentes de software, a fim de passar por tais processos de certificação. Desta forma, o reuso traz maior confiança ao projeto, além da redução do escopo, e da redução do tempo do processo de certificação dos projetos, uma vez que, apenas os projetos-alvo serão o objeto.

2.4. Processos

Os processos são fundamentais para que uma equipe de engenharia tenha sucesso na aplicação do reuso. Pode-se citar: processo que visa a alta confiabilidade para sistemas de software embarcados; processo que visa a desburocratização, trazendo atividades segundo os princípios do manifesto ágil, através de um pensamento orientado à prototipação iterativa e incremental; processo aplicável aos sistemas espaciais de bordo, escritos em meio acadêmico e que já trazem as características da reusabilidade, bem como, processo clássico que considera o reuso. Eles são mostrados abaixo e são considerados específicos para o tema do Reuso:

- DO-178C (RTCA, 2011);

  Processo de desenvolvimento de software de alta confiabilidade, aplicado em indústrias aviônicas e de defesa.

- Harmony-ESW (DOUGLASS, 2009);

  Processo de desenvolvimento de software ágil para sistemas embarcados de alta confiabilidade.

- Software Process for Adaptability, safe REuse and Variability - SPAREv (DOS SANTOS, 2008); e

  Processo de desenvolvimento de software aplicáveis em sistemas de bordo que traz atividades de reuso.

- Processo de Reuso de Software do JACOBSON (1997).

Processo clássico de desenvolvimento de software com atividades de reuso.

Tais processos são considerados técnicos e específicos para este tema, pois é importante antes, haver uma orientação mais generalista, que possa guiar a aplicação das técnicas, e das atividades das pessoas que os processos técnicos demandam, em função de um nível de capacidade, ou seja, uma representação que permita à organização, escolher uma determinada área de processo (ou grupo de áreas de processo), que melhor se adeque à estratégia da companhia ou instituição, de acordo o nível de maturidade e os prazos em que se deseja chegar. A flexibilidade de um Reuso planejado, inicia-se desde a escolha de seus processos.

Níveis de capacidade para caracterizar a melhoria associada a uma dada área de processos são úteis. Sendo assim, a capacidade é medida por áreas de processos separadamente, onde é possível ter áreas de processos com baixo nível de atendimento, e, outras áreas de processos com alto nível, variando de acordo com os interesses da empresa.

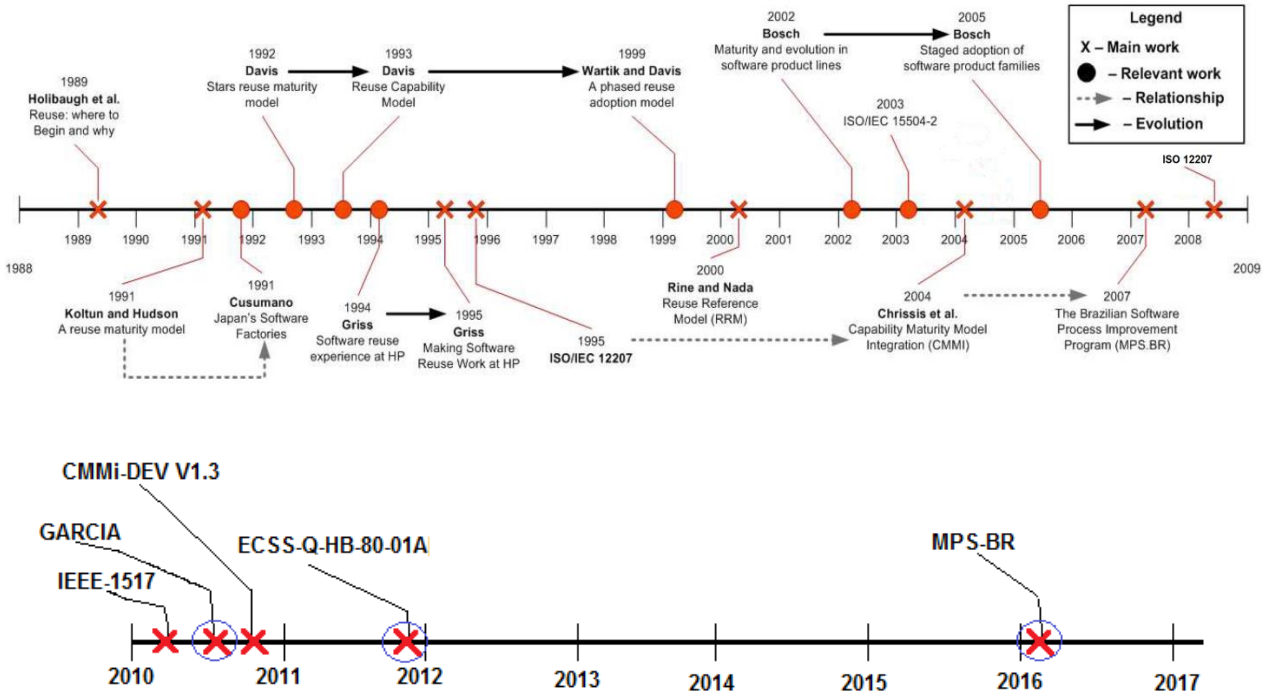
Logo, será investigado nas seções posteriores, processos considerando modelos de capacidades, porém, orientados ao reuso do software, a fim de explorar, quais as atividades e as técnicas são aplicáveis em função, de um nível de capacidade que a instituição está interessada em adquirir. Como a pesquisa está orientada à área espacial, também trará preocupações da ECSS, quanto ao tema do Reuso.

2.4.1. Aderência da Proposta de Reuso aos processos da Engenharia de Software

Diversos Trabalhos (estudos ou pesquisas), Padrões e Recomendações (TPRs), tem surgido, e orientado os profissionais da Engenharia de Software. A linha do tempo desde 1988 à 2017, sumarizada, na Figura 2.9, mostra alguns TPRs, que abordam práticas de reuso de software, incluindo modelos de maturidade. As linhas pontilhadas, mostram trabalhos, que foram baseados em

trabalhos anteriores. As setas, indicam que os trabalhos foram estendidos por outros. Os TPRs principais, estão marcados com um “X”.

Figura 2.9 – Linha do Tempo de pesquisas sobre modelos de maturidade que incluem o reuso de software.



Fonte: Adaptado de GARCIA (2010).

Vale ressaltar que Garcia (2010), trabalhou na Figura 2.9 até 2009, e esta pesquisa a estendeu até 2017.

As principais referências mundiais, também são sumarizadas na Figura 2.9, quanto ao assunto de processos do software. Dentre os TPRs mostrados, pode-se, destacar três que reúnem os conteúdos, e levam em consideração todo o restante. Estes são MR-MPS-SW (2016), ECSS-Q-HB-80-01A (2011), e GARCIA (2010). Estes estão sinalizados com “X” e circulados.

No trabalho de GARCIA (2010), observa-se, uma série de processos estritamente ligados ao reuso de software, e com interfaces ou dependências da Std 12207 (2008).

O trabalho, está fundamentado em diversos TPRs, e entre eles pode-se destacar como principais: CMMi-DEV V1.2, ISO12207, ISO1517:1999 e MPS.BR:2007. Ele organiza os processos em níveis de maturidade, a fim de, poder ser aplicado a qualquer empresa que possui interesse no reuso, e assim, gradativamente as exigências requeridas aumentam conforme o nível de maturidade desejado pela organização, também aumenta.

O MPS.BR foi projetado, e, mantido pela Associação para Promoção da Excelência do Software Brasileiro. Ele está baseado na ISO/IEC 12207:2008 e CMMI-DEV: 2010, conforme mostra a Figura 2.10, e, recentemente, foi publicada uma nova versão do seu Modelo de Referência MR-MPS-SW: 2016, o qual trata do reuso do software explicitamente em dois processos:

- Gerência de Reutilização (no nível de maturidade “E” do MPS.BR); e o
- Desenvolvimento para Reutilização (no nível de maturidade “C” do MPS.BR).

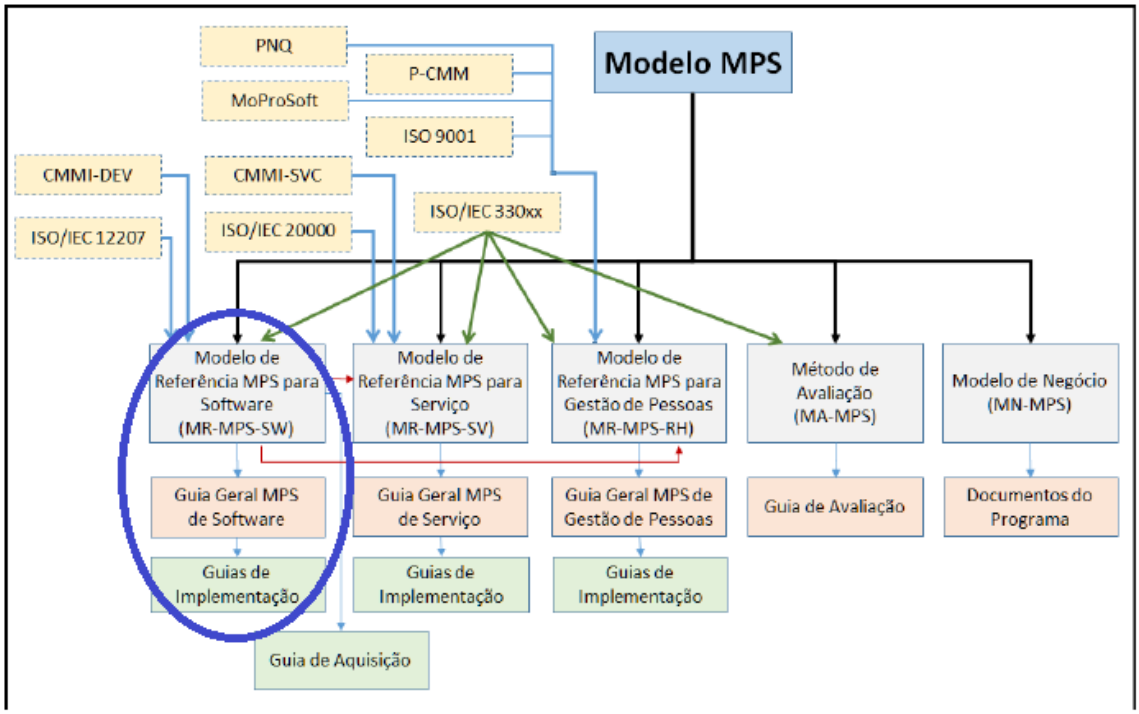
Estes processos, não sofreram alteração quando comparados ao MPS.BR:2007 já considerados por GARCIA (2010).

Na ECSS-Q-HB-80-01A (2011), *Space product assurance Reuse of existing software,* encontra-se pontos de atenção ligados ao reuso de softwares, nos sistemas de software espacial. Tais pontos servem como advertências para o reuso, propondo atividades de mitigação de risco quando da aplicação do reuso.

A ECSS-Q-HB-80-01A (2011), não está preocupada em como construir um software reutilizável, e sim, com as ações que deveriam ser executadas no projeto-alvo espacial quanto a decisão de “uso” de um software existente.

Logo, é esperado que tais considerações indiquem ações, tanto para os processos do ciclo de vida da biblioteca, quanto para os processos de desenvolvimento do projeto-alvo espacial.

Figura 2.10 – Componentes do Modelo MPS.BR 2016.



Fonte: MR-MPS-SW (2016).

Uma vez que o modelo GARCIA (2010), encontra-se voltado estritamente ao reuso e ainda traz uma visão, de forma atualizada, considerando a manutenção dos processos de reuso do MPS.BR:2007 no MR-MPS-SW:2016, é saudável a união destes com as contribuições relativas ao reuso da ECSS-Q-HB-80-01A (2011), a fim de se ter um conjunto de processos completos[15] e voltados para o campo espacial. Esta pesquisa se adequa, em parte, aos processos combinados de GARCIA (2010), e ECSS-Q-HB-80-01A (2011), e aqui denotados por “GARCIA+ECSS”, como forma de realização das atividades lá propostas.

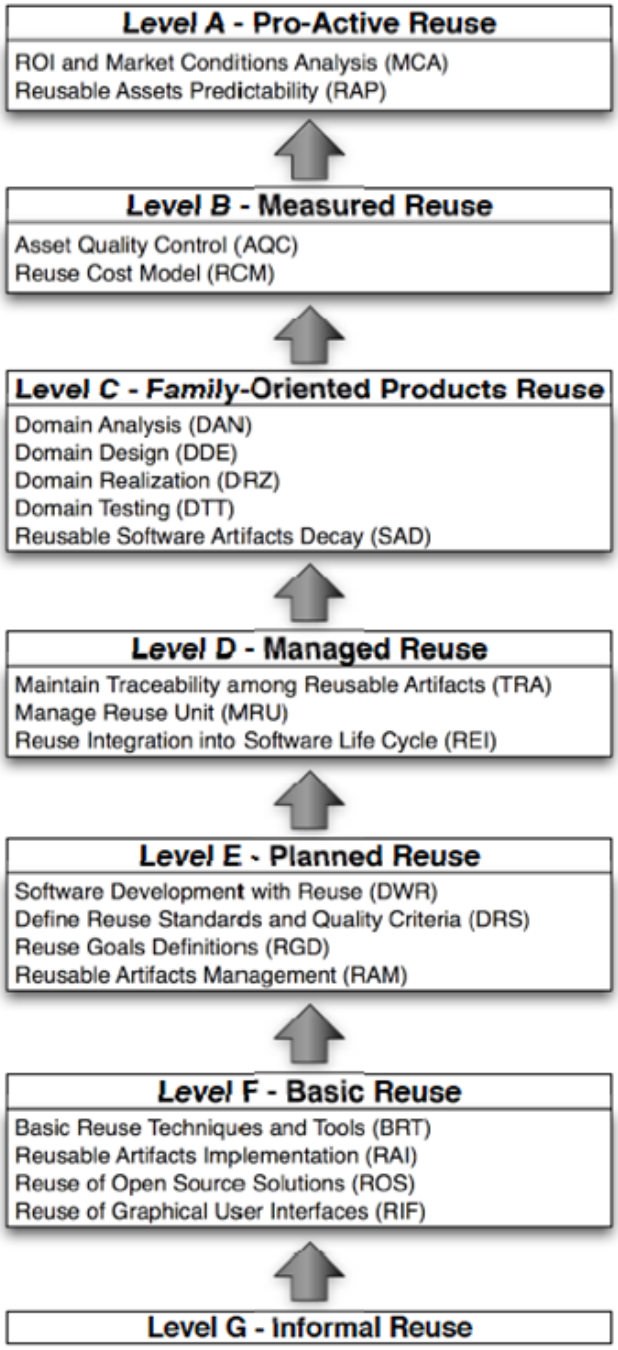
[15] Importante notar, que o IEEE 1517:2010 “IEEE *Standard for Information Technology – System and Software Life Cycle Processes – Reuse Processes*” serve como uma extensão à IEEE Std 12207:2008 somando tarefas e resultados aos processos e atividades definidas no padrão, com o foco no reuso, uma vez que a versão anterior, IEEE 1517:1999, encontrava-se defasada. Entretanto GARCIA (2010), apesar de ter considerado a IEEE 1517:1999 não fez uso da IEEE 1517:2010, porém considerou os mesmos processos da 12207:2008, além de fazê-los estruturados em um modelo de maturidade que propicia a escolha à companhia.

2.4.2. Níveis de capacidades segundo GARCIA+ECSS

Os processos de GARCIA (2010), são mostrados na Figura 2.11, e estão de acordo com os níveis de maturidade.

É possível para uma instituição alinhar tais processos com esta pesquisa, entretanto, não é esperado que todos os processos sejam cobertos por ela, pois isto, a levaria a se afastar do tema abordando contextos particulares como: reuso de soluções de código-aberto *(Reuse of Open Source Solutions - ROS),* e, reuso de interfaces gráficas *(Reuse of Graphical User Interfaces - RIF)* (Figura 2.11). Tratamento semelhante, pode ser aplicado às questões de padronização, decisões e documentações, como: as definições dos objetivos de reuso *(Reuse Goals Definitions - RGD)*; definições das responsabilidades do comitê de reuso *(Manage Reuse Unit - MRU)*, e até mesmo, a integração dele, dentro do ciclo de vida do software, já adotado pela companhia *(Reuse Integration into Software Life Cycle - REI)*, inclusive discutido na ECSS-Q-HB-80-01A (2011).

Figura 2.11 – Modelo de Processos de acordo com GARCIA (2010).



Fonte: GARCIA (2010).

Um exemplo, é o processo de Gerenciamento de Artefatos Reusáveis *(Reusable Artifacts Management - RAM)* (Figura 2.11), onde um sistema de repositório aliado à Gestão de Configuração é usado para armazenar, pesquisar, buscar e aprimorar os Produtos-PLE, conforme sugere o resultado

RAM2[16] (atividade de decomposição de RAM), porém, os resultados RAM1[16] e RAM3[16], são detalhamentos, e, dependem muito mais dos processos da companhia, estando fora do contexto desta pesquisa.

Tabela 2.1 - Aderência da pesquisa aos processos GARCIA+ECSS.

Processos (GARCIA, 2010)		(ECSS, 2011)	Adequação à Pesquisa
F	RIF		NA
	ROS		NA
	RAI	x	x
	BRT	x	x
E	RAM		x
	RGD		NA
	DRS		x
	DWR		x
D	REI	x	x
	MRU		NA
	TRA		x
C	SAD		x
	DTT	x	x
	DRZ		x
	DDE		x
	DAN	x	x
B	RCM		x
	AQC		NA
A	RAP		x
	MCA		NA

Fonte: Produção do autor.

[16] *RAM1: A reusable artifacts management process is implemented, to register and manage (analyze, control and audit) the reusable artifacts of the software company in a centralized way* (GARCIA (2010)*;*

RAM2: A library/repository system is used to manage (store, search, retrieval and evolve) the different versions of the reusable artifacts GARCIA (2010);

RAM3: An asset classification scheme is used to organize the reusable artifacts, to facilitate management, storage, search and retrieval. This classification scheme is constantly reviewed and updated to reflect the evolution of the registered artifacts (GARCIA (2010).

São mostrados na Tabela 2.1, os processos do modelo GARCIA (2010), bem como, os que devem ser influenciados pela ECSS-Q-HB-80-01A (2011), e, em quais esta pesquisa é aplicável mesmo de formal parcial, “x”, ou não, “NA”. Assim nota-se, que em 70% dos processos, sejam de forma total ou parcial, é possível considerar a adequação desta pesquisa.

Deixa-se para trabalhos futuros, um melhor detalhamento sobre a adequação da prosposta aos processos GARCIA+ECSS, mas, ao mesmo tempo, sem comprometer o conteúdo desta pesquisa, uma vez que, as referências e a tabela de relacionamento foram dadas, além, das descrições dos estágios do ciclo de vida conforme será visto no Cap. 3 (Figura 3.2).

Este tópico teve o objetivo, de mostrar a grande aderência desta pesquisa aos processos de maior referência na Engenharia de Software, voltados ao tema do reuso e em função de capacidades. Entretanto, não foi abordado nada sobre como as instituições e companhias, podem se organizar para praticar tais processos, pois existem formas diferentes de organizar os times.

Primeiramente, em função das preocupações do time - time que desenvolve produtos reutilizáveis, não deveria focar em um único projeto. Logo, pode-se trabalhar com times diferentes, um para o projeto-alvo e outro para o desenvolvimento dos produtos reutilizáveis. Segundo, em função dos níveis de abstração que o projeto demanda – sabe-se que a preocupação com as demandas relacionadas à missão, são diferentes das relacionadas ao *design*, onde o foco, está nas plataformas do hardware. Assim, é possível pensar, em equipes que desenvolvem as aplicações e, outras que fazem os *drivers*, por exemplo.

Logo, em seguida vê-se, algumas formas de como organizar as equipes das companhias dentro do estágio de Desenvolvimento, a fim de, usufruir dos repositórios de reuso.

2.4.3. Organização das equipes: Projeto-alvo e Projeto-PLE

Quando o software é planejado de tal forma que minimize seus acoplamentos, sempre será possível melhorar a produtividade, pois, equipes distintas poderão trabalhar ao mesmo tempo em partes do projeto de software, devendo respeitar suas devidas interfaces.

O desenvolvimento dos *drivers,* é uma atividade que permite o desacoplamento total da aplicação, pois o foco, é construir as funções que operam um determinado *device* (componente eletrônico) de um microcontrolador por exemplo, e portanto, é independente da aplicação.

As funções que apoiam a missão, estão relacionadas à capacidade do projeto-alvo em utilizar as mesmas funções sistêmicas, como por exemplo, as funcionalidades de alguns Módulos são comuns a vários outros Módulos de um mesmo tipo, ou então, as similaridades que os serviços SOIS do CCSDS poderiam ser implantados em vários satélites, logo, conclui-se um nível de reuso destas funções.

Desta forma, tanto o desenvolvimento dos *drivers*, quanto o das funções que apoiam a missão, poderão ocorrer independentemente, e dentro de cada esfera: '*drivers'* ou 'funções de apoio à missão'. Assim, pode-se aplicar o desenvolvimento paralelo, indicando tanto a separação dos times do projeto-alvo, em relação aos das bibliotecas, quanto aos times de *drivers, em relação aos das* 'funções de apoio à missão', incluindo o paralelismo simultâneo, ocorrendo para diferentes Produtos-PLE.

2.4.4. Diferenças dos processos - Produtos-PLE e Projetos-alvo

A proposta, é que, o uso dos repositórios seja gradualmente aumentado conforme os projetos-alvos são desenvolvidos, inclusive, nunca precisará de dedicação exclusiva aos repositórios, se assim os processos de desenvolvimento da companhia determinar. Ou seja, é possível que as bibliotecas existam mesmo não trabalhando diretamente nelas, em um determinado momento, o Potencial de Reuso (PR) ocorrerá de qualquer forma.

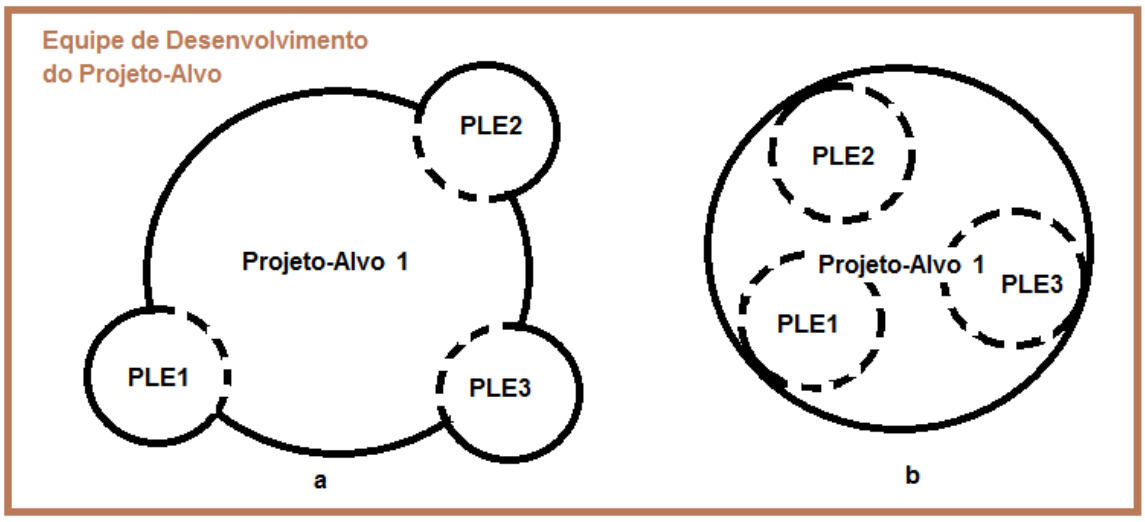
Para explicitar as diferenças dos processos entre os Produtos-PLE e os Projetos-alvo, considere o cenário a seguir:

2.4.4.1. Simulação de um cenário de reuso

Uma companhia está iniciando o reuso e portanto, ainda não conta com uma quantidade de Produtos-PLE no repositório, mas há, a intenção de poupar esforços. A escolha é fazer o desenvolvimento do Projeto-alvo, de forma que os *drivers* e as funções intermediárias, sejam de alguma forma segregadas como componentes, ou seja, aplicando-se uma visão funcional e alocando-se tais funcionalidades nestas estruturas.

Por exemplo a Figura 2.12, mostra, hipoteticamente, o cenário de um projeto-alvo-1 sendo desenvolvido de duas formas “a” e “b”, mas ambas indicam, que os componentes segregados PLE1, PLE2 e PLE3 são os primeiros do repositório desta companhia.

Figura 2.12 – Exemplo de cenários inciais para reuso via PLEs.



Fonte: Produção do autor.

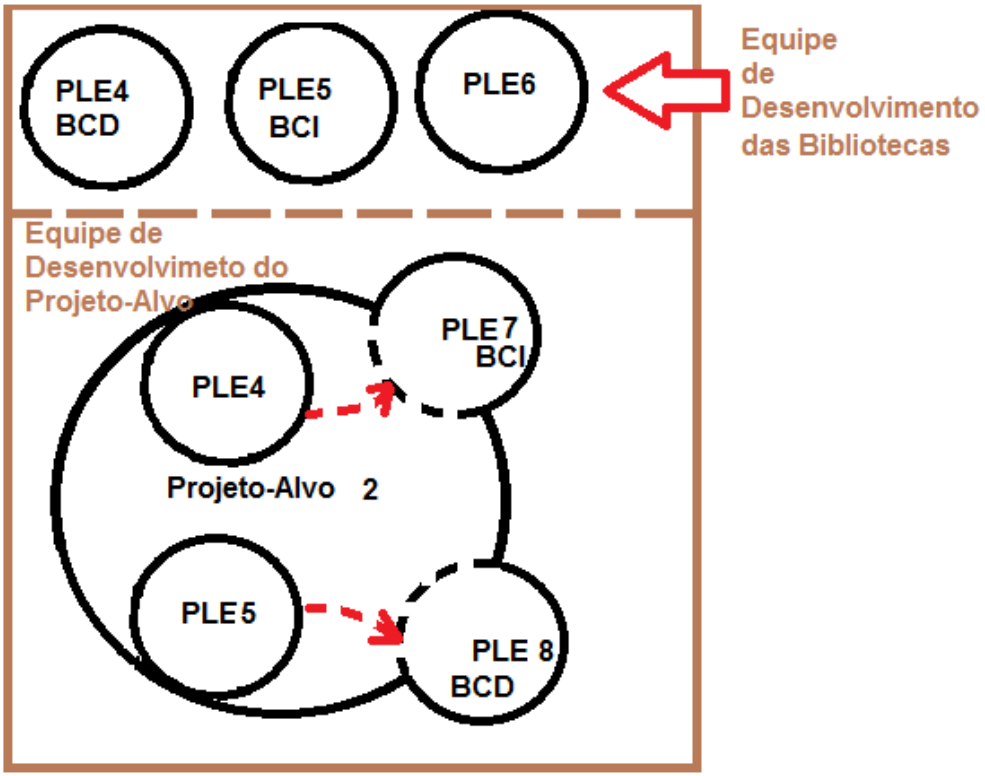
Observa-se que quando repetido este processo, ou seja, a própria equipe de desenvolvimento do projeto-alvo segrega os componentes reutilizáveis, mais Produtos-PLE são gerados, e uma vez estando numa linha de produtos similares, ou que compartilham de similaridades, seja através, do

desenvolvimento de fornecedores para os componentes eletrônicos de processamento, ou através das similaridades funcionais de missão, maior será o PR.

É possível desenvolver os produtos reutilizáveis de forma independente do projeto-alvo. De acordo com a Figura 2.13, o PLE4 foi desenvolvido por uma equipe focada em reuso, neste caso, trata-se de um *driver* que pertence à biblioteca da camada de *drivers* (BCD).

Por exemplo um *Analog-Digital Converter (ADC)* foi desenvolvido, e encapsulado para um determinado microcontrolador no componente PLE4, pela Equipe de Desenvolvimento das Bibliotecas, mas, em um momento posterior, durante o desenvolvimento das funções de apoio à missão para um termopar do projeto-alvo-2, foi encapsulado tais funções no PLE7 e então composto pelo PLE4 na equipe de desenvolvimento do projeto-alvo 2.

Figura 2.13 – Desenvolvimento.



Fonte: Produção do autor.

A mesma equipe de reuso encapsulou serviços SOIS, através do PLE5 na Biblioteca da Camada Intermediária (BCI), que serve como repositório de funções de apoio à missão, e, durante o desenvolvimento do projeto-alvo-2 (Figura 2.13), foi composta ou agregada aos demais *drivers* da BCD deste microcontrolador em PLE8, pela equipe de projeto.

Com isso mostra-se a dinâmica entre as equipes de reuso e do projeto, bem como a possibilidade dos componentes PLE, serem desenvolvidos paralelamente aos projetos-alvo, e, ainda usados como base de outros componentes PLE, desenvolvidos durante o projeto-alvo.

Voltando à análise das diferenças, entre os processos de Produtos-PLE e Projetos-alvo, após o entendimento do cenário dado anteriormente, percebe-se que, devido à proposta de reuso desta pesquisa estar fundamentada sobre os 3 pilares de conhecimentos já explicados anteriormete no Cap. 2, isto irá acarretar, em duas formas de relacionamentos entre os repositórios de reuso, e o desenvolvimento do projeto-alvo, conforme “a” ou “b” indicado na Figura 2.12**.**

No Cenário “a” as atividades relacionadas com a prática de reuso, de acordo com os pilares já informados, agregam atividades quando comparadas às normas da companhia, segundo seus processos de desenvolvimento do software, enquanto em “b”, são práticas já adotadas pela companhia, não do modo exatamente como apontado neste trabalho, mas sim por meio de práticas conhecidas e adotadas de alguma forma dentro dos seus processos de desenvolvimento.

Com isso nota-se em “a” uma nítida diferença de área, e aqui significa escopo, quando comparado a “b”. Em “a”, é esperado que os processos de desenvolvimento de software e claramente, as práticas de reutilização como: o uso de desenvolvimento baseado em componentes; padrões de *design;* entre outros, sejam esporadicamente utilizados, e ignorados ou desencorajados por gestão. Estas práticas, podem ser realizadas como uma iniciativa individual (objetivo pessoal; de acordo com as restrições de tempo). Os custos e os benefícios da reutilização são desconhecidos (GARCIA, 2010).

Logo, em “a”, o projeto é inflado ao menos nos Produtos-PLE, que são concebidos mediante às práticas indicadas no Cap. 2, em que se pode diferir no restante do projeto-alvo1 (Figura 2.12). Recomenda-se, que as iniciativas mostradas no Cap. 2.1, sirvam ao menos, para encorajar os dirigentes nesse início, principalmente, as companhias em que a confiabilidade do software é exigida.

Os modelos de maturidade são níveis construídos uns sobre os outros, e indicam, que um conjunto de atividades para desenvolver software estão unidas e gerenciadas para consistentemente se alcançar alta qualidade nos resultados, dentro das restrições de custo e prazos.

Segundo Lim (GARCIA, 2010), no final dos anos 90, afirmou que pesquisadores e praticantes do reuso não acordaram sobre o relacionamento, entre o reuso e o *Capability Maturity Model Integration* (CMMi). De fato, o reuso pode ser praticado em múltiplos níveis do CMMi, entretanto, os benefícios do reuso aumentam, progressivamente, conforme ele é praticado nos níveis mais altos do CMMi.

A prática de reutilização nos níveis mais baixos pode não resultar em um benefício líquido. Por exemplo, uma organização que realiza a reutilização nos níveis inferiores, com um processo de desenvolvimento de software propenso a defeitos, irá propagar, ativos reutilizáveis com defeitos. Alguns benefícios podem ser experimentados, pois, como não é necessário criar funcionalidades similares (que podem estar igualmente ocultas por defeitos), a correção de um defeito ocorreria apenas uma vez.

No entanto esse benefício, é compensado pela probabilidade do processo de reutilização em si, ser igualmente não sistemático e, portanto, as correções podem não ser feitas ou entregues corretamente segundo Lim (GARCIA, 2010).

Logo, dependendo do nível de maturidade, onde os processos da companhia se encontram para se obter os benefícios de um reuso mínimo, ou acima do nível G, isto pode causar impactos na forma de se trabalhar com o software.

Para finalizar seguem relatos de grandes companhias que apostaram no CMMi, divulgando seus sucessos (CMMI INSTITUTE, 2015) : (1) "*Tufts Associated Health Plans decreased software defects by 25%*"; (2) "General *Dynamics reduced maintenance staff costs by* 64% *while doubling the size of the organization*"; (3) "*Siemens increased their customer satisfaction index an average of 42% in three technical areas*" e (4) "*JPMorgan Chase reduced average slippage of project delivery dates by 70-80%".*

As companhias em geral, que implataram o CMMi, são apontadas na Figura 2.14. Um círculo vermelho revela empresas e instituições do ramo Espacial e de Defesa.

Figura 2.14 – Empresas que implantaram o CMMi.

Fonte: CMMI INSTITUTE (2015).

2.5. Uma abordagem sobre a economia através do Reuso

Um tópico a ser considerado para o reuso é quanto sua quantificação e métricas que refletem seus ganhos.

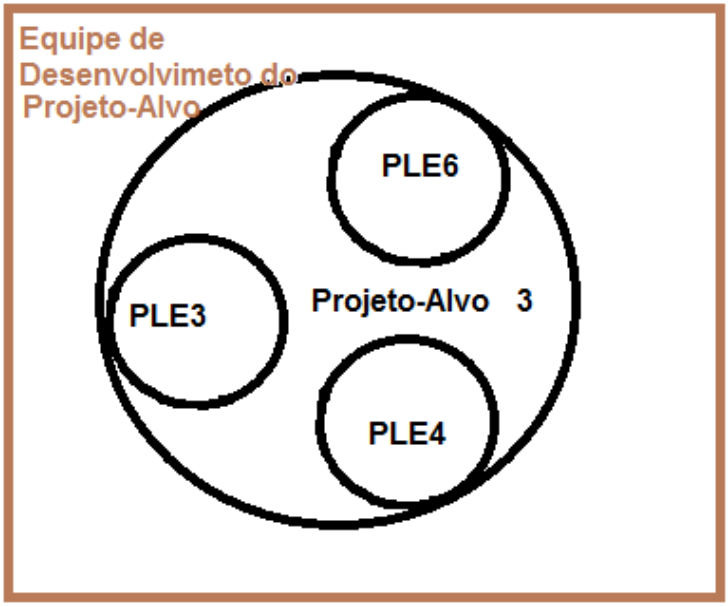
Tanto os conceitos de produtividade, quanto de viabilidade estão fundados na economia em tamanho que o reuso oferece. Ou seja, são blocos de software prontos, que estão sendo usados em um novo produto, logo, como já estão prontos, não foi necessário desenvolvê-los no produto, e a economia torna-se evidente.

Detalhamentos sobre as métricas de tamanhos e produtividade, podem ser melhor entendidas em Bauer (2009), e De Almeida (2007), sendo a viabilidade uma estimativa desses valores baseados em dados históricos.

Logo, o objetivo deste tópico é formalizar o conceito da enconomia, introduzir conceitos de Potêncial de Reuso, Índice de Reuso, e Ganho não Econômico de Reuso, usando, da continuação do cenário já abordado no Cap 2.4.4.1, e ao mesmo tempo, propondo um *mindset* ao leitor, necessário ao contexto de viabilidade, que será detalhado posteriormente, no Cap 3.5.

Os ganhos com o uso dos repositórios conforme foi visto no Cap 2.4.4.1, nas Figuras 2.12 e 2.13, são ilustrados na Figura 2.15, pois pode-se observar que os produtos reutilizáveis: PLE3, PLE6 e PLE4, já desenvolvidos no projeto-alvo-1, e, pela equipe de reuso anteriormente, agora fazem parte do escopo do projeto-alvo-3.

Figura 2.15 – Desenvolvimento com Reuso Completo.



Fonte: Produção do autor.

Tomando-se a área dos círculos, equivalente ao escopo de cada projeto-alvo, e cada Produto-PLE apresentado nas Figuras 2.12, 2.13 e 2.15, inferi-se hipoteticamente, que cada Produto-PLE, completa 12,5% do escopo total dos projetos-alvo.

Note-se que, apesar do reuso ter ocorrido no projeto-alvo-2, através do PLE4 e PLE5, econômicamente, o reuso aparecerá apenas no projeto-alvo-3, através dos PLE3 e PLE4, ou em torno de 25%, baseado no escopo das aplicações envolvidas. O PLE4, foi desenvolvido pela equipe de reuso e aplicado nos projeto-alvo-2 e projeto-alvo-3. Logo, supondo-se esforços equivalentes para a sua construção, por exemplo, 100 horas, pode-se entender o tempo total economizado devido ao reuso como:

$$t + t_{PLE4\text{-}E} = t_{PLE4\text{-}2} + t_{PLE4\text{-}3} \qquad (2.1)$$

onde:

$t_{PLE4\text{-}2}$ = esforço economizado com o reuso no desenvolvimento do PLE4 no projeto-alvo-2;

t_{PLE4-3} = esforço economizado com o reuso no desenvolvimento do PLE4 no projeto-alvo-3;

t_{PLE4-E} = esforço da equipe de reuso no desenvolvimento do PLE4;

t = tempo total economizado devido ao reuso;

Sendo:

$$t_{PLE4-2} = t_{PLE4-3} = t_{PLE4-E} = 100 \quad (2.2)$$

$$t + t_{PLE4-E} = t_{PLE4-2} + t_{PLE4-3} \quad (2.3)$$

Então t = 100 horas para PLE4;

Supondo situação análoga, para PLE3 temos t = 100 horas, logo, a economia do ponto de vista do escopo total, é de 200 horas, equivalente a 25% do projeto-alvo-3.

Note-se que, implicitamente, adota-se uma equipe de reuso com um rendimento equivalente ao da equipe de desenvolvimento, mas há cenários em que isto não ocorre. Trata-se de uma análise interessante, mas desnecessária à esta pesquisa.

O produto reutilizável PLE3 foi desenvolvido dentro do esforço do próprio projeto-alvo-1 da Figura 2.12, seja em “a” ou em “b”,em seguida, ele foi reutilizado apenas no projeto-alvo-3, mas, neste caso, parece ter ocorrido também outra economia de tempo, pois para o PLE3 o reuso foi imediato, afinal, não foi necessário que nenhum outro projeto-alvo pagasse pelo feito de uma equipe de reuso, como ocorreu com o PLE4, mas esta afirmação pode não ser aplicável para todos cenários. Assim, tem que se entender melhor, a economia do ponto de vista do tempo.

Existe a economia em termos de escopo, que pode ser traduzida em esforço (tempo), de fato evitado no desenvolvimento dos projetos-alvo e, é possível fazer uma relação direta com o custo do projeto. Porém, existe também, o

Potencial de Reuso (PR) ou a disponibilidade do Produto-PLE, para que seja utilizado em qualquer projeto, refletindo em uma independência entre os projetos da biblioteca e do projeto–alvo, e o desenvolvimento de um, não é afetado pelo o de outro, podendo resultar em uma economia em termos de prazos.

Na Figura 2.13 o PLE4, estava pronto para ser usado no projeto-alvo-2, pois não teve que aguardar o desenvolvimento de outro projeto-alvo anteriormente. Indiscutivelmente, a duração do projeto-alvo-2 foi menor em torno de 12,5% devido ao PLE4, apesar, de econômicamente ainda ser um encargo. Por outro lado, se não fosse o PR gerado pela equipe de reuso, tal economia não seria possível.

Em relação ao que se observou anteriormente, conclui-se que, houve uma economia de tempo, mas não necessariamente devido ao PLE3, que não gerou PR, podendo ter contribuído inclusive de forma contingente, mas sim, através da PLE4 que foi propositalmente feita para gerar reuso.

Do ponto de vista do usuário ou cliente, tanto o projeto-alvo-2, quanto o projeto-alvo-3, foram beneficiados pelo reuso e, disponibilizados aos mesmos, antes do que seriam, caso o reuso não fosse praticado. Porém, do ponto de vista da companhia, fabricante dos projetos-alvos, economicamente, apenas o projeto-alvo-3 lucrou com o reuso. A isto chama-se de Ganho não Econômico do Reuso (GnER).

Para uma equipe de reuso trabalhar, é necessário um histórico dos projetos, a fim de conhecer as plataformas de hardware[17], e as funções de apoio às missões mais utilizadas, diferentemente dos projetos-alvo, onde existem questões comerciais envolvidas e que, podem atrapalhar qualquer estratégia

[17] Componentes eletrônicos de processamento como os microcontroladores por exemplo. A equipe de reuso pode até colaborar com o desenvolvimento destes fornecedores afim de praticar algum tipo de padronização destas plataformas.

maior de sequenciamento dos ciclos de vida dos projetos, podendo prejudicar a Taxa ou índice de Reusabilidade (TR), em um curto prazo.

Como os projetos-alvo são desenvolvidos por questões comerciais, o reuso é secundário e acidental e, mesmo que praticado, não é esperada alta TR. Diferentemente da equipe de reuso, onde é esperado possível aumento da TR, em prazos menores, quando comparado ao modo de reuso acidental, mesmo sabendo dos encargos, que tais Produtos-PLE terão, quando do seu uso no primeiro projeto-alvo habilitado.

Como constatado uma equipe de reuso precisa de um histórico de projetos para trabalhar, logo, não é estranho esperar-se que, no início, os projetos sejam desenvolvidos através do modo de reuso acidental, a fim de criar um volume de trabalho para esta equipe, equivalente, aos níveis F e E do modelo de maturidade, apresentado na Figura 2.11. Então, em um segundo momento, se aplicaria o reuso de forma profissional ou nos níveis superiores ao E.

Com isso levanta-se a hipótese das diferenças do crescimento da TR em função dos modos de operação do reuso, e claro, do indiscutível reflexo na economia dos projetos-alvo, em função desta escolha. Isto poderá ser alvo de um trabalho futuro, uma vez que neste objeto deve-se preocupar com as fundamentações desta proposta.

3. DESCRIÇÃO DA PROPOSTA DE REUSO

Este capítulo apresenta sucintamente a proposta desta pesquisa, quanto as táticas para o desenvolvimento dos componentes reutilizáveis, sua utilização em projetos de nanosatélites, e um modelo de viabilidade econômica.

3.1. Introdução à proposta

Em essência a abordagem proposta para realizar o reuso não é diferente em termos macro das abordagens das agências internacionais de referência, ou seja, a intenção é disponibilizar repositórios de ativos de software para que possam ser consumidos conforme, as necessidades dos projetos-alvo.

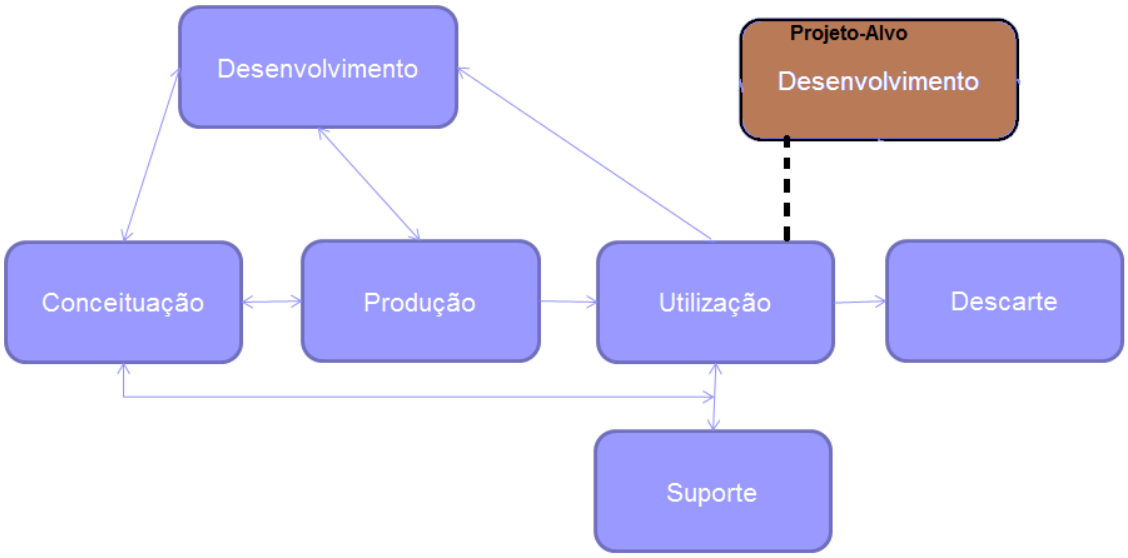
O conceito é que tais bibliotecas contenham componentes sistemicamente conectados com o propósito de serem Produtos-PLE e, através desta visão, proverem serviços úteis aos projetos-alvo (Figura 3.1). Tais bibliotecas se utilizam de padrões de arquitetura (Cap. 3.3), onde é importante a separação do *core* dos Produtos-PLE das suas interfaces, tanto das externas trocadas com o projeto-alvo, quanto das internas trocadas entre suas dependências de agregação e de composição.

Figura 3.1 – Relacionamento entre o Projeto-Alvo e o Produto-PLE.

Fonte: Produção do autor.

Além das características intrínsecas das bibliotecas, e para contextualizar o seu uso efetivo, deve-se entender seus processos de ciclo de vida, e o relacionamento, com os processos do ciclo de vida do projeto-alvo.

Figura 3.2 – Estágios do ciclo de vida da Biblioteca.



Fonte: Adaptado de INCOSE (2015).

De acordo com o INCOSE (2015), a Figura 3.2, mostra os estágios do Ciclo de Vida do sistema, e pode-se fazer uma relação com esta proposta. Uma vez que cada item da biblioteca é um Produto-PLE, e este será utilizado por um projeto-alvo, entende-se, que a biblioteca será utilizada no estágio de Desenvolvimento do projeto-alvo, mas, ao mesmo tempo, equivale ao estágio de Utilização do projeto da própria biblioteca. Fazendo com que o estágio de Desenvolvimento do projeto da biblioteca, signifique o desenvolvimento do próprio Produto-PLE.

Com isso, as principais atividades nos estágios do ciclo de vida do projeto da biblioteca, são:

- **Conceituação -** Neste estágio, será analisada a viabilidade do Produto-PLE, logo, se pretende mostrar argumentações que justifiquem um novo produto para a biblioteca.

  É de interesse da companhia, em ter o maior índice de reuso dentro do projeto-alvo, e para isso, informações retiradas de documentos do próprio processo de desenvolvimento, deveriam servir como indicador de funcionalidades e *drivers* que mais se repetem dentro do portfólio de produtos da empresa, para que assim, possam ser priorizados e

escolhidos a ingressar a fila de Produtos-PLE do estágio de Desenvolvimento da biblioteca proposta.

Neste momento, se ressalta a análise de viabilidade, através do uso de ferramentas ligadas à ontologia[18], conforme será mostrado no Cap 3.5.

Neste estágio, é discutida a definição da missão, ou dos objetivos deste produto, avalia-se o nível de maturidade das tecnologias TRL / RRL (DOWNS, 2010), utilizadas para resolver o domínio da solução, propõem-se protótipos e mitiga-se o risco para certos níveis, podendo, não serem aceitos para níveis considerados baixos (ou de alto risco).

- **Desenvolvimento -** Neste estágio, será desenvolvido um novo Produto-PLE, baseado, nos resultados positivos da sua viabilidade proposta na concepção.

  O desenvolvimento, é feito baseado nas abordagens da Modelagem, PLE e Serviços, abordados no Cap. 2.2, bem como as técnicas de suporte, abordadas no Cap. 2.3.

  Neste estágio, são aplicados os processos específicos de desenvolvimento de software, conforme recomendados no início do Cap. 2.4.

- **Produção -** Uma vez que o Produto-PLE foi desenvolvido, a produção reflete a prontidão dos ativos necessários à utilização, como o arquivo executável e seu pacote de dependências, dando condições de efetuar a configuração da sua *baseline* e dos Produtos-alvo afetados, além de, registrar as versões dos ativos através da Engenharia de Produtos e das autoridades certificadoras.

[18] Ontologia possui um significado muito maior, entretanto nesta pesquisa, seu uso restringe-se a identificação de ativos semelhantes dentro do domínio do Reuso.

- **Utilização -** Neste estágio, os produtos da biblioteca são consumidos pelos projetos-alvo. Os Produtos-PLE, são chamados pelo projeto-alvo através das interfaces externas, ou públicas de seus serviços.

  A utilização da abordagem de reuso, baseada na ontologia[19], poderá ser útil, pois do ponto de vista da biblioteca de *drivers*, tanto o arquiteto de hardware quanto o *designer* de hardware[20], desenvolvedores do projeto-alvo, poderão fazer uso das ferramentas que aplicam a ontologia, a fim de, filtrar quais são os controladores utilizados pelos Produtos-PLE já existentes na biblioteca, e que, estão disponíveis para serem reutilizados.

  De acordo com a biblioteca de funções de sistemas alocadas para o software, o arquiteto de software do Produto-alvo, poderá também, filtrar ativos que relatam as funcionalidades dos Produtos-PLE, utilizando ferramentas ligadas à ontologia.

  No ciclo de vida da utilização, os processos de desenvolvimento dos projetos-alvo são importantes, pois como existem os repositórios de reuso, é importante que os processos específicos relacionados ao reuso sejam considerados, conforme abordado no início do Cap. 2.4.

  Por fim, é neste ciclo onde ocorrerá, a coleta de dados para a validação da viabilidade dos produtos reutilizáveis, ou, o conhecimento da efetividade da biblioteca de reuso, contribuindo com a confiança que ela traz. Uma vez que tais produtos foram implantados nos produtos-alvo, será possível contabilizar seus tamanhos e comparar com as estimativas

[19] Ontologia possui um significado muito maior, entretanto nesta pesquisa, seu uso restringe-se a identificação de ativos semelhantes dentro do domínio do Reuso.

[20] Entende-se que o arquiteto de hardware trata do domínio da solução do hardware, através dos conceitos da Engenharia, e o *Designer*, conhecendo de tais conceitos, os aplicam em tecnologias disponíveis.

de viabilidade. Também será possível conhecer o índice de reuso de cada Produto-PLE, que é proporcional à quantidade de sua reutilização.

- **Suporte -** As atualizações serão resolvidas neste estágio, os *upgrades* deverão seguir para a concepção. Problemas com os Produtos-PLE, vão requerer: a atualização de seus ativos quanto ao controle de versão; sua regeração; e a disponibilidade dos arquivos necessários à utilização.

  Os Produtos-alvo que fazem uso deste Produto-PLE, poderão ou não serem afetados, uma vez, que as interfaces externas dos serviços podem se manter as mesmas, já que são utilizadas com o propósito de diminuir o acoplamento. Caso contrário, isto refletirá, e será necessário regerar o Produto-alvo afetado.

- **Descarte -** O descarte, trata da inutilização de um determinado Produto-PLE de forma segura, ou seja, tanto as chamadas com o projeto-alvo, quanto na organização da biblioteca que contém dependências com os outros Produtos-PLE, devem ser desfeitas e resolvidas.

  Conforme a reutilização dos Produtos-PLE, o índice de reutilização (DE ALMEIDA, 2007), irá aumentar, indicando portanto, um uso crescente com o passar do tempo, por outro lado uma curva decrescente, indica uma diminuição do reuso para determinado Produto-PLE. A diminuição do índice de reuso para um Produto-PLE, pode levar ao seu descarte, dependendo de uma política adotada pela própria companhia. Entretanto, deve-se atentar que, Produtos-alvo antigos, ainda podem estar fazendo uso de Produtos-PLE antigos, e que, não existe ainda uma intenção de atualização, logo, a manutenção deste Produto-PLE, apesar dele apresentar um índice de reuso baixo, pode não ser suficiente para o seu o descarte.

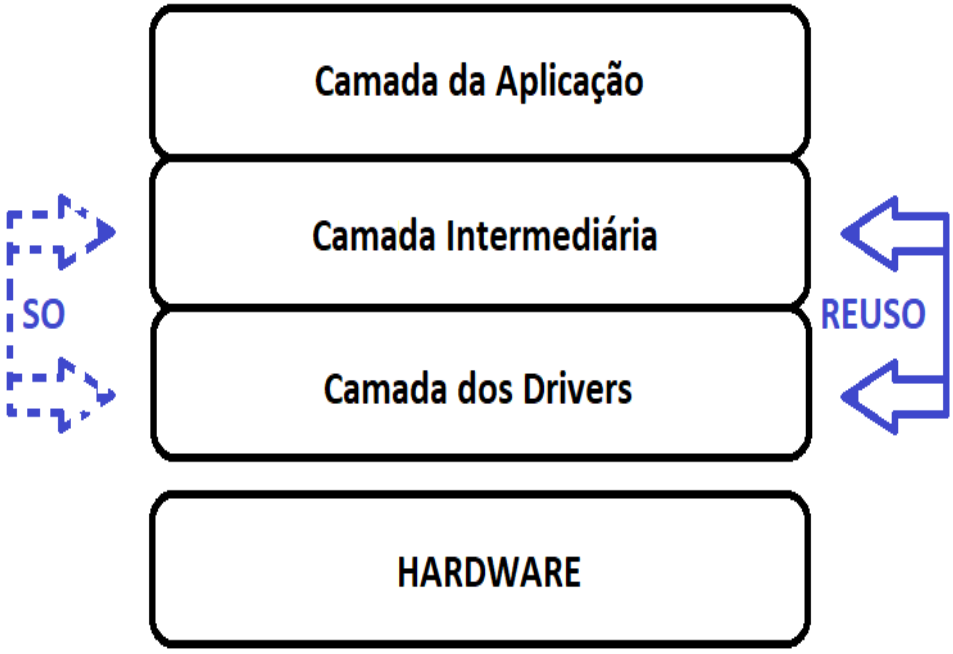
Nesta pesquisa, será detalhada, a análise de viabilidade usada no processo de conceituação, bem como, o projeto dos produtos reutilizáveis que pertencem ao estágio de Desenvolvimento.

O próximo tópico, dá início ao projeto dos produtos reutilizáveis, eles estão localizados dentro das camadas de software embarcados (as mesmas utilizadas em nanosatélites), então, explicita-se as camadas alvo desta pesquisa, e as suas diferenças com relação aos produtos reutilizáveis.

3.2. Organização dos repositórios

Separando os ativos de software nos repositórios, conforme apresentado na Figura 3.3, propõe-se a seguir, a organização da biblioteca.

Figura 3.3 – Camadas do software embarcado.



Fonte: Produção do autor.

A Figura 3.3, expõe que o SO, deverá contribuir tanto com as funções para a camada de *drivers,* conforme seus fornecedores dispõem em imagens, quanto com as funções da camada intermediária, não necessariamente ligadas ao hardware, como funções de gerenciamento de tarefas, arquivos, etc.

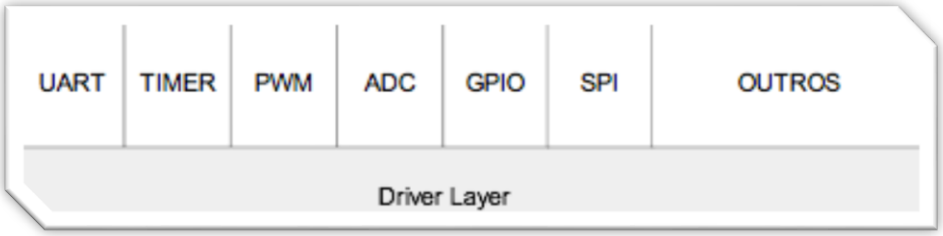
A Figura 3.3, também mostra que, o foco dos ativos de reuso, estão contidos tanto na camada mais próxima do hardware, quanto da camada mais próxima da aplicação, e isto será melhor detalhado nos tópicos que se seguem.

3.2.1. Biblioteca da camada dos drivers (BCD)

A camada de *drivers,* possui a responsabilidade de comunicar com as plataformas de hardware, e disponibilizará, seus serviços com interfaces públicas para a camada intermediária. Muitas vezes, esta camada possui outros nomes tais como: *Board Support Package* (BSP); *Hardware Abstract Layer* (HAL); e Software Básico por exemplo, e pode-se considerar, que a essência é a mesma.

Eventualmente, a biblioteca conterá *drivers* que se somam à imagem de um determinado SO, além de conter os *drivers* do próprio SO. Nesta pesquisa, este repositório possui *drivers* e será chamado de Biblioteca da Camada dos *Drivers* (BCD).

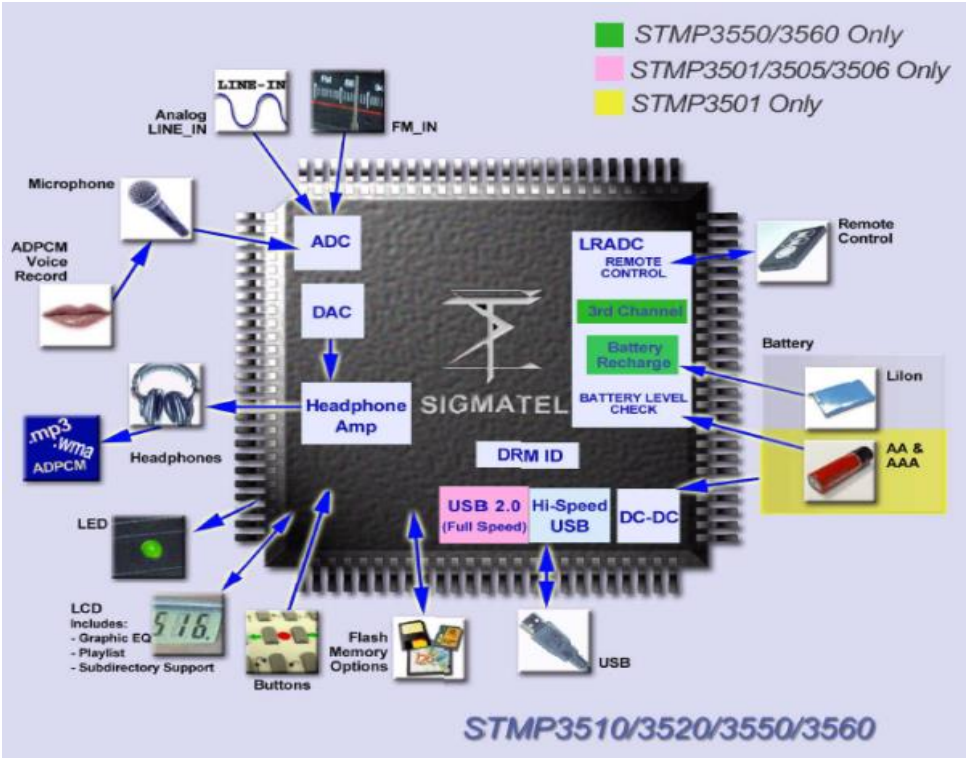
Figura 3.4 – Camada de *Drivers*.



Fonte: Produção do autor.

Os componentes eletrônicos responsáveis pelo processamento, possuem internamente periféricos diversos com responsabilidades definidas. Estas responsabilidades são configuradas, iniciadas e operadas, de forma geral, através de softwares básicos ou *drivers*. Pode-se observar, na Figura 3.5, um microcontrolador onde muitos periféricos foram construídos em hardware como *Universal Serial Bus* , *Inputs/Outputs*, *Digital-Analog Conveter*, *Conversor DC-DC*, *Analog-Digital Converter* (ADC), etc.

Figura 3.5 – Periféricos internos de um Microcontrolador.



Fonte: STMP3510 (2016).

Uma vez escritos os *drivers* para cada periférico, pode-se operar aquele determinado periférico através de um componente de software, mas muitas vezes, em aplicações mais complexas, ou a fim de moldar características comuns, visando um futuro reuso deste mesmo componente, precisa-se elaborar outras camadas de software básico, como no caso de perféricos ligados à comunicação, onde é possível extrapolar serviços sendo utilizados, por exemplo, em até 7 camadas básicas, equivalentes às camadas *Internacional Organization For Standardization (*ISO) / *Open System Interconnection* (OSI). Entretanto, este trabalho não pretende evoluir o assunto até este ponto, uma vez que questões fundamentais deverão primeiramente serem tratadas, deixando este aprofundamento como sugestão para trabalhos futuros.

Um exemplo de software básico para um componente simples, (ou não complexo, aquele que não requer mais do que duas camadas) é o *TIMER*. Neste, funções como **`iniciaContagem()`**, **`configuraContagem()`**,

`setaInterrupcoesTimer()`, são bastante comuns para qualquer *timer* de qualquer microcontrolador, quando operando como "*Counter*", por exemplo. Pois, o mesmo periférico poderia operar em outros contextos, e portanto, seriam outros *drivers* como Comparadores, *Clocks*, etc.

A camada de *driver* é a mais próxima do hardware, e também tem o objetivo de configurar o *TIMER*, ou seja, setar os registradores respectivos a fim de que, o mesmo, possa operar de acordo com a estratégia da aplicação, conforme destacado no parágrafo anterior.

As principais diferenças entre os diversos fabricantes de microcontroladores com relação aos periféricos, em geral, estão na sua constituição interna, em termos de:

- Numeração dos pinos;
- Compartilhamento de pinos entre seus periféricos;
- Quantidade de registradores para cada periférico; e
- Designação dos registradores.

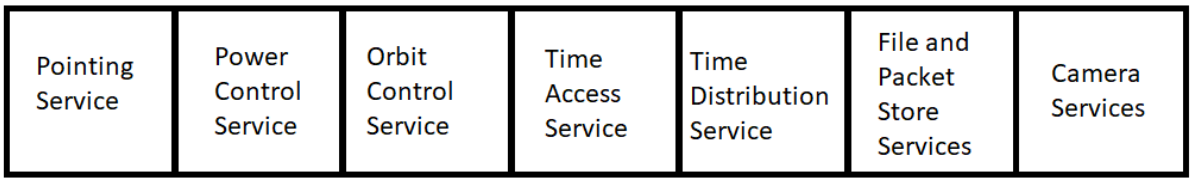
Este é o objetivo principal da camada de *drivers:* abstrair tais particularidades de cada periférico de hardware, a fim de, garantir um modo comum de acesso para as camadas superiores da biblioteca. Esta camada, pode oferecer tanto serviços (que funcionam de forma autônoma), quanto bibliotecas de forma estática ou dinâmica.

3.2.2. Biblioteca da camada intermediária (BCI)

A camada intermediária possui funções de sistemas que serão realizadas por software, seu objetivo é apoiar as aplicações, e por isso, disponibilizam serviços ou *libraries* (.a; .so; .dll, etc.) com interfaces públicas para as aplicações. Suas funções se afastam dos *drivers*, tendo o propósito, de apoiarem a missão dos projetos de um dado portfólio, ou segmento da companhia. Este repositório, possui funções de sistemas alocadas e

particionadas para software, e será chamado de Biblioteca da Camada Intermediária (BCI).

Figura 3.6 – BCI e seus constituintes.



Fonte: Produção do autor.

No exemplo da Figura 3.6, o *driver Timer* poderia ser usado em 2 contextos CCSDS-SOIS diferentes na camada intermediária, seja com o *Time Access Service* (TAS) ou *Time Distribution Service* (TDS).

3.2.3. A camada de aplicação

Os componentes de software desta camada, são aqueles que trazem exatamente a missão ou o propósito de toda a aplicação. Normalmente trazem uma máquina de estados particular para o projeto em questão, além de permitir as devidas configurações e modos de operação.

Logo, trata-se da camada com menor índice de reuso, a menos que funcionalidades comuns, possam ter suas interfaces externas isoladas e serem encapsuladas em serviços. Desta forma, por exemplo, os serviços de controle térmico, a dinâmica orbital, o controle de atitude, e etc. são muitas vezes parametrizáveis, e poderiam ser tratadas como aplicações reutilizáveis.

O que esta pesquisa propõe, é que tais aplicações sejam tratadas como serviços (aplicações autônomas), e portanto, elegíveis a se tornarem Produtos-PLE, com a diferença, que estes não requerem nenhum outro software de controle acima para ditar o seu funcionamento, como possivelmente, ocorrerá com alguns componentes da BCD e BCI, que poderiam operar como bibliotecas estáticas ou dinâmicas.

Por outro lado, as aplicações únicas dedicadas com máquina de estados individuais, e portanto com baixo índice de reuso, devem ser dependentes de *Application Programming Interface* (API), que é provida pela BCI e BCD. Estes tipos de aplicações, é que são colocadas na camada de aplicação (Figura 3.3).

3.3. O projeto dos componentes reutilizáveis (Produtos-PLE)

Uma vez conhecidos os repositórios e como estão organizados, conforme visto no Cap 3.2, este tópico detalhará considerações para o projeto dos mesmos. O projeto será visto em sua questão de planejamento, notadamente, a maior deficiência dos produtos reutilizáveis conforme Cap 1.3 e Cap 1.4 mostraram, logo, questões sobre os requerimentos, arquitetura e *design* serão explorados.

3.3.1. Adoção das *Intent Specifications* aos Requerimentos dos Produtos-PLE

Conforme já adiantado no Cap. 1.4, o planejamento do reuso, é tido como chave para o sucesso e isto inclui, a escrita dos requisitos e o desenho ou, o projeto de soluções que robusteçam o reuso. Uma abordagem a ser utilizada que contribui com o levantamento dos requisitos dos Produtos-PLE, é a adoção das *Intent specifications* (IE).

Segundo DOS SANTOS (2008), *Intent specifications* são baseadas em uma pesquisa de como os seres humanos resolvem os problemas, e sobre os princípios fundamentais da teoria dos sistemas. A estrutura das *Intent specifications* é projetada com os seguintes propósitos:

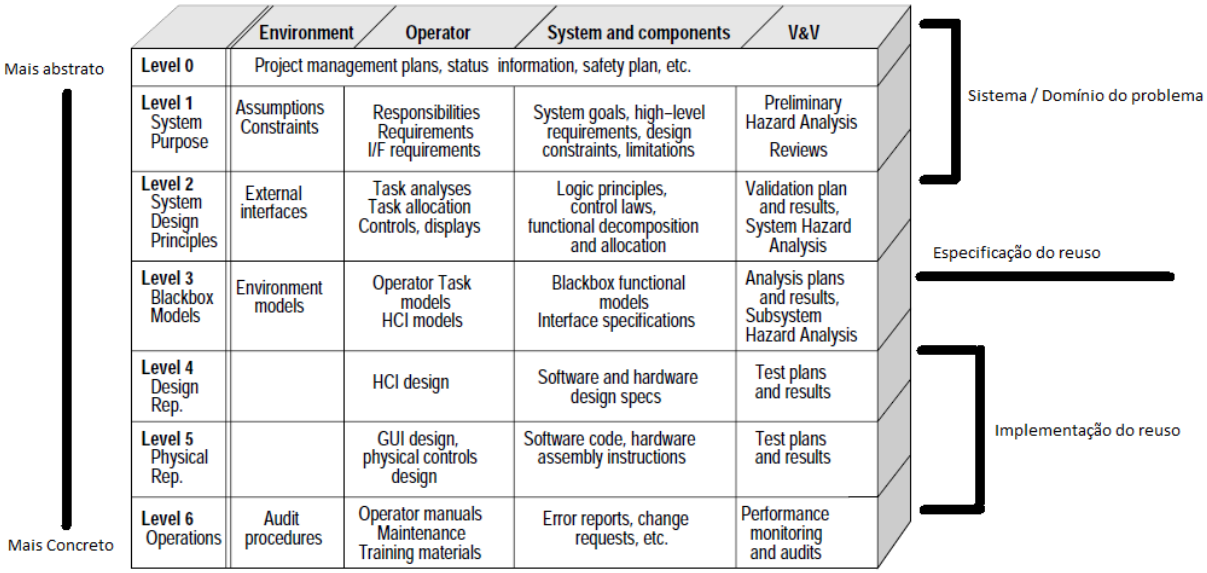
- Facilitar o rastreamento de requisitos no nível de sistema e restrições de projeto, em projetos detalhados e sua implementação;

- Auxiliar na garantia de várias propriedades do sistema (como segurança, *safety*, manutenção, etc.) do projeto inicial até a implementação; e

- Reduzir os custos do sistema ao implementar mudanças, e reanalisá-lo quando for alterado, como inevitavelmente será.

As *Intent specifications,* foram originadas na pesquisa sobre como melhorar a resolução de problemas humanos, elas devem melhorar o processamento humano, o uso de especificações e a capacidade de se realizar atividades de *design* e evolução do sistema. Não é necessária nenhuma especificação extra (supondo-se que os projetos, produzam as especificações usuais, de acordo com os processos adotados pela companhia, e aqui comentados no Cap. 2.4.1), mas apenas uma estruturação e ligação diferenciadas da informação, de tal forma que as especificações forneçam mais assistência ao processo de desenvolvimento e evolução.

Figura 3.7 – *Intent Specifications Levels.*



Fonte: Adaptado de WEISS (2004).

A chave para a reutilização bem sucedida das especificações dos componentes, reside na incorporação de toda a informação necessária para a reutilização segura na própria especificação, e documentada, de forma que possa ser facilmente encontrada, quando necessário. O uso desta abordagem, IE, satisfaz este requisito.

Apesar de todas as colunas da Figura 3.7, colaborarem com a especificação dos requisitos, a coluna *Systems and Components* é a que mais agregará quanto à arquitetura dos Produtos-PLE, alvo deste planejamento. Porém, para

fins de completude do produto, os requisitos ligados à operação, ambiente e V&V, deveriam ser considerados.

O nível do modelo caixa-preta (Nível 3), vide Figura 3.7, inclui modelos projetados para especificar e justificar (*reasoning*) o *design* lógico do sistema como um todo, e as interações entre seus componentes, bem como, seu estado funcional, sem ser levado ou sobrecarregado por problemas de implementação. Este nível, atua como uma interface inequívoca entre os níveis de sistemas e de subsistemas. Esta interface, auxilia na comunicação e na revisão dos requisitos comportamentais do componente caixa-preta, e na justificafiva (*reasoning*) sobre o comportamento do subsistema, usando revisões formais, análises formais e simulação. É neste nível que a reutilização é mais eficaz e segura.

Desta forma, os níveis 0 e 1 ficam restritos à missão do nanosatélite, sendo no nível 2, a primeira decomposição funcional do sistema. Logo, é esperado que os requisitos da BCI apareçam neste nível, e somente no nível 3, sejam encapsulados em *black-boxes*, formalizando o Produto-PLE. Como o foco desta pesquisa é o Reuso, os requisitos do nível 2 não são ainda do Produto-PLE, mas são ativos que merecem ser analizados quanto as suas similaridades, podendo serem alvos das análises de viabilidade, como será abordado no Cap. 3.5. Para só então, já no nível 3, tornarem-se Produtos-PLE no nível de abstração de subsistemas, como também de seus componentes, ambos, conteúdos da BCI.

É importante ressaltar, que a aplicação de POA se inicia na visão das funções tranversais, que podem estar presentes nos componentes (nível 3). Logo, a fim de evitar, e ao mesmo tempo colaborar com o baixo acoplamento dos futuros serviços, é no nível 2, quando as funções de sistema vêm sendo decompostas e alocadas para o software, que deverão estar devidamente declaradas. Para que assim, no nível 3, também existam os componentes transversais (e reutilizáveis conforme explicado no Cap. 2.2.4), que são criados e declarados como aspectos do sistema.

Após a elicitação dos componentes do nível 3, pode-se iniciar a escrita dos requisitos do nível 4. Neste, a preocupação está no *design* das *black-boxes* e invariavelmente na aproximação das plataformas de hardware. Nesta etapa, o objetivo é conhecer quais funcionalidades de hardware, de fato, serão necessárias para suportar os componentes do nível 3. Como por exemplo, uma função de “agendamento” necessita de um *timer,* e consequentemente de um “*driver clock*” ou uma função “tirar foto pode necessitar de uma memória e um “*driver mass storage*”, etc. Através desta análise, dará-se origem aos requisitos da camada de *drivers*, mas estes, somente serão formalizados em Produtos-PLE, quando houver similaridades de requisitos de nível 4, de cada nanosatélite de uma instituição ou companhia.

O nível 4, é uma decomposição do nível 3*,* onde em uma visão estrutural de *design*, deve-se chegar em classes, com suas associações, métodos e atributos, bem como em uma visão comportamental, define-se estados, e sequenciamentos, enfim, o necessário à equipe de implementação para que no nível 5, chegue-se aos códigos-fonte.

As informações necessárias, sobre a justificativa de projetos para a reutilização bem sucedida, incluindo os pressupostos subjacentes, sobre os quais o projeto e a validação se baseiam, são integrados diretamente na *Intent specifications*, nos requisitos das demais colunas.

Para evitar acidentes e perdas, os componentes reutilizados devem ser analisados para determinar se eles violam a lógica de projeto, e os pressupostos do sistema no qual eles devem ser usados. Este processo geralmente é impraticável, se não impossível, para a reutilização no nível do código (WEISS, 2004), mas não para a reutilização no nível 4 ou, anteriores de especificação.

Os Produtos-PLE devem atender aos requisitos do sistema, porém, eles possuem sua própria especificação, logo, uma vez que os requisitos do Produto-PLE estão no nível 3, eles devem ser rastreáveis com os do nível 2 de cada projeto-alvo, conforme o eixo *Multi-products* da Figura 2.1 recomenda. Da

mesma forma, os requisitos dos *drivers* do nível 4, devem ser rastreáveis com os do nível 3, a fim de mostrar suas dependências.

A seguir, será apresentado as táticas ligadas à arquitetura dos Produtos-PLE.

3.3.2. Adoção do padrão *Microkernel* à arquitetura dos Produtos-PLE

O padrão de arquitetura *Microkernel* (também referido como padrão de arquitetura de *plug-in*) (RICHARDS, 2017), é um padrão para a implementação de produtos baseados (*product-based*) em aplicativos.

Um produto baseado em aplicativos é aquele, que é empacotado e disponibilizado para *download* em versões, como um produto de terceiros (*third-party*) típico. No entanto, muitas instituições e companhias também desenvolvem, e lançam seus aplicativos de negócios internos (que podem estar organizados ou não em bibliotecas), como produtos de software, com versões, *release notes* e com meios de estender e somar funcionalidades àquelas já existentes (*plug-ins*).

O padrão de arquitetura *Microkernel,* permite adicionar recursos através de *plug-ins* à funcionalidade principal (*microkernel*), proporcionando extensibilidade, bem como a separação e o isolamento destes recursos.

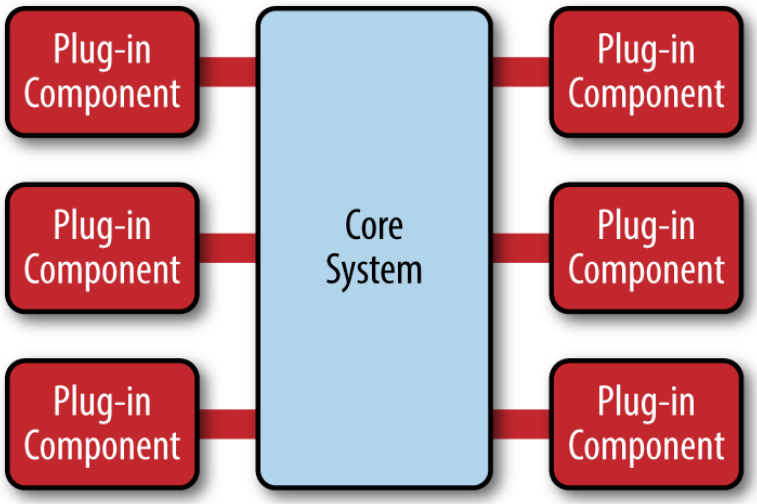
O padrão*,* consiste em dois tipos de componentes de arquitetura: um único sistema-núcleo e os módulos *plug-ins*. A lógica das aplicações, está dividida entre os módulos de *plug-ins* independentes e o sistema-núcleo básico, de forma a proporcionar extensibilidade, flexibilidade e isolamento das características-chave do núcleo, e da lógica adicional do processamento plugado. A Figura 3.8, ilustra o padrão básico de arquitetura *Microkernel.*

O núcleo (*core system*) do padrão*,* tradicionalmente, contém somente a mínima funcionalidade necessária para se tornar o sistema operacional.

É possível construir a biblioteca a partir de núcleos (*microkernels*) definidos, de tal forma, que o desenvolvedor possa escolher quais conjuntos de serviços destes núcleos, são necessários a um projeto-alvo. Tal característica aumenta

o índice de reuso da biblioteca, pois os serviços podem ser escolhidos parametrizados, e assim usados.

Figura 3.8 – Padrão Microkernel.



Fonte: RICHARDS (2017).

Fazendo uma relação com projetos de *nanosats,* os núcleos podem ser os subsistemas como AOCS, *Telemetry, Tracking, and Command*, C&DH, e etc, e de forma mais geral são equivalentes aos Produtos-PLE, com nível de abstração equivalente aos subsistemas, conforme Figura 3.9 exemplifica.

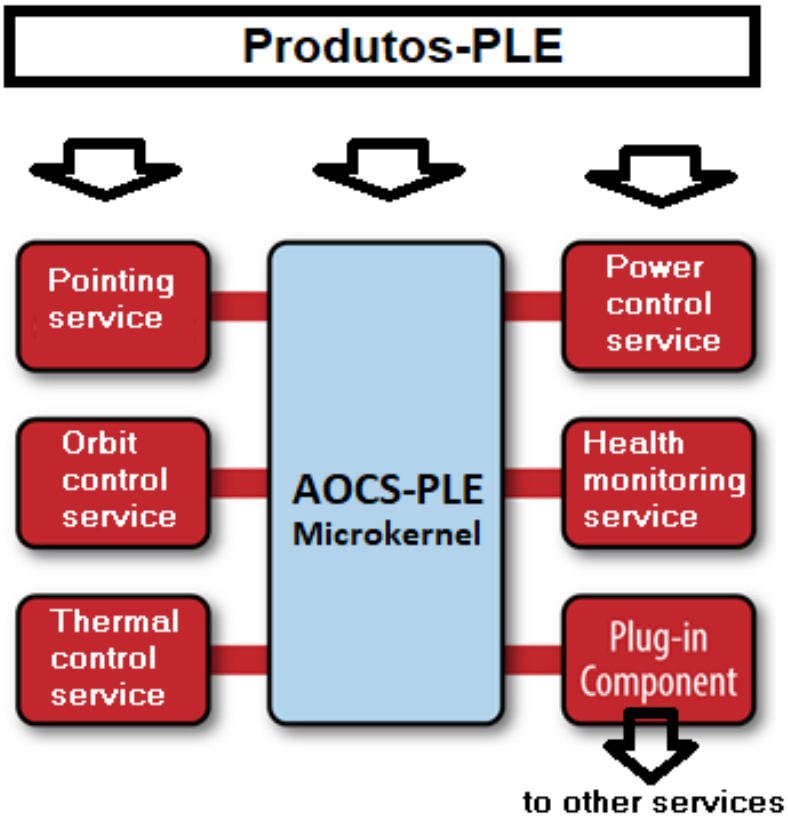
Um exemplo de núcleo comum é o SO. Todos, contêm um conjunto básico de serviços (como criar uma tarefa, excluir uma tarefa, alocar e desalocar memória, fornecer eventos de tarefas e filas de mensagens, agendar / executar uma tarefa ou conjunto, etc.).

Assim como ocorre nos SOs, onde é permitido, o desenvolvedor adicionar mais componentes para fornecer mais serviços, a biblioteca possui um conjunto de serviços, que podem ser opcionalmente aumentados em tempo de compilação do projeto-alvo, podendo ser reutilizáveis em uma variedade de contextos[21].

[21] Assume-se que o projeto-alvo de um *nanosat* faz parte de um portfólio de produtos de satélites do instituto ou companhia e que tal Módulo foi identificada com alta probabilidade de reuso através de processos e ferramentas ligadas a fase de concepção do ciclo de vida deste Produto-PLE.

Observa-se que, o AOCS-PLE[22] da Figura 3.9, quando aplicado à diferentes missões, poderá necessitar de outros serviços que são menos frequentes, mas que poderão ser “plugados” ao *microkernel* AOCS-PLE.

Figura 3.9 – Uso de *Microkernel* para subsistema AOCS-PLE.



Fonte: Adaptado de RICHARDS (2017).

Já os serviços (*plug-ins),* apesar de considerados também Produtos-PLE, com nível de abstração equivalente ao nível de componentes contidos nos subsistemas, para estes Produtos-PLE seus tamanhos não podem ser muito pequenos, pois não serão úteis para o reuso, porém, também não devem ser demasiadamente grandes, senão, o conjunto de requisitos que o satisfazem não será comum para os diferentes projetos. Normalmente para tal determinação, questões gerenciais tem maior peso frente aos fatores técnicos

[22] Componente de software reutilizável que possui, as caracteríticas do subsistema AOCS para um nanosatélite.

(DOS SANTOS, 2008), e neste caso existe um agravante, seus conteúdos, não poderão ser desplugados.

Todos os componentes dentro da biblioteca fornecem interfaces, cuja soma formam a API <<*facade*>> (FREEMAN, 2009). O padrão de design *facade,* contêm interfaces que permitem a parametrização e o acesso aos serviços, ou seja, é o meio de acesso do desenvolvedor do projeto-alvo. O padrão *facade* é implementado no componente núcleo, logo, atua como servidor tanto para o projeto-alvo, que terá acesso aos serviços dos *plug-ins,* bem como para outros Produtos-PLE, em uma relação de dependência.

Sintetizando, a ordem de dependência é que os *plug-ins* dependam apenas do núcleo ao qual pertencem, sendo sua visibilidade oculta aos demais. Os núcleos podem usar os serviços ou *libraries* dos seus *plug-ins* diretamente, via API, mas também, podem usar os serviços de outros Produtos-PLE.

O Projeto-alvo tem acesso a todos os serviços, e estes, são providos pelo núcleo. Então, existem 2 maneiras de se beneficiar dos Produtos-PLE: (1) quando os Produtos-PLE prestam serviços, a parametrização destes é a forma de tornar seu reuso factível entre os diversos produtos-alvo similares (ou da mesma família), e, (2) quando os Produtos-PLE requerem a camada de aplicação, e o produto-alvo usa da API fornecida pelo núcleo, para satisfazer suas dependências.

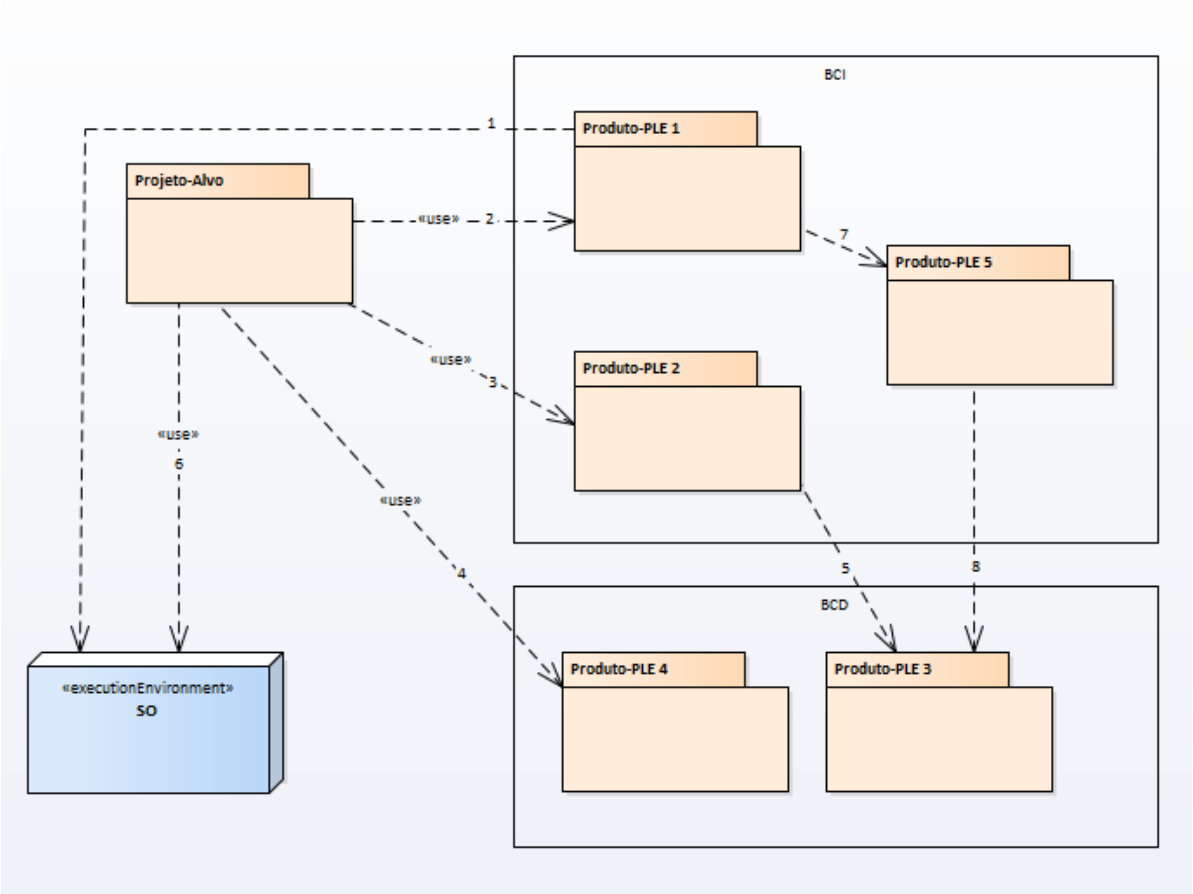
A Figura 3.10 ilustra algumas possibilidades de operação, e detalha a Figura 3.3, para propiciar o entendimento de cada dependência mostrada:

1. A dependência “2”, mostra que o Projeto-alvo usa as funções intermediárias do Produto-PLE1, porém a dependência “1,” mostra que as funções intermediárias da BCI alocadas no Produto-PLE1, precisam dos *drivers* do SO para serem executadas, e a dependência “7” mostra que as funções do Produto-PLE5 agrega ou compõe o Produto-PLE1.

Este é um típico exemplo, onde o Produto-PLE1 funciona como um serviço, logo, não requer a camada de aplicações. Tais serviços são providos ao Projeto-alvo;

2. Já a dependência “8”, mostra que funções intermediárias da BCI alocadas no Produto-PLE5, precisam dos *drivers* do Produto-PLE3 da biblioteca BCD, para serem executadas;

Figura 3.10 – Dependências dos Produtos-PLE.



Fonte: Produção do autor.

3. A dependência “3”, mostra que o Projeto-alvo também usa as funções intermediárias do Produto-PLE2, e a dependência “5,” mostra que funções intermediárias da BCI alocadas no Produto-PLE2, precisam dos *drivers* do Produto-PLE3 da biblioteca BCD, para serem executadas; e

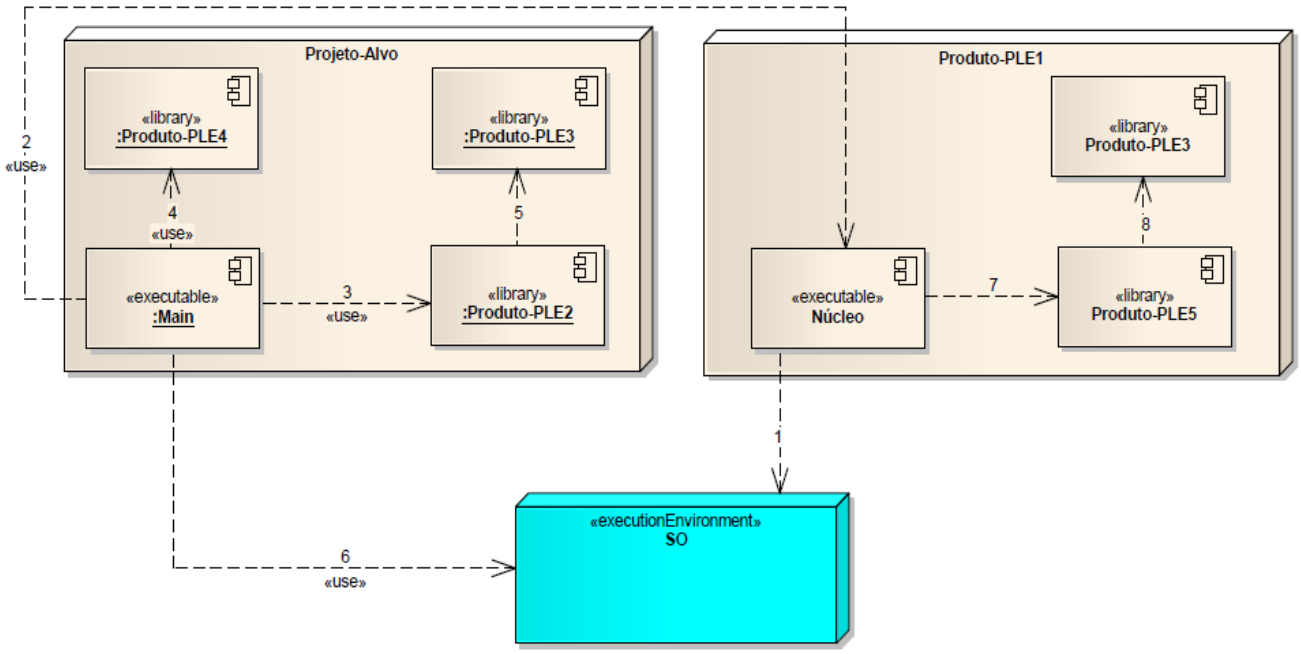
4. A dependência “4”, mostra que o Projeto-alvo usa os *drivers* do Produto-PLE4 e do SO através da dependência “6”.

A Figura 3.11, complementa o entendimento das relações de dependências visto na Figura 3.10, mostrando, que os Produtos-PLE funcionam como serviços, portanto são instâncias executáveis, como o Produto-PLE1.

Porém, os Produtos-PLE também funcionam como bibliotecas, requerendo a camada de aplicação como o Produto-alvo fez, através da API com os Produtos-PLE4 e 2.

Por último, é possível se ter múltiplas istâncias, como o Produto-PLE3, que foi implantado no Produto-PLE1 e no Projeto-Alvo, conforme mostrado na Figura 3.11.

Figura 3.11- Exemplo de implantação dos Produtos-PLE.



Fonte: Produção do autor.

Esta é a flexibilidade que o padrão *Microkernel* oferece à arquitetura das bibliotecas, uma característica essencial ao Reuso. A seguir, será detalhada a comunicação entre os Produtos-PLE que prestam serviços, bem como a

comunicação entre estes e os Produtos-alvo, equivalente a dependência “2” da Figura 3.10.

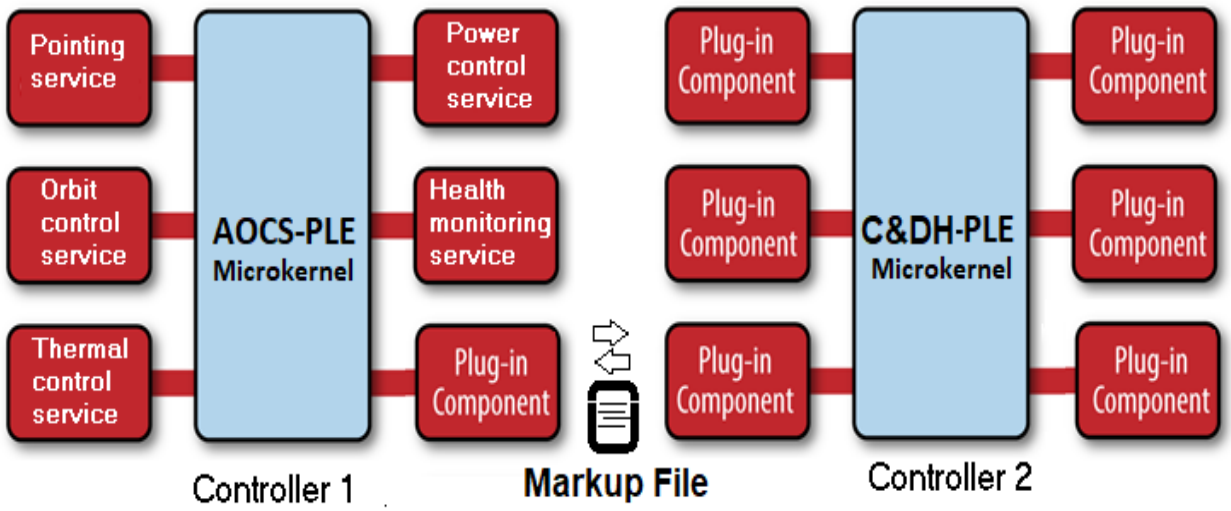
3.3.3. Adoção do padrão *Service Component Architecture* (SCA) à arquitetura dos Produtos-PLE

Service Component Architecture (*SCA)* (CHAPPELL, 2007) é uma forma de colaborar com a comunicação entre os componentes da biblioteca, para o estabelecimento das seguintes interfaces com relação aos serviços prestados: Produto-PLE com Produto-PLE; Produto-PLE com Projeto-alvo; e Produto-PLE com Componentes de terceiros, assim, ganha-se a flexibilidade da utilização de software de terceiros, e software livres em um mesmo sistema.

Esta, é o tipo de arquitetura que favorece a portabilidade, pois abre a possibilidade da comunicação entre os controladores, trocarem informações através de *markups files,* como o *eXtensible Markup Language*, ou o *JavaScript Object Notation*, por exemplo.

Na Figura 3.12, permite-se que processos sejam gerenciados por SO diferentes, se usados, e com isso colabora com a abordagem da comunicação *inter-machines*.

Figura 3.12 – Bibliotecas com SCA.



Fonte: Produção do autor.

Na Figura 3.12, o SCA viabiliza tecnicamente as dependências entre os Produtos-PLE que rodam em diferentes controladores. Pode-se pensar, que o C&DH-PLE[23] foi escrito para Linux e C++, porém, o AOCS-PLE roda em outra máquina com VxWorks ou QNX ou até mesmo sem o SO, usando somente linguagem "*C"*. Ele possui um nível de dependência com o C&DH-PLE , mas esta dependência, está desacoplada das tecnologias supracitadas, devido ao uso de *markups files,* fazendo com que as máquinas (controladores) possam empregar qualquer uma delas.

Enquanto o *Microkernel* se preocupa com a arquitetura de cada Produto-PLE, a SCA infere nos inter-relacionamentos entre eles.

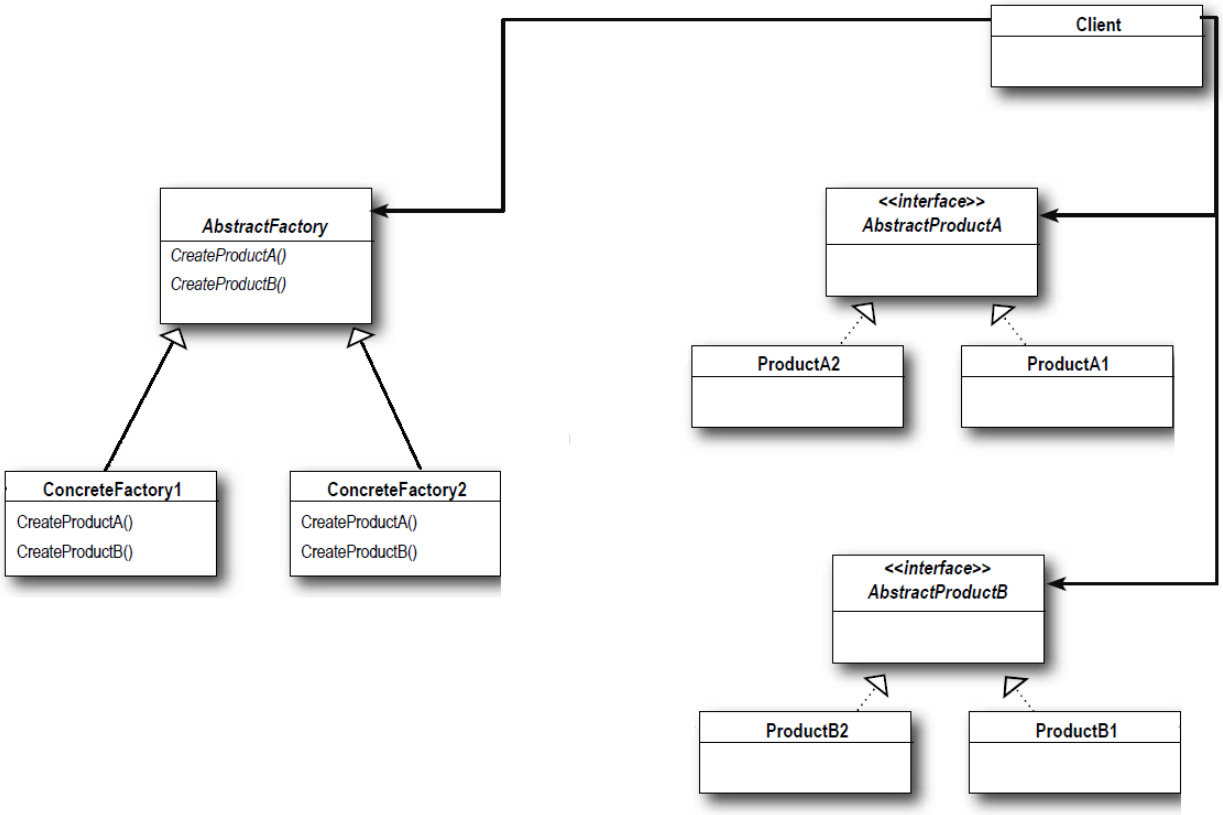
3.3.4. Adoção do padrão *abstract factory* ao Design dos Produtos-PLE da BCD

Foi visto que o *Microkernel* e o SCA organizam, e relacionam os diversos Produtos-PLE dentro do repositório, mas nada foi dito sobre como intrinsicamente o se desenvolve. Então, será aprofundado o nível de abstração, desde os serviços, até classes, de acordo com a Programação Orientada a Objetos (POO).

Para a BCI, o seu *design*, está voltado às técnicas utilizadas na Engenharia de Sistemas. O INCOSE (2015) é uma execelente referência, mas DOUGLASS (2009), pode ser aplicado após a definição da missão dentro da visão do Sistema, e das preocupações com os diversos cenários do ciclo de vida. O desenho da arquitetura seguem pontos de vista estrutural, comportamental, de interfaces, de implantação, da confiabilidade e *safety*. E não será modelado em *design* nesta pesquisa.

[23] Componente de software reutilizável que possui, as caracteríticas do subsistema C&DH para um nanosatélite.

Figura 3.13 – Padrão de *design* da BCD.



Fonte: Adaptado de FREEMAN (2009).

Diferentemente da BCI, a BCD está próxima do hardware, uma forma de realizar o seu *design,* é fazendo uso do *pattern Abstract Factory* (FREEMAN, 2009), pois, dá a flexibilidade para inclusão de novos controladores e ao mesmo tempo, a inclusão de novas funcionalidades de acordo com os periféricos disponíveis neles.

Quando se faz o uso da abstração, uma para a criação de um controlador abstrato, e outras para a criação de funcionalidades abstratas, logo, como em um cruzamento de linhas e colunas, será possível fabricar um controlador qualquer concreto, com seus devidos periféricos, a fim de apoiar as funcionalidades necessárias à *feature*-PLE.

Tomando como referência à Figura 3.13, a classe abstrata *abstract Factory,* faz o papel de um microcontrolador abstrato, que poderia ser atribuído a qualquer

nanosatélite. Nela declara-se como métodos, os Produtos-PLE da camada de *drivers*.

Tais produtos, são especializados através das classes *ConcreteFactory1,* e *ConcreteFactory2*, que fazem o papel dos microcontroladores reais dos nanosatélites utilizados nos projetos das instituições e companhias. Nota-se, que tais microcontroladores, deverão ter os mesmos *drivers* ou Produtos-PLE da classe *abstractFactory.*

Na Figura 3.13, as interfaces *AbstractProductA* e *AbstractProductB* represetam os Produtos-PLE abstratos, e neles contém os métodos respectivos de um determinado *driver*.

Tais interfaces, são realizadas por *ProductA1* e *ProductB1* a fim de resolver suas funcionalidades através do microcontrolador da classe *ConcreteFactory1,* enquanto as interfaces realizadas por *ProductA2* e *ProductB2,* tem suas funcionalidades resolvidas através do microcontrolador da classe *ConcreteFactory2.*

Isto é possível, pois o método *CreateProductA()* da *ConcreteFactory2, por exemplo,* cria um objeto do tipo *ProductA2* e retorna esse objeto, como do tipo da interface *AbstractProductA*.

Por fim, a classe *client* usa esta infraestrutura, fazendo o papel das entidades das camadas superiores à BCD.

3.4. O uso dos Produtos-PLE nos nanosatélites

Uma vez que os Produtos-PLE estão disponíveis em seus devidos repositórios, será apresentado formas de trazer esses produtos para dentro do *design* dos *nanosats*. Para isso, será explorado o uso do Produto-PLE em quatro situações que influênciam no *design* de um *nanosat*: (1) o uso ou não de sistema operacional, (2) o uso na arquitetura federada, (3) o uso na arquitetura *Integrated Modular Avionics* (IMA), e (4) a adequação do Produto-PLE aos serviços CCSDS-SOIS.

3.4.1. Reuso em *Nanosats* com Sistema Operacional (SO)

O uso de um SO, somará funções intermediárias e de *drivers* no projeto (gerenciamento de memória, *Inputs/Outputs*, *file system*, *tasks manager*, etc.), entretanto, tais funções são relativas aos diversos gerenciamentos que o SO faz do hardware, e não agregará funções intermediárias relativas à missão do projeto, ou seja, o SO não agrega à BCI.

O uso do SO requer *drivers* para controlar os periféricos, e uma vez que o projeto use componentes de hardware suas interfaces se utilizam dos *drivers* que o SO já fornece, então o banco de *drivers* da biblioteca já está definido,e influênciando na BCD.

Assim, os projetos ficam dependentes destes *drivers* e do SO como um todo, entretanto, deve-se analisar a viabilidade dos demais projetos do portfólio poderem se utilizar do mesmo componente, porém, sem fazer o uso do SO, bem como o mesmo componente fazer uso de outros SOs.

Seja no uso de uma arquitetura IMA, onde é requerido um SO com uma API que permite criar e gerenciar as diversas partições das aplicações, ou no uso de uma arquitetura federada, onde o SO não é obrigatório, os repositórios de reuso poderão contribuir em ambas as camadas:

a) As camadas de *driver* (BCD); no caso da ausência de SO ou de *drivers* não inclusos na imagem do SO; e

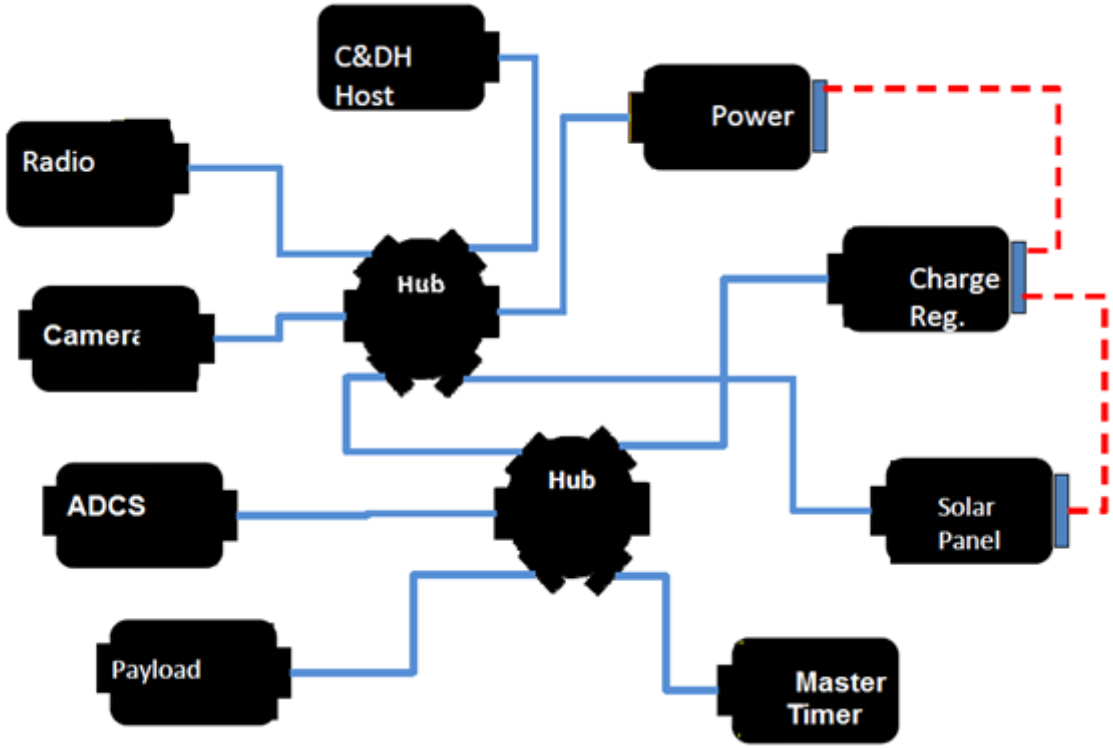
b) Na camada intermediária (BCI); através de funções que auxiliam as aplicações e as funções de abstração do sistema operacional, a fim de permitir que o componente seja controlado em mais de um SO.

A seguir, aborda-se as principais escolhas de arquiteturas que regem o desenvolvimento do Projeto-alvo, e detalha-se como as bibliotecas de reuso poderiam contribuir dentro destas escolhas.

3.4.2. Reuso em *Nanosats* com Arquitetura Federada

A arquitetura federada está baseada sobre módulos computacionais ou unidades de processamento distribuído. Em *nanosats* projeta-se esta arquitetura através de unidades e genericamente cada uma possui objetivos bem definidos e aplicações que resolvem estes objetivos dentro do próprio módulo. Tais unidades estão conectadas umas às outras através de uma rede comum de comunicação, portanto formando *nanosats* como um todo, e pode desta forma ser substituída, adicionada, retirada e reparada individualmente, conforme mostrado na Figura 3.14 (LYKE,2010); (NASA, 2014).

Figura 3.14 – Exemplo de satélite utilizando a Arquitetura Federada.



Fonte: Adaptado de LYKE (2010).

Na Figura 3.14, cada Módulo possui seu próprio controlador em hardware, bem como os periféricos necessários para fornecer as suas características, que podem ser internos ou externos ao controlador. Todos os módulos possuem

periféricos comuns de rede para comunicação, como por exemplo, *Spacewire*, *Ethernet*, RS422, CAN, I2C e etc.

Nesta arquiteturura, cada Módulo pode representar um Produto-PLE, quando dispensar o uso da camada de aplicações. Neste caso, proverão serviços que devem ser reutilizáveis não em apenas um *nanosat*, mas que através de parametrizações, possam ser implantados em vários de um mesmo portfólio.

Produtos-PLE possuem outros Produtos-PLE, que são os *plug-ins*, e estão no nível de componentes internos deste Módulo. Os serviços do Módulo são providos por suas interfaces externas, e estão acessíveis através dos *markup files* do SCA, como em um "servidor"[24]. Desta forma, outros Módulos, "clientes"[24], podem requerer tais serviços.

Entretanto, os Produtos-PLE agindo com bibliotecas dinâmicas ou estáticas, terão uma camada de aplicação, que controla a máquina de estados do Módulo, dispensando o uso do SCA, fazendo o uso das bibliotecas através da sua API, disponível pelo *facede* do núcleo (*Microkernel*) do Produto-PLE.

O Produto-PLE, pode ser aplicado nas camadas de *drivers* e intermediárias de cada Módulo, pois, escolhido o controlador e os periféricos, então escreve-se os *drivers* para cada periférico uma única vez ,e escreve-se as funções de missão, ou da camada intermediária daquele Módulo. Com os códigos disponíveis nas bibliotecas de reuso, estes ficarão disponíveis para quaisquer *nanosat* (projeto-alvo[25]) onde for necessário o mesmo Módulo .

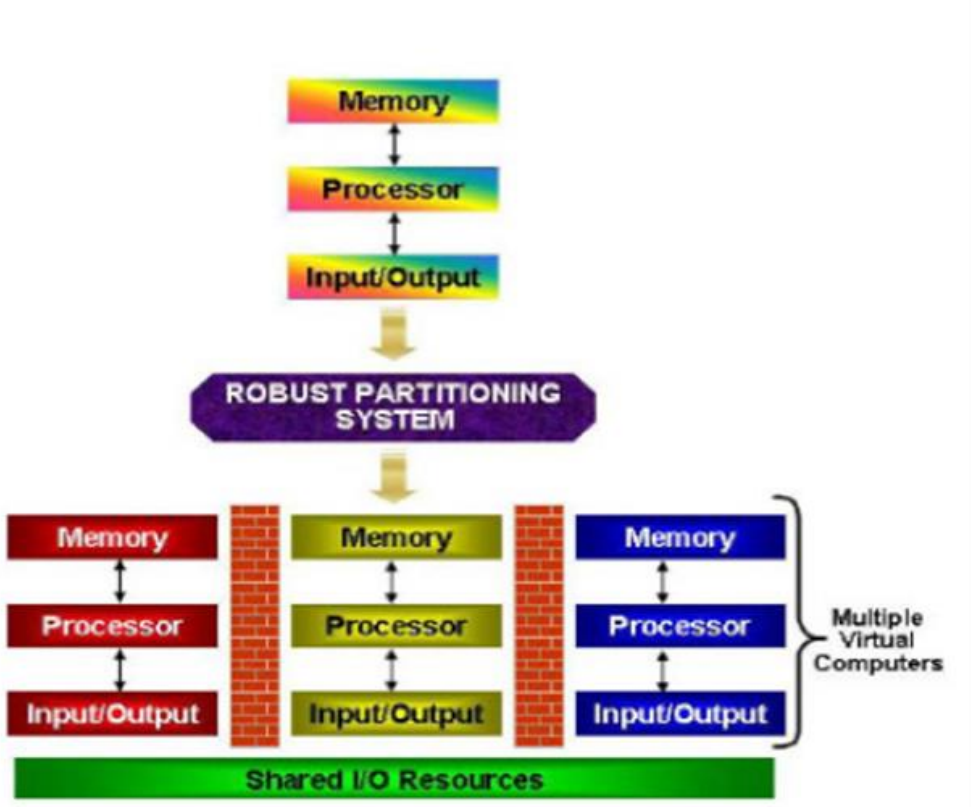
[24] Do modelo cliente-servidor (TANENBAUM, 2011).

[25] Assume-se que o projeto-alvo de um nanosat faz parte de um portfólio de produtos de satélites do instituto ou companhia e que tal Módulo foi identificada com alta probabilidade de reuso através de processos e ferramentas ligadas a fase de concepção do ciclo de vida deste Produto-PLE.

3.4.3. Reuso em *Nanosats* com a arquitetura *Integrated Modular Avionics* (IMA)

Numa arquitetura IMA, existem várias aplicações de diferentes funcionalidades, porém, cada conjunto de aplicações, que implementa uma função de sistema possui um robusto mecanismo de partição de forma que o controlador e os periféricos do hardware compartilhe estes recursos, simulando vários computadores virtuais que hospedam cada aplicação. A Figura 3.15 mostra três partições, operando como computadores virtuais, cada uma possuindo um espaço de memória, processador e Entrada/Saídas.

Figura 3.15 – Arquitetura Modular - IMA.



Fonte: TAGAWA (2011).

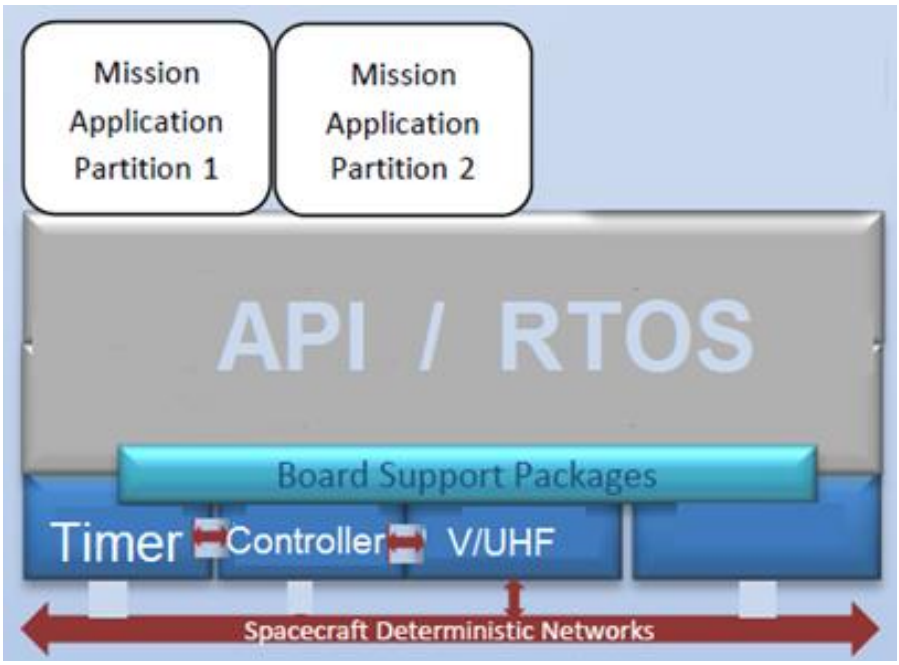
Isso é possível graças a uma API que fornece funções como: gerenciamento da partição; gerenciamento do processo, onde as aplicações irão rodar; gerenciamento do tempo das aplicações; gerenciamento de memória usada pelas aplicações, a fim de garantir independência entre as partições;

comunicação entre partições realizada, normalmente, entre canais específicos; entre outras.

Esta API, conforme mostrado na Figura 3.16, representa uma camada de software intermediária de funções, que usa os *drivers* (BSP) desenvolvidos para o controlador e os periféricos.

Dentre os objetivos desta arquitetura, destaca-se o compartilhamento dos recursos computacionais, de tal forma que o hardware será usado por muitas aplicações. A Figura 3.16 mostra duas aplicações, *Mission Application Partition 1 - MAP1* e MAP2, que podem representar Módulos de *nanosats* ou funções dos mesmos, para o caso de um Módulo ser composto por mais de uma MAP, ou Produtos-PLE, que irão reduzir as necessidades de hardware do *nanosats* e, consequentemente o seu peso, tamanho, e consumo, .

Figura 3.16 – MAPs de uma Arquitetura IMA.



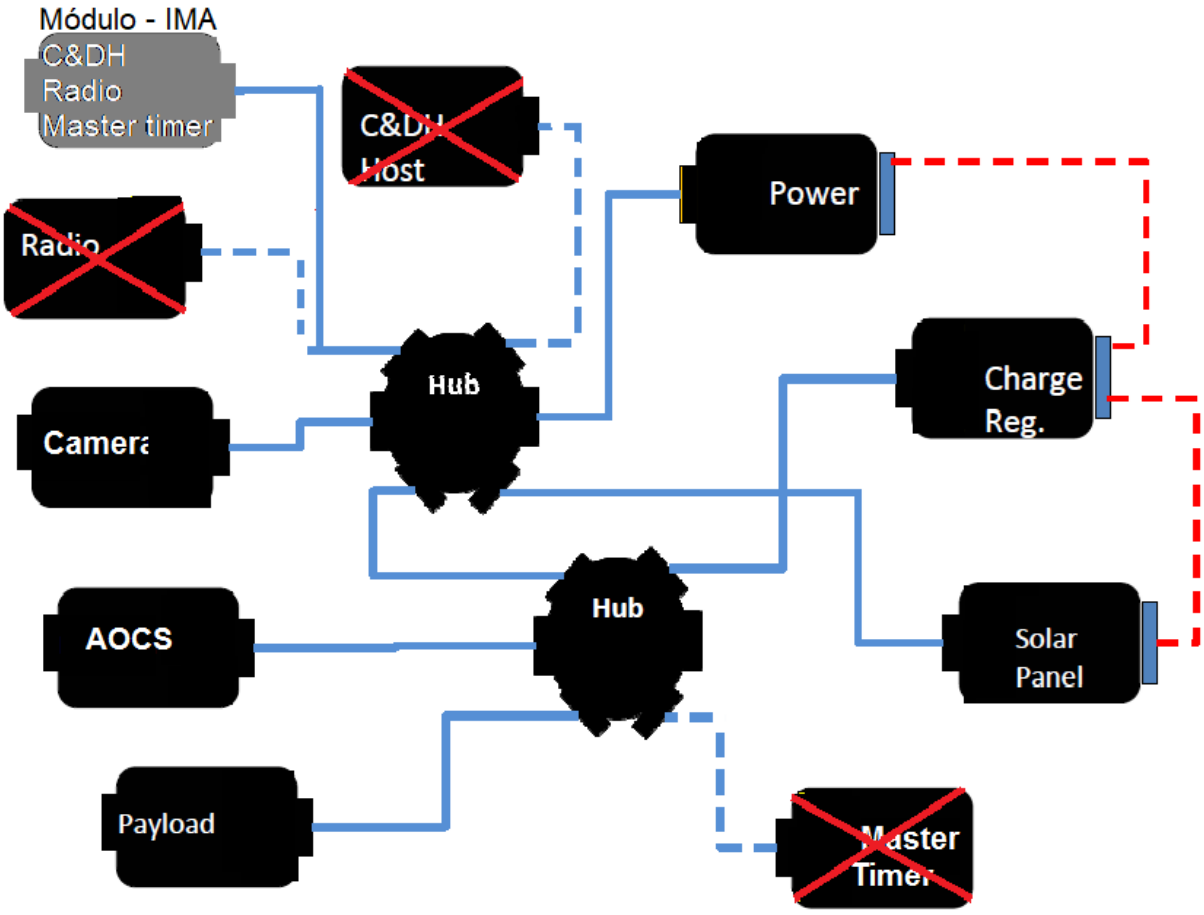
Fonte: Produção do autor.

Por exemplo, pode-se utilizar um único controlador para assumir funções de C&DH, *Radio* e *Master Timer* e, consequentemente, teria-se os mesmos periféricos necessários aos respectivos módulos de uma arquitetura federada,

como *transceivers* VHF/UHF e relógio de alta precisão, mas na arquitetura IMA, utilizá-se apenas um único controlador, vide Figura 3.16. Logo, com esta composição usando um Módulo, quando comparada ao *nanosat* da Figura 3.14, se reduziria ou integraria as funcionalidades, vide Figura 3.17.

Outro benefício desta arquitetura é que existe uma API comum e, normalmente de mercado, entre o hardware e as camadas de aplicação, como o ARINC 653 que implementa a API para uma gama de SO, logo, a aquisição de softwares COTS que resolvem determinadas funções tendo como premissa, operar sobre a mesma API aumenta as alternativas de escolhas de fornecedores especialistas no assunto (*best-in-class)*.

Figura 3.17 – *Nanosat* híbrido com Arquiteturas Federada e IMA.



Fonte: Produção do autor.

Diante deste aprimoramento de uma arquitetura centralizada, devido à API, pode-se trabalhar com diversas funções de diferentes níveis de criticidade sobre a mesma plataforma de hardware do Módulo. Porém, as aplicações estão particionadas, proporcionando uma segurança maior no compartilhamento dos recursos disponíveis pelas plataformas.

Caso opte-se pela construção de um *nanosat,* usando uma arquitetura IMA em partes do projeto ou em sua totalidade, a biblioteca de reuso terá a Camada de *Drivers* e a Camada Intermediária de funções. Entretanto agora, ambas as camadas dependem da API, pois devem ser escritas de forma compatível com o *Real-Time Operational System* (RTOS) aplicado, e utilizando a API fornecida pelo mesmo.

Para os *drivers* os Produtos-PLE, podem representar periféricos novos, agregados ao microcontrolador e que por sua vez não pertencem a imagem padrão do *kernel* do RTOS, devendo ser construído e integrado ao mesmo. Neste caso, recomenda-se a leitura do *paper da* Windriver (FIET, 2015), no qual é exposto considerações de *Safety* sobre o particionamento e o problema da concorrência de acesso aos periféricos de hardware, no caso da implementação de *devices drivers* para IMA, usando o RTOS VxWorks.

Os Produtos-PLE da camada intermediária, contêm funções orientadas à missão e são implementadas diretamente através do sistema de particionamento do RTOS dentro das MAPs. Exemplos para esta camada são: as funções de rede, apesar de normalmente serem cobertas pelo próprio RTOS; funções genéricas do SOIS, e as funções ligadas diretamente à missão como: o C&DH*, Mastertimer, Radio* e etc.

Quando os Produtos-PLE não requerem a camada de aplicação, as interfaces públicas dos serviços em uma comunicação entre Módulos (vide Figura 3.17), são realizadas através de *markups files* da SCA. Já, para a comunicação dentro dos Módulos, ou seja, entre as MAPs (vide Figura 3.16), são realizadas através da própria API do *plug-in* diponibilizada pelo *facade* do núcleo (*Microkernel*) da MAP. Afinal, como as MAPs estão sob o mesmo controlador, o

mesmo sistema operacional e a mesma linguagem, não requerem tal adaptador.

A comunicação se dá através da API disponível pelo núcleo do Produto-PLE, quando a MAP contém os Produtos-PLE e a aplicação.

Os Produtos-PLE, quando usados no projeto-alvo IMA, dentro das MAPs, são objetos (*runtime*) e operam como serviços do Módulo-IMA (vide Figura 3.17), estes, são implantados através de *Threads (*que são instâncias de um processo) e arquivos de configuração no *startup* do SO, que fazem as configurações iniciais e a inicialização do processo da MAP, executando-a.

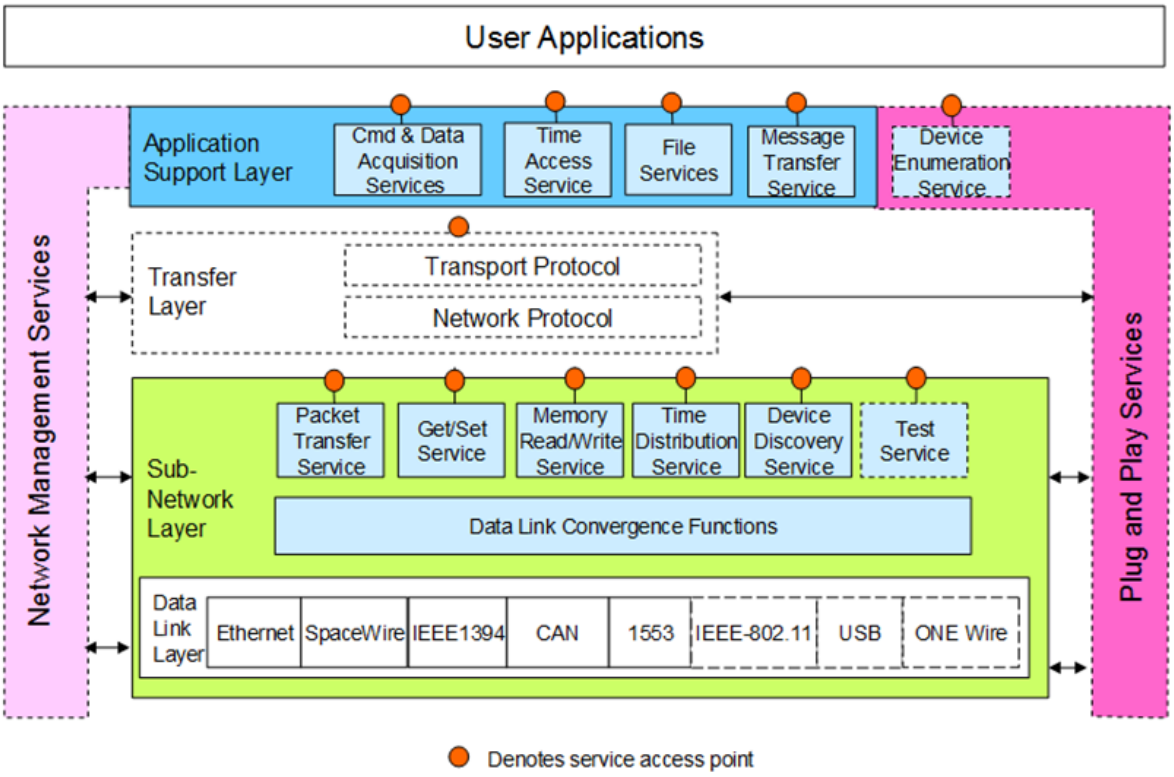
Uma vez que a plataforma de hardware do Módulo foi padronizada, e os Produtos-PLE escritos, todos projetos de novos *nanosats* com a mesma plataforma, poderão reutilizá-los, seja através dos serviços ou da API disponível.

3.4.4. Reuso de software com CCSDS-SOIS

O conjunto de serviços oferecidos pela arquitetura *Spacecraft Onboard Interface Services* (SOIS), uma das áreas de concentração do *Consultative Committee for Space Data Systems* (CCSDS), é mostrado na Figura 3.18. Como o foco do SOIS inclui a padronização de funções genéricas do satélite, uma série de serviços poderiam tornar-se Produtos-PLE.

Serviços como o TAS e o TDS, podem ser implementados em um módulo C&DH, a fim de receber a referência temporal do sistema, *Mission Elapsed Time*, do módulo *Master Timer,* numa arquitetura federada por exemplo (Figura 3.14). Neste caso, se teria um controlador e um periférico *timer,* com dispositivo de alta precisão que envia dados do relógio mestre (*Master Timer*) para sincronizar um relógio local do Módulo C&DH.

Figura 3.18 – Arquitetura SOIS.



Fonte: CCSDS-SOIS (2013).

Não apenas o C&DH faz uso dos serviços de tempo SOIS, mas, estes serviços também poderiam fazer parte da maioria dos módulos nos *nanosats,* devido à qualquer necessidade de sincronismo.

Outros serviços, como *Message Transfer Service* (MTS), que permite as aplicações de bordo se comunicarem umas com as outras, usando mensagens: assíncronas; *ad-hoc e* mensagens discretas com uma latência limitada incluindo *multicast* e *broadcast,* independente de suas localizações. Os *File and Packet Store Services* (FPSS) são serviços para acessar, gerenciar e transferir dentro do *nanosat* arquivos e pacotes que poderiam conter qualquer tipo de dados, incluindo telemetria, comandos e sequências de comandos, atualizações de software, imagens e outras observações científicas, também

estão presentes na maioria dos Módulos, e portanto são candidatos a Produtos-PLE.

Outro exemplo é o *Packet-based message protocol*, da ECSS *Telemetry and telecommand packet utilization* (ECSS-E-ST-70-41C, 2016), que padroniza os protocolos de *Telecommand* (TC) e *Telemetry* (TM) para a operação e, se escolhido para compor uma *feature*-PLE em termos da camada intermediária, poderia ser usado em vários satélites. Logo, os serviços SOIS abrem a possibilidade do reuso para uma grande gama de sistemas de satélites.

O reuso destas funções são elevados dentro da camada intermediária, logo, pode-se disponibilizá-las na BCI.

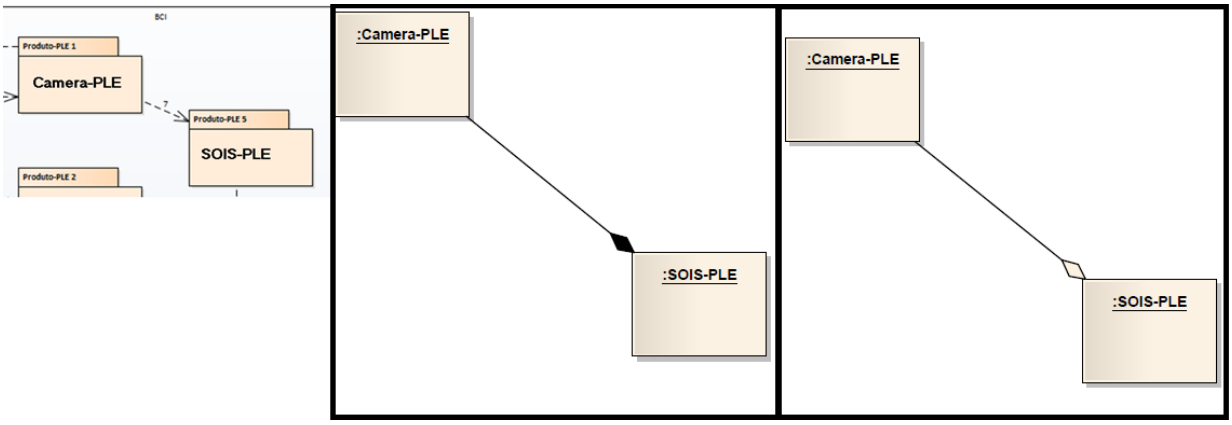
Note-se que a arquitetura SOIS atende às funções genéricas de um satélite, por isto o seu alto índice de reuso, mas os Módulos de *nanosats* com missões específicas não poderão conter tais serviços internamente, mas assim mesmo contarão com suas camadas intermediárias de funções, que são reutilizáveis.

Um exemplo disto é o Módulo Câmera, cujo objetivo pode ser o de tirar fotos de forma periódica, agendada ou atendendo à evento assíncrono. Neste caso, uma função ***`tirarFotos()`*** é útil , assim como ***`iniciarFilmagem()`***, ***`pararFilmagem()`*** e etc. Tais funções, compõem a BCI do Módulo Câmera. E os serviços da arquitetura SOIS, podem complementar estes serviços através do TAS, TDS, MTS, FPSS, etc. Contudo os mesmos serviços já foram mencionados também para o C&DH, por isso deve-se haver uma isolação através de componentes e interfaces para cada serviço da arquitetura SOIS.

Conclui-se que o Módulo Câmera na BCI é formado junto com os serviços da arquitetura SOIS, propiciando, uma associação de composição, caso se deseje que a Câmera-PLE não exista sem os serviços SOIS-PLE, ou com apenas uma agregação, caso entenda-se que as perdas de capacidades e desempenhos da Camera-PLE, devido a perda dos serviços SOIS-PLE, não comprometem sua missão.

A Figura 3.19 apresenta a modelagem dessas situações e relaciona a dependência “7” da Figura 3.10, como forma de possíveis *designs*.

Figura 3.19 – Composição e agregação da Camera-PLE e SOIS-PLE.



Fonte: Produção do autor.

Cada serviço da arquitetura SOIS poderá ser portado para outros satélites, conforme padronização dos controladores e periféricos de hardware, incluindo o tipo físico da interface de rede. Logo, existe uma dependência do *design* ou das plataformas de hardware (nível 4 da IE), mas, em termos da camada intermediária trata-se de funções genéricas, existindo uma grande probabilidade destes serviços estarem presentes na maioria dos subsistemas de um satélite.

3.5. Viabilidade Econômica

Foram abordadas táticas aplicadas ao desenvolvimento do produto reutilizável ao longo do Cap. 3.3, entretanto, a viabilidade econômica, tem o objetivo de contribuir na tomada de decisão, procurando saber se vai, ou não ser lucrativo investir no desenvolvimento deste produto. Tal análise é realizada no estágio de conceituação do ciclo de vida do produto.

É importante definir o entendimento dado nesta pesquisa com relação à viabilidade. Para isso, alguns conceitos são importantes: (1) a quantidade de reuso, ou seja, a porcentagem dos componentes reutilizados dentro da aplicação como um todo. Este é o reuso do ponto de vista dos artefatos ou dos

ativos reais, e ele é altamente mensurável e independe da equipe, ou de seu esforço.

Porém, o reuso é muito mais útil para as companhias, quando ele faz aumentar a produtividade, então, (2) o tempo, de fato economizado devido ao reuso é importante ser mensurado, e este é outro resultado importante, mas totalmente dependente da equipe e consequentemente do seu esforço.

A proposta da viabilidade econômica do reuso de software, se dá entre outras técnicas, através de métricas, pois elas mostrarão quanto tempo será economizado, e qual o tamanho do reuso em cada aplicativo de um dado projeto-alvo.

Quando se fala em tempo, existe uma economia real dada através da contabilização do tamanho do reuso (BAUER, 2009) no aplicativo do projeto-alvo, bem como existe através do Ganho não Econômico do Reuso (GnER), no qual uma equipe dedicada ao reuso desenvolve um Produto-PLE, mas este, ainda não teve utilização em um projeto-alvo. Porém, do ponto de vista do usuário final (ou cliente), ambas as formas resultarão na economia dos prazos e na satisfação do cliente.

Sendo assim neste trabalho, considera-se a viabilidade econômica baseada no tempo da economia e na quantidade do reuso. O cálculo do tempo economizado, depende do esforço da equipe e do tamanho do reuso, devendo ser medido de forma empírica, ou através de taxas de esforços já conhecidas pelo gestor das equipes. Por exemplo “a” horas/FP ou “b” horas/UCP ou “c” horas/SLOC[26] , ou também poderão ser estimadas conforme mostrada em Douglass (2009).

[26] Contabilizar o tamanho do reuso atualmente destaca-se de três formas mais usuais: contabilização das linhas de código, soma dos pontos-de-função ou soma dos pontos-de-Caso de Uso.

a,b,c $\in \mathbb{R}^{*}_{+}$; FP = *Function Point;* UCP = *Use Case Point;* SLOC = *Source Line of Code;*

Uma vez conhecido o tempo economizado, será necessário julgar se ele é viável ou não. Para contribuir neste aspecto, recorre-se ao uso de ferramentas ligadas à Ontologias[27].

3.5.1. Ferramentas ligadas à Ontologias

As ferrament

as ligadas às Ontologias[27], podem ser úteis em vários momentos, seja contribuindo com o aumento do índice de reuso dos Produtos-PLE já desenvolvidos, como também, contabilizando o índice de uso real de uma função ou de um *driver* e, consequentemente, se aposta em uma probabilidade de reuso para os novos projetos. E é nisto, em que a viabilidade está baseada.

Para a viabilidade, o desenvolvedor de software da biblioteca de *drivers* ou da biblioteca de funções, requer que a ferramenta analise os *drivers,* e funções utilizados nos projetos do portfólio da instituição, baseando-se nos documentos da arquitetura dos projetos-alvo e ao final, indique aqueles que mais se repetem dentro de uma lista priorizada.

Os *drivers* e funções de maiores prioridades, serão sugeridos como novos Produtos-PLE, seja para a BCD ou BCI. A prioridade é dada em função dos tempos da economia[28] e das repetições destes *drivers* e funções, logo, a soma destes tempos terão maior prioridade, sendo possível se estabelecer critérios de viabilidade econômica, para a BCD e BCI.

É importante frisar que, desta somatória de tempos da economia, deveria ser descontado os encargos intransponíveis, devido às alterações do processo da utilização das bibliotecas de reuso e, de outras ferramentas ligadas à viabilidade econômica do reuso. Afinal, quando não se faz reuso, os processos

[27] Ontologia possui um significado muito maior, entretanto nesta pesquisa, seu uso restringe-se a identificação de ativos semelhantes dentro do domínio do Reuso.

[28] Tamanhos dos ativos, dado em: FP, UCP ou SLOC.

de desenvolvimento podem ser outros, não existirão Produtos-PLE e nem ferramentas de ontologia para avaliar.

Por outro lado, estas questões podem ser ignoradas, pois os encargos de reuso só existem quando não há a preocupação com a confiabilidade do software, pois tais processos farão parte de qualquer desenvolvimento de um produto espacial, além destas questões se tornarem desprezíveis, frente aos tempos economizados devido ao próprio reuso.

Figura 3.20 – Uso de ferramentas de Ontologia na viabilidade.

Fonte: Produção do autor.

De forma resumida, a Figura 3.20 mostra que as ferramentas ligadas à ontologia devem: (1) localizar *drivers* para a BCD e as funções para a BCI no portfólio de projetos-alvo e (2) retornar os produtos (*drivers* ou funções) que mais se repetem, incluindo seus tempos já contabilizados quando foram desenvolvidos. Assim, se conheçará a probabilidade de um determinado produto reincidir num próximo projeto, e a probabilidade de economia em esforços (tempo), que este produto trará ao projeto. Estas probabilidades indicam portanto, a viabilidade econômica para um futuro Produto-PLE.

3.5.2. Modelo e métricas para a viabilidade econômica

A probabilidade de um determinado produto “K” reincidir num próximo projeto-alvo, pode ser dada pelas relações:

a) Seja Q_k, a quantidade total (em número absoluto) do produto “K” (*driver* ou função) encontrado dentro do portfólio.

b) Seja Q_t, a quantidade total (em número absoluto) de produtos-alvo, onde empregou-se ou não, o produto “K”.

A equação 3.1:

$$A = \frac{Q_k}{Q_t} \tag{3.1}$$

Mostra, baseado em números absolutos, o quanto um produto “K” representa no portfólio, ou indiretamente a probabilidade dele estar nos próximos projetos-alvo.

De outra forma, a probabilidade de economia em esforços, pode ser dada pelas relações :

a) Seja T_{ki}, o tamanho do produto “K”, dado em *Function Points* (FP), *Use Case Points*(UCP) ou Source Line Of Code(SLOCs), usado no projeto-alvo$_i$ de um portfólio.

b) Seja T_i, o tamanho total , dado em FP, UCP ou SLOCs do projeto-alvo$_i$ de um portfólio, uma vez que:

$$i =]0; n] \tag{3.2}$$

Representa o conjunto de todos os projetos-alvo do portfólio, que possuem o produto “K” e “n” é número total, destes projetos-alvo no mesmo portfólio.

A equação 3.3:

$$B_i = \frac{T_{ki}}{T_i} \quad (3.3)$$

Mostra, o quanto em tamanho, o produto “K” ocupa dentro do projeto-alvo$_i$.

E completando, a equação 3.4:

$$C = \frac{\sum_{i=1}^{n}(B_i)}{n} \quad (3.4)$$

Mostra, a média aritmética de ocupação em tamanho do produto “K” nos projetos-alvo que o utilizam. Ou seja, é possível inferir que, “A%” dos projetos-alvo terão o produto “K”, e este ocupará, “C%” do total deste projeto.

Para finalizar, uma vez conhecidas as taxas de esforços correntes de uma equipe em: horas/FP, ou horas/UCP ou horas/SLOC, quando multiplicadas por:

$$D = \frac{\sum_{i=1}^{n}(T_{ki})}{n} \quad (3.5)$$

“D”, que é dado em FP, UCP ou SLOC, pode-se pensar numa probabilidade (Z) de ganho de produtividade que um projeto-alvo terá, empregando o produto “K”, uma vez que, algumas horas seriam economizadas, através da redução do escopo.

Com isso, a viabilidade é dada por esses valores de probabilidades: A%, C% e Z, que se complementam. Quando da existência de critérios, como A > a, C > c e Z > z, onde a, c e z são valores (*set-points*) de probabilidades, considerados pela gestão como suficientes para a tomada de decisão, em eleger o produto “K” um Produto-PLE ou K-PLE, então se dará inicio à fase do desenvolvimento com maior confiança, atributo este, alvo do estágio de conceituação no ciclo de vida do produto (Figura 3.2).

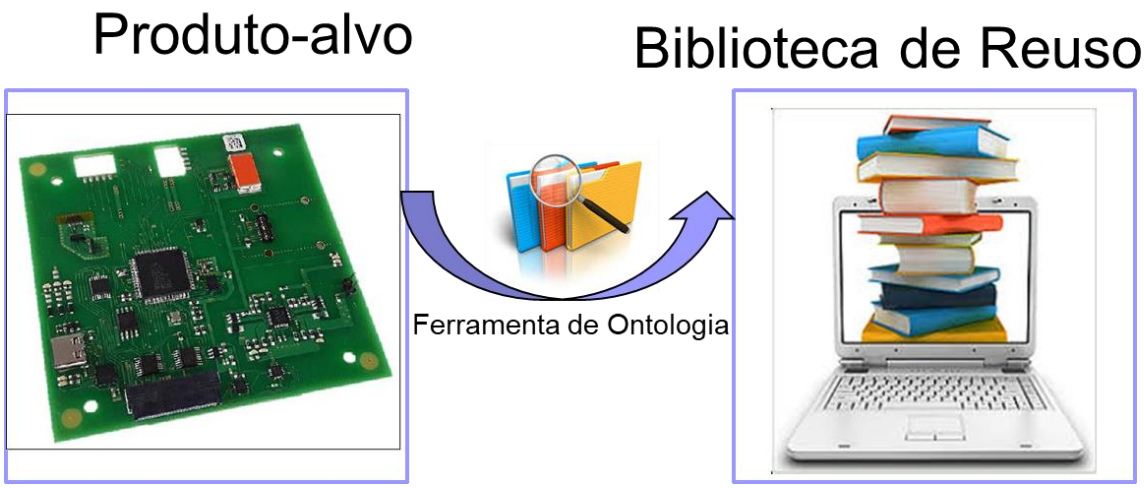
3.6. Índice de reusabilidade dos Produtos-PLE

O índice de reuso, relaciona o quanto os Produtos-PLE são usados nos projetos-alvo. Isso é importante não só para mostrar a efetividade da biblioteca,

mas, principalmente a confiança que ela traz. O ganho em produtividade deixa de ser provável, como na viabilidade, e passa a ser certo.

A contextualização da Figura 3.21, pode ser feita da seguinte forma no estágio de desenvolvimento do projeto-alvo, o *designer* de Hardware deseja, que a ferramenta de Ontologia, procure todos os microcontroladores que contenham os periféricos indicados no documento de arquitetura do Hardware daquele projeto-alvo na BCD. Desta forma, o *designer* poderá priorizar sua escolha de tecnologias (fabricantes e fornecedores) iniciando pelos *drivers* [29], que já estão na biblioteca.

Figura 3.21 – Uso de ferramentas de Ontologia no índice de reusabilidade.



Fonte: Produção do autor.

Isto servirá também para o desenvolvedor de software do projeto-alvo, pois ele usará os respectivos *drivers* da BCD. Semelhantemente, uma analogia pode ser feita na camada intermediária, onde a ferramenta localiza todas as funções de middleware disponíveis na BCI, que são necessárias ao projeto-alvo, segundo os requisitos de nível 3 (Figura 3.7), do mesmo.

[29] Através dos *drivers* da BCD é possível conhecer os periféricos de hardware e os microcontroladores respectivos que compõem a BCD.

Este modelo de análise trata de um outro uso das ferramentas de Ontologia, onde o alvo de busca de similaridades não está mais na base de projetos, mas sim na própria biblioteca (Figura 3.21). Ele mostra como o índice de reusabilidade, aumenta naturalmente em função do reuso dos mesmos componentes da biblioteca, sejam eles, *drivers* ou funções.

4. ESTUDOS DE CASOS

Os estudos de caso apresentados a seguir, possuem objetivos quanto à aplicação das propostas vistas no Cap.3. O primeiro mostra uma aplicação do Produto-PLE para firmware, de acordo com o projeto desenvolvido no Cap. 3.3, O seguinte, apresenta uma aplicação do modelo de viabilidade da economia de acordo com Cap. 3.5.

4.1. Estudo de Caso para um reuso de firmware, através do componente "CLOCK" em dois nanosatélites

Uma aplicação direta do reuso está no *nanosat* AESP-14 (ERENO, 2014) e no *picosat* UbatubaSat (DOS SANTOS, 2014). Ambos, são projetos que compartilham da mesma plataforma, neste caso o AT90S8535, um microcontrolador de 8-bits de baixo consumo baseado na arquitetura AVR RISC.

Uma vez que a plataforma de processamento é a mesma, o compartilhamento dos *drivers* é de forma direta. Para isso, deve-se levantar seus requisitos de nível 4, trazendo as similaridades para os Produtos-PLE desenvolvidos segundo o padrão da fábrica abstrata.

O *cubesat*-1U Trailblazer (KIEF, 2011) é a prova de aplicação do conceito da arquitetura *Plug-and-play (Space Plug-and-play Architecture* - SPA), o qual contém uma série de módulos de interface padrão (*Applique Sensor Interface Module* – ASIM), que permitem a construção de um satélite inteiro.

Este *cubesat* foi construído através de uma arquitetura distribuída, onde os seus componentes são do tipo COTS. Tais componentes são módulos ASIM, os quais padronizam a rede de comunicação e os serviços prestados (SPA *Services Manager*).

O Trailblazer possui o ASIM baseado no microcontrolador PIC24FJ256GA110 e em uma rede I2C, no caso do módulo C&DH, ele está baseado em um COTS da Pumpkin Corporation, que o divide em duas partes, uma *motherboard*

padrão, e em um módulo de processador plugado *(Pluggable Processor Module* – PPM).

Figura 4.1 – *Módulo C&DH do Cubesat-1U* Trailblazer *(Arduino Based SDM-Lite - Single Board Computer Motherboard for Harsh Environments).*

Fonte: KIEF (2011); PUMPKIN (2009).

Figura 4.2 – *Módulo C&DH do Cubesat-1U Trailblazer (PPM with Microchip® PIC24 for CubeSat Kit Motherboard).*

Fonte: KIEF (2011); PUMPKIN (2010).

O microcontrolador PIC24 de 16-*bits* possui arquitetura Microchip RISC.

4.1.1. Os requisitos, segundo a IE

Ambas as famílias de microcontroladores, o AT90S8535 e o *PIC*24, foram usadas em projetos espaciais e implementaram o mesmo módulo, C&DH. Para a demonstração do reuso usa-se, como exemplo, o periférico *TIMER*, pois trata-se de um periférico comum, não só para estes microcontroladores, como na maioria deles, trata-se de um periférico que implementa funcionalidades ligadas ao controle do tempo do C&DH, como por exemplo:

- TC *sync*.: Comandos recebidos da Terra podem ser sincronizados em função do relógio;

- *Schedule*: Comandos para a Terra ou para outros módulos poderão ser agendados;

- *Date-time*: Registros ou *logs* do momento em que os comandos e os resultados chegaram ou foram enviados;

- *Deadline tasks*: Controle da latência na execução de tarefas do C&DH; e

- *Deadline tasks*: Manter o tempo rastreado. Isto geralmente é utilizado pelo *watchdog timer*. O *watchdog timer*, garante que o microcontrolador opera normalmente. O microcontrolador, deveria reiniciar o *watchdog timer* caso as instruções sejam executadas normalmente, pois senão, o *timer* alcançará um valor crítico, significando que alguma coisa provavelmente está errada, e isto causará o reinicio do microcontrolador.

Segundo a IE (Figura 3.7), existem funções ligadas à missão desses satélites (AESP-14 , UbatubaSat *e* Trailblazer), de nível 3, e que são realizadas pelos *drivers* no nível 4. A Tabela 4.1 sintetiza tais funções e *drivers*, relacionando-os, de acordo com o modelo IE na coluna *System and Components.*

Tabela 4.1 – Funções e *Drivers* segundo a IE para *System and Components*.

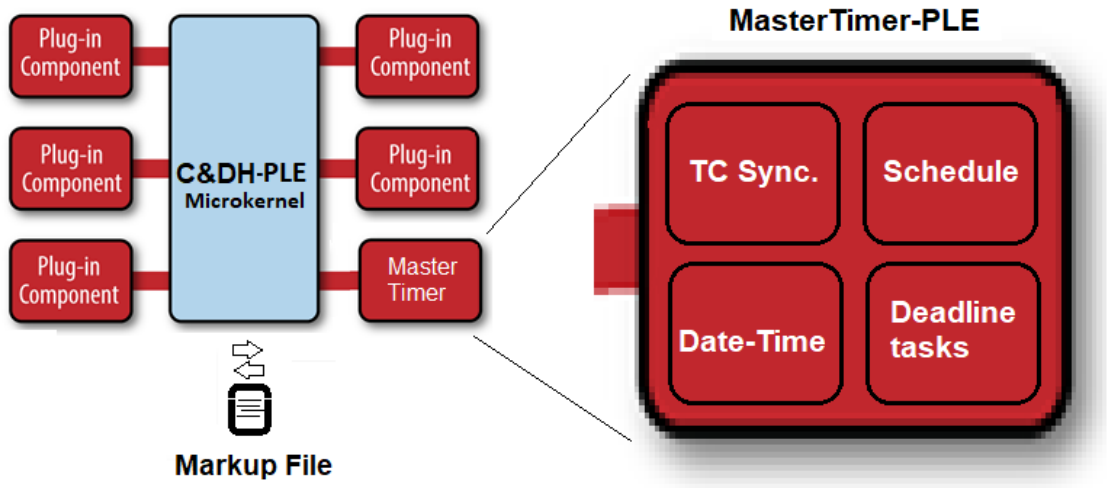
Level 3 (*orientado a missão*) – Decomposição das funções de tempo do C&DH.	TC sync.	Schedule	Date-time	Deadline tasks
Level 4 (*orientado ao hardware*)– Funções no nível de drivers.	Clock	Clock	Clock	Counter

Fonte: Produção do autor.

4.1.2. Arquitetura do Produto-PLE

As funções do nível 3 podem ser encapsuladas em um único Produto-PLE para a BCI, visto que, proporciona serviços comuns ligados às funções de tempo dos nanosatélites, aqui chamados de MasterTimer-PLE.

Figura 4.3 – Funções para a BCI do MasterTimer-PLE.



Fonte: Produção do autor.

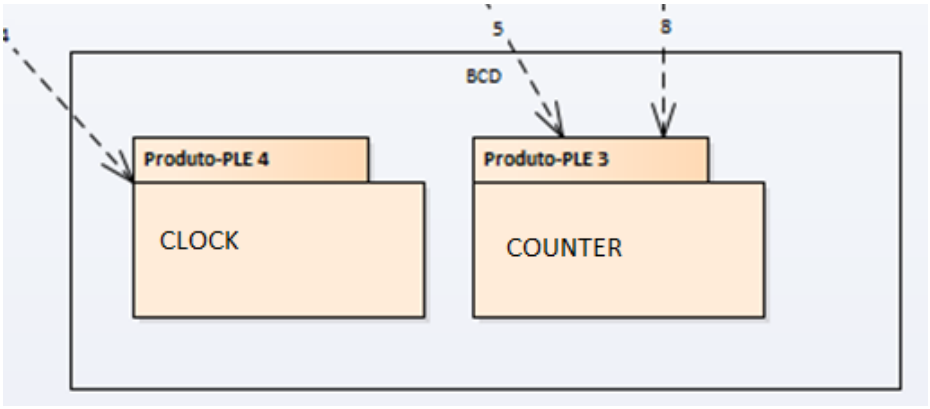
Tanto o AESP-14 quanto o Ubatubasat possuem arquiteturas centralizadas, pois, a lógica de todo o satélite é feita por um único microcontrolador, diferentemente da arquitetura do Trailblazer que é federada, possuindo um microcontrolador dedicado ao C&DH.

Estas diferenças de arquitetura não implicam em arquiteturas de reuso diferentes. Conforme a Tabela 4.1 mostra, tais satélites deverão compartilhar do mesmo MasterTimer-PLE na BCI, *Clock*-PLE e *Counter*-PLE na BCD.

O MasterTimer-PLE é um *plugin* do C&DH-PLE, que presta 4 serviços, contudo está fora do escopo deste estudo de caso, pois seria necessário expandir suas similaridades entre os satélites aqui explorados e neste caso, foi escolhido somente o *timer*.

Com o objetivo de elaborar o *design* do MasterTimer-PLE, será realizado os *drivers* sugeridos conforme o nível 4 da Tabela 4.1. Os *drivers* farão uso do periférico *Timer* [30] do microcontrolador e nele, será feita a configuração necessária para que o *Timer* possa operar como *Clock* e *Counter*, que são exemplos de formas genéricas e possíveis de operar um. Fazendo a analogia com a Figura 3.10, a Figura 4.4 mostra o Produto-PLE4 = *Clock* e o Produto-PLE3 = *Counter* da BCD.

Figura 4.4 – Exemplo de repositório BCD.



Fonte: Produção do autor.

A Figura 4.6 mostra o *design* da BCD, através da aplicação do padrão fábrica-abstrata. O PLE-*Clock* foi melhor detalhado para contribuir com o entendimento do Leitor.

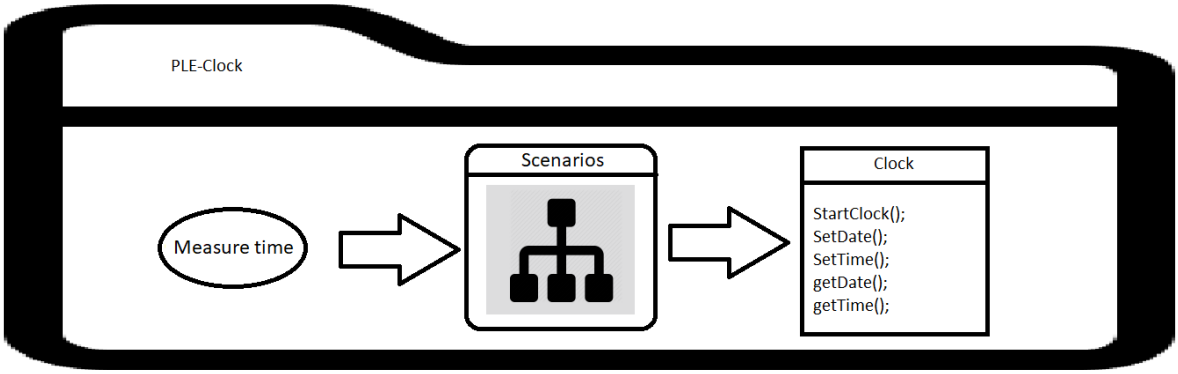
4.1.3. O *design* da BCD

Pode-se entender que o PLE-*Clock* é um plugin do microcontrolador abstrato (*microkernel*). De acordo com a Figura 4.5, são mostrados através de um resumo de modelos orientados pelos processos escolhidos pela instituição, alguns dos principais serviços externos providos pelo PLE-*Clock*.

[30] Existe uma generalização, como se o periférico *TIMER* fosse o único responsável pelas atribuições de *Clock* e *Counter* porém, dependendo do fabricante poderá existir periféricos dedicados à estas atribuições evitando, portanto, o uso do *TIMER*. Esse é o caso do PIC24FJ256GA110 que possui o RTCC responsável pelo *Clock.* Mas este é exatamente o papel da BCD e consequentemente desta PLE: abstrair tais diferenças ,e prover uma interface comum para o usuário.

A classe abstrata “*abstractMicrocontroller*” na Figura 4.6 permite criar qualquer microcontrolador, mas neste caso tem-se apenas 2 definidos nas classes concretas “PIC24FJ256GA110” e “AT90S8535”. A idéia deste conjunto é conter a declaração dos *drivers* do microcontrolador.

Figura 4.5 – *PLE-Clock.*



Fonte: Produção do autor.

Os produtos abstratos são definidos pelas interfaces: *Clock* e *Counter*. No caso do *Clock* abstrato, declara-se os métodos conforme as Figuras 4.5 e 4.6.

Focando no Clock, a Fábrica Abstrata, “*abstractMicrocontroller*” é uma classe abstrata. Ela declara o método abstrato ***`useClock(),`*** este método é responsável por retornar um objeto do tipo *AbstractClock*, cujo o papel é prover todas as funções deste *driver.* Mas ele não é implementado nesta classe; a implementação fica por conta das classes “PIC24FJ256GA110” e “AT90S8535”.

Nesta classe está declarado também, o método estático ***`createConcreteMicrocontroller().`*** O seu trabalho é verificar, em um arquivo de configuração, qual é o *Clock* ativo no aplicativo. Se for o do PIC24F, irá criar e retornar uma instância de PIC24FJ256GA110; se for o do AT90S, irá criar e retornar uma instância de AT90S8535.

A classe PIC24FJ256GA110 implementa o método ***`useClock().`*** O método cria um objeto do tipo PIC*Clock* e retorna este objeto como do tipo interface *AbstractClock*. De forma análoga, ocorre com a classe AT90S8535.

AbstractClock é uma interface, e declara todos os métodos do *driver*. Eles são implementados pelas classes PIC*Clock* e AT90S*Clock*.

PIC*Clock* é uma classe que implementa a interface *AbstractClock*, logo, implementa seus métodos que representam o *drive* para o *Clock* do microcontrolador PIC24FJ256GA110. De forma análoga, ocorre com a classe AT90Sclock para o microcontrolador AT90S8535.

O usuário utiliza a infraestrutura do *driver* projetada através do padrão *Abstract Factory.*

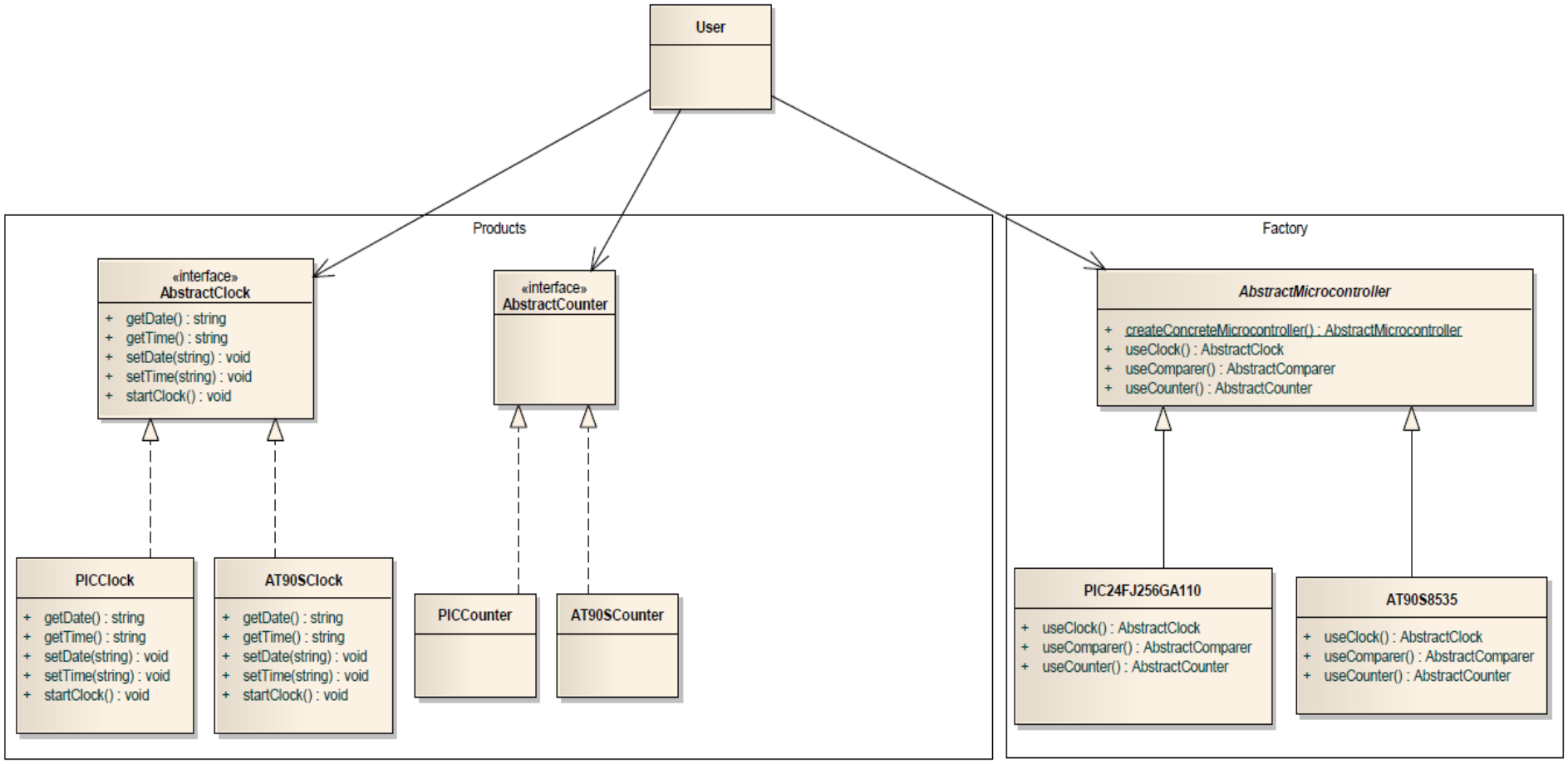
Um exemplo de utilização é:

```
AbstractMicrocontroller.createConcreteMicrocontroller().useClock().startClock();
```

Assim, o arquivo de configuração dirá se as camadas superiores BCI ou Aplicação, usarão o PIC ou o AT90S, porém, de forma desacoplada, todo o conjunto de métodos que descreve o *Driver Clock* seja para o PIC ou AT90S, fica disponível de maneira idêntica para o usuário.

E este é o benefício do padrão fábrica-abstrata para a BCD. Nele, é permitido se somar novos controladores e/ou novas funcionalidades para os *drivers,* de forma desacoplada, fazendo com que a BCI ou o próprio projeto-alvo possam usufruir sempre da mesma forma. Tem-se portanto, um PLE com o cofigurador de perfis voltado para a BCD, conforme exposto inicialmente no Cap. 2.2.3 (pág. 22)**.**

Figura 4.6 – *Design da BCD com 2 Drivers e 2 Microcontroladores.*



Fonte: Produção do autor.

4.1.4. A flexibilidade do *design* para mais *nanosats*

A flexibilidade deste *design* para novos *nanosats* se dá de duas formas: (1) os novos projetos, podem usufruir dos mesmos microcontroladores já na biblioteca, como o Ubatubasat e o AESP-14 o fizeram, e (2) novos microcontroladores podem ser adicionados no *design.* Desta forma, teria que ser implementado os *drivers,* já existentes no *design,* para que as mesmas interfaces pudessem ser usufruídas.

É importante citar que os testes *whitebox* e *blackbox* devem ser feitos, e como em qualquer outro projeto. Testes de integração, tanto no caso de componentes da BCI, quanto da própria aplicação quando fizerem uso destes *drivers* serão necesários. Isto é especificado na IE (Figura 3.7), através da coluna de V&V.

Assim se encerra o primeiro Estudo de Caso quanto à aplicação do Produto-PLE para o reuso de firmware, em pelo menos 2 *nanosats* com microcontroladores diferentes, representando um dos objetivos desta pesquisa conforme o proposto no Cap 1.6.

4.2. Estudo de Caso para a aplicação da viabilidade econômica

O Estudo de Caso apresentado é um modelo de aplicação da viabilidade econômica, vista no Cap. 3.5. A intenção, é aplicar as equações 3.1, 3.3, 3.4 e 3.5, mostrando ao final os dados estatísticos da viabilidade, que contribuirão com o Gestor na sua tomada de decisão, quanto ao desenvolvimento ou não de um Produto-PLE.

A viabilidade deve se basear nos dados históricos, visto, que o Gestor precisa saber de antemão, se ele deve ou não investir em um dado *driver* ou função, alvos da viabilidade e candidatos a Produtos-PLE.

O Estudo de Caso, é uma exemplificação fundada em três formas de se expressar a viabilidade. (1) O conjunto de aplicativos desenvolvidos pela instituição; a primeira abordagem, é saber quantos deles possuem um determinado *driver* ou determinada função, ou seja, saber o quanto estes

ativos de fato, participam ou não dos aplicativos desenvolvidos pela instituição.

(2) Uma segunda abordagem, é focar no conjunto de todos os aplicativos onde consta o determinado *driver* de interesse, ou função de interesse quanto aos seus tamanhos. Toma-se o tamanho total do aplicativo e dos ativos (*drivers* ou funções) escolhidos de forma separada, com isso, pode-se pensar que um Produto-PLE representaria uma média destes tamanhos, quando usados em aplicativos futuros; e

(3) A terceira, e última forma, é o ganho de produtividade devido ao reuso. Nesta, usa-se da taxa de esforços correntes da equipe, dadas em horas/FP, ou horas/UCP ou horas/SLOC (BAUER, 2009), para mostrar a quantidade em horas, que os ativos (*drivers* ou funções) escolhidos economizariam nos aplicativos futuros.

É importante salientar que, o uso de ferramentas ligadas à ontologia, fazendo um uso restrito do termo, são fundamentais para filtrar tais similaridades tanto de *drivers*, quanto de funções. Uma busca na base de requisitos de cada projeto é o começo. Trata-se do uso das ferramentas ontológicas para os ativos de software da base dos projetos-alvo, ou seja, ainda não viraram Produtos-PLE e portanto, seriam candidatos mensuráveis através dos resultados desta viabilidade.

4.2.1. Resultados relativos à viabilidade

Um exemplo ilustrativo para a aplicação, é baseado na Tabela 4.2, onde dados históricos hipotéticos de um conjunto de oito C&DHs para *nanosats,* representam todo o conjunto disponível da instituição até um dado momento.

Para cada um dos C&DH, foi extraído o seu tamanho total, bem como o tamanho relacionado às funções de *clock* incluindo, a configuração e a inicialização do *timer* do microcontrolador, que foram contabilizados

separadamente. Tais dados estão em *Use Case Point* (UCP) (BAUER, 2009).

Tabela 4.2 –UCP do conjunto de C&DH de *nanosats* de uma instituição

	C&DH1	C&DH2	C&DH3	C&DH4	C&DH5	C&DH6	C&DH7	C&DH8
UCP Total	50	100	80	45	50	20	400	220
UCP Clock	10	20			8	3		25

Fonte: Produção do autor.

Pela primeira abordagem observa-se que, o *driver "Clock"* foi utilizado em cinco aplicativos dos oito C&DH, ou seja, de acordo com a Equação 3.1 (Cap. 3.5.2):

$$A = \frac{5}{8} \equiv 62{,}5\%$$

62.5% dos aplicativos contém *Clock*, logo, investir recursos num PLE-*Clock,* tem uma probabilidade de 62,5% de ser utilizado em aplicações futuras. Quanto mais as aplicações pertecem a produtos de um mesmo portfólio (que possuem similaridades, como C&DH de nanosatélites), maior é a confiança deste valor para a viabilidade.

A segunda abordagem, inicia-se pela aplicação da Equação 3.3 (Cap. 3.5.2), mostrada na Tabela 4.3, onde observa-se as ocupações (B_i) do *clock* no C&DH como um todo.

Com isso, aplicando a Equação 3.4 (Cap. 3.5.2):

$$C = \frac{\sum_{i=1}^{n}(B_i)}{n} \equiv 16{,}4\%$$

Em média, o *clock* ocupa 16,4% do tamanho total de um C&DH, e é claro, que este número varia de acordo com o número de componentes da aplicação, e de seus respectivos tamanhos ou complexidade. Logo, quanto maior forem as similaridades, mais realista será tal estimativa.

Tabela 4.3 – Ocupação do *clock* nos C&DH de cada *nanosat*

	C&DH1	C&DH2	C&DH3	C&DH4	C&DH5	C&DH6	C&DH7	C&DH8
UCP Total	50	100	80	45	50	20	400	220
UCP Clock	10	20			8	3		25
$B_i = \frac{T_{ki}}{T_i}$	20%	20%			16%	15%		11%

Fonte: Produção do autor.

Como última análise à viabilidade, é apresentado o ganho de produtividade do reuso, que o *Clock*-PLE poderá conferir ao projeto-alvo. Em média, de acordo com a Tabela 4.3 e a Equação 3.5 (Cap 3.5.2):

$$D = \frac{\sum_{i=1}^{n}(T_{ki})}{n} = 13 \text{ UCPs}$$

Um *clock* possui o tamanho de 13 UCPs, e uma vez conhecido o esforço médio corrente da equipe, como por exemplo, 3h/UCP, o *clock* levaria em média 39 horas para ser desenvolvido. Caso tal desenvolvimento fizesse uso de um equipe focada em reuso, o primeiro C&DH pagaria tal conta, mas, para os demais, essas 39 horas seriam revertidas em ganho de produtividade, já que o esperado, é que o *clock* seja utlizado em mais de 60% dos C&DHs futuros.

Sabe-se que a produtividade depende da equipe, logo, quanto mais estável for a equipe de desenvolvimento, maior será a confiança nestes valores.

Os cálculos e procedimentos, para levantar tais tamanhos, relacionando-os com a produtividade podem ser vistos em maiores detalhes em Bauer (2009).

Este estudo encerra, a aplicação dos modelos de viabilidade econômica, declarado no Cap. 3.5 e apresentado com um dos objetivos desta pesquisa no Cap. 1.6.

Com isso, se encerra a apresentação geral dos objetivos propostos para esta pesquisa, sendo os principais resultados e contribuições sumarizados no próximo capítulo.

5. CONCLUSÃO

Neste capítulo final, será apresentado as contribuições da pesquisa proposta, suas limitações, as recomendações e sugestões para trabalhos futuros, e alguns comentários finais.

5.1. Contribuições desta pesquisa

Esta pesquisa apresentou táticas de reuso, visando lidar com os problemas de recorrência na engenharia, e *time-to-market,* relacionado ao domínio do software, enfatizando, aqueles que demandam alta confiabilidade como os de bordo espacial.

Foram apresentadas, algumas táticas para promover a reusabilidade nos projetos de software, trazendo como fundamento o estado-da-arte das abordagens, técnicas, e processos conhecidos que alavancam o reuso do software.

A pesquisa teve como objetivos, apresentar como se dá, o emprego dos componentes reutilizáveis nos projetos de *firmware,* e *midleware* em nanosatélites e focando em um reuso planejado, a fim de diminuir acidentes e problemas com os projetos que o aplicam, uma vez, que foi identificado como sua maior deficiência.

Apesar da pesquisa nascer para uma abordagem de software que possa ser embarcado em ambientes espaciais, nas táticas de reuso apresentadas, também é possível trabalhar com níveis de maturidade diferentes, e ao mesmo tempo com software com níveis de criticidade também diferentes.

5.1.1. Contribuições principais

A pesquisa apresentou, o Produto-PLE na forma estrutural de componente de software orientado à serviços, como o resultado do Reuso. Sendo ele, um produto resultante das etapas do processo de

desenvolvimento focado em: Requisitos, Arquitetura e *Design*, devido à pressão em propiciar um reuso planejado

O Produto-PLE foi construído, a fim de contribuir com as camadas de *firmware* e *middleware* por elas apresentarem o maior índice de similaridades entre os projetos de nanosatélites. Logo, foi apresentado o emprego destes produtos reutilizáveis, em módulos de nanosatélites de arquiteturas centralizadas e distribuídas, com e sem Sistema Operacional, além dos módulos de *middleware* para os *Spacecraft Onboard Interface Services,* a fim de mostrar a sua aplicabilidade, nas maiores diferenças, em termos de arquitetura, entre os *nanosats*.

Um Estudo de Caso de aplicação do Produto-PLE para *firmware* foi apresentado, e nele, mostrou-se o reuso do *driver 'clock'* em 2 nanosatélites, com microcontroladores diferentes, sendo um de arquitetura distribuída e o outro, centralizada. Isso foi possível graças ao *design* apresentado, que permitiu, a agregação de novos microcontroladores, além de novos *drivers* de forma desacoplada, fornecendo uma interface padrão às camadas superiores de *middleware* e/ou *application*.

Outra contribuição está relacionada à viabilidade do reuso, no sentido de conhecer os possíveis ganhos em escopo, e produtividade que o projeto-alvo (aquele que o aplica) terá, caso os componentes em prol do reuso, sejam desenvolvidos.

O modelo de viabilidade econômico apresentado, é baseado em métricas, para a medição dos tamanhos dos projetos de software, na relação de esforços do time, além do uso de ferramentas ligadas à ontologia[31], a fim de descobrir as possíveis similaridades dos projetos-alvo, e com isso permite-se o cálculo da viabilidade de futuros Produtos-PLE.

[31] Ontologia possui um significado muito maior, entretanto nesta pesquisa, seu uso restringe-se à identificação de ativos semelhantes, dentro do domínio do Reuso.

Um Estudo de caso de exemplificação do uso do modelo de viabilidade, mostrou que, em uma instituição hipotética, com um determinado número de projetos de *Command and Data Handling* (C&DH), existe uma probabilidade de mais de 60% dos futuros projetos de C&DH, terem o *driver 'clock'*, onde uma vez aplicado, traria uma economia de 13 *Use Case Points*, que naquela equipe, refletiu uma econômia de 39 horas em cada projeto de C&DH.

5.1.2. Contribuições adicionais e complementares

- O *design* apresentado do Produto-PLE, quando usado em nanosatélites com o mesmo microcontrolador, ou em módulos que compartilham da mesma plataforma de procesamento, podem compartilhar dos *drivers* através das mesmas interfaces, como mostrado no Ubatubasat e o AESP-14;

- Foi mostrado um cenário de implantação do reuso com equipes distintas, para *drivers* e *middleware;*

- Foi mostrado que, em prol do aumento do Potêncial de Reuso, poderá ser feito a separação da equipe de desenvolvimento em: equipes de reuso e equipes de produtos.

- A pesquisa mostrou um índice de 70% de aderência aos processos (de forma total, ou parcial) das capacidades de um modelo de referência para reuso de software, que foi baseado e atualizado no MPS-Br 2016.

- O índice de reusabilidade, mostrou uma forma de mensurar a efetividade de cada produto reutilizável. Desta forma, serve como incentivo aos projetistas, em usufruir cada vez mais das bibliotecas.

- A pesquisa introduziu, o uso de Níveis de Maturidade ou de Prontidão, como o *Reuse Readiness Levels* da NASA, a fim de conhecer, o quanto determinado componente reutilizável já foi

aplicado e testado, inclusive em ambientes operacionais idênticos às possíveis futuras aplicações. Com isso, faz-se a avaliação dos riscos em aplicar tais produtos reutilizáveis no projeto, bem como, a necessidade da execução de testes a fim de mitigá-los.

- A pesquisa considerou a certificação dos componentes reutilizáveis, e para isso apresentou o programa *Reusable Software Component* (RSC) da *Federal Aviation Administration* (FAA) como uma proposta.

Pela revisão bibliográfica percebe-se, uma lacuna deixada pelas agências espaciais com relação ao "como" realizar o Reuso, fala-se sobre seus resultados de implementação, em termos de componentes e arquiteturas reutilizáveis, fala-se sobre os benefícios que trouxeram, entretanto, não é falado sobre sua construção, sobre como atingir os níveis de flexibilidade, sobre como descobrir as similaridades em um portfólio, sobre como mensurar a viabilidade do investimento, e como operacionalizá-lo numa equipe.

Logo, de forma geral, a pesquisa contou com vários conceitos da Engenharia de Software relacionados a: processos de desenvolvimento; requisitos; arquitetura; e *design*, onde foram organizados em prol do reuso. E a partir disto, construiu-se componentes reutilizáveis que são aplicáveis em sistemas espaciais de bordo nas camadas de *firmware* e *middleware*, além de um modelo de viabilidade para estimar o investimento do desenvolvimento destes componentes.

5.2. Limitações desta pesquisa

Observa-se, como uma limitação desta pesquisa, a ausência de dados práticos, a fim, de embasar o estudo de caso ligado à viabilidade quanto à produtividade. Isto poderia ser corrigido através de um '*CASE'* de sucesso, inclusive, foi sugerido como trabalho futuro, contudo se reconhece como uma grande valia e limitação para este trabalho, pois também, seria possível a partir dele, mensurar questões relacionadas ao

custo do reuso, como: (1) o custo do reuso da integração dos Produtos-PLE com o Projeto-alvo; (2) custo da elaboração do próprio Produto-PLE dentro do processo proposto; e (3) custo do uso das ferramentas ligadas à ontologia[32].

5.3. Trabalhos futuros

Os trabalhos futuros, servem de incentivo aos pesquisadores que desejarem continuar este trabalho ou complementá-lo.

5.3.1. Recomendações

Os tópicos a seguir, detalham possíveis trabalhos que derivaram diretamente deste tema:

- Um detalhamento demonstrando como se dá o relacionamento dos processos de maturidade em GARCIA (2010) - *RiSE Reference Model for Software Reuse Adoption in Brazilian Companies* com a influência da ECSS-Q-HB-80-01A (2011) - *Space product assurance Reuse of existing software*;

- Um estudo sobre a necessidade de uma equipe dedicada ao reuso. Por esta pesquisa, pode-se inferir que o Ganho não Econômico de Reuso seria maior, gerando Potencial de Reuso e isto, afetaria positivamente o Índice de Reusabilidade, fazendo crescer a produtividade. Porém, deve-se considerar que, uma equipe de reuso agregou custo no caso de uma contratação, ou diminuiu a produtividade no caso de realocação das pessoas;

- Uma extensão da pesquisa, quanto ao *design* da Biblioteca de reuso de funções de apoio à missão (BCI), análogo ao mostrado para a Biblioteca de reuso de *Drivers* (BCD);

[32] Ontologia possui um significado muito maior, entretanto nesta pesquisa, seu uso restringe-se à identificação de ativos semelhantes dentro do domínio do Reuso.

- *Cases* de aplicações prática, focando em (1) na viabilidade dos ativos similares; (2) no desenvolvimento dos Produtos-PLE; (3) na aplicação dos Produtos-PLE em projetos de nanosatélites; e (4) nos custos do reuso, que afetam a produtividade; e

- Um estudo que considera a questão da Verificação e da Validação do Produto-PLE. Com o aumento do reuso do mesmo Produto-PLE em contextos diferentes, e somado às suas manutenções, o produto tende a aumentar sua confiabilidade. Porém, deve-se procurar saber, quando um determinado teste de integração e/ou de regressão poderiam ser dispensados.

5.3.2. Sugestões

Os tópicos a seguir detalham possíveis trabalhos que complementam o entendimento desta pesquisa:

- Uma demonstração de como trabalhar com o reuso nas interfaces de comunicação. Produtos-PLE poderiam representar cada camada do modelo ISO/OSI conforme a demanda do projeto; e

- O desenvolvimento, ou a adaptação de ferramentas ligadas à ontologia[33] que contribuem com as análises de viabilidade e o índice de reusabilidade.

5.4. Comentários finais

A pesquisa mostrou meios para iniciar a implantação do reuso do software aplicado a sistemas espaciais de bordo, bem como efetuar estimativas visando o melhor retorno do investimento. Assim acredita-se na contribuição com as instituições, e empresas que procuram uma alternativa para usufruirem dos benefícios do reuso do Software, e ao mesmo tempo, terem uma forma comprobatória de viabilidade.

[33] Ontologia possui um significado muito maior, entretanto nesta pesquisa, seu uso restringe-se à identificação de ativos semelhantes dentro do domínio do Reuso.

REFERÊNCIAS BIBLIOGRÁFICAS

AGUIAR, M. A produtividade dos projetos de Desenvolvimento, **Developers Magazine,** p. 41-42, 2003.Disponível em: <http://www.metricas.com.br/downloads/A_Produtividade_dos_Projetos_de_Desenvolvimento.pdf>. Acesso em: 10 nov. 2017.

ALDAWUD, O.; *et al.* **UML Profile for Aspect-Oriented Software Development**. Disponível em: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.13.4608&rep=rep1&type=pdf> Acesso em: 1 nov. 2016.

BAUER, T. O. **Produtividade no Desenvolvimento de Software,** 2009, 96 p. Trabalho de Conclusão de curso - Centro Universitário Feevale, Novo Hamburgo. Disponível em: <http://fattocs.com/files/pt/livro-apf/citacao/TiagoOtoBauer-2009.pdf>. Acesso em: 5 dez. 2016.

BAYER, J; *et al*. PuLSE: A Methodology to Develop Software Product Lines**,** SYMPOSIUM ON SOFTWARE REUSABILITY (SSR), 1999, Los Angeles. Disponível em: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.560.6937&rep=rep1&type=pdf>. Acesso em 3 nov. 2016.

BELLAMY III, W. FACE Avionics Software Reuse Expands With NAVAIR, **Avionics Today,** 2016. Disponível em: <http://www.aviationtoday.com/2016/01/20/face-avionics-software-reuse-expands-with-navair/>. Acesso em: 1 nov. 2016.

BRUGALI, D.; et al. Service Component Architectures in Robotics: The SCA-Orocos Integration. In: Leveraging Applications of Formal Methods, Verification, and Validation: International Workshops. Vienna, Austria, 2011. p.46-60. Disponível em: < https://www.researchgate.net/publication/248380458_Service_Component_Architectures_in_Robotics_The_SCA-Orocos_Integration>. Acesso em: 12 mar. 2017.

CCSDS. **Sobre o CCSDS.** Disponível em: <https://public.ccsds.org/default.aspx>. Acesso em: 1 nov. 2016.

CCSDS-SOIS. **Report Concerning Space Data System Standards, SPACECRAFT ONBOARD INTERFACE SERVICES**, INFORMATIONAL REPORT, GREEN BOOK, CCSDS 850.0-G-2, 2013. 88p. Disponível em: < https://public.ccsds.org/Pubs/850x0g2.pdf> Acesso em: 15 nov. 2016.

cFS. **THE FIELD OF AEROSPACE.** Disponível em: <https://cfs.gsfc.nasa.gov/Applications.html>. Acesso em: 12 dez. 2017.

CHAPPELL, D. **Introducing SCA**,CHAPPELL & ASSOCIATES, 2007. 22p. Disponível em: <http://www.davidchappell.com/writing/Introducing_SCA.pdf>. Acesso em: 4 jun. 2017.

CMMi-DEV, **Technical Solution (TS) (CMMI-DEV)**. Disponível em: < https://www.wibas.com/cmmi/technical-solution-ts-cmmi-dev>. Acesso em: 2 maio 2017.

CMMI INSTITUTE. Build Capabilities to Win**,** 2015. 27 transparências. Disponível em: < <https://cmmiinstitute.zendesk.com/hc/en-us/article_attachments/205847598/CMMI_Benefits_and_Who_Uses_CMMI_Presentation_2015.pdf>. Acesso em 7 jul. 2017.

CMMI-DEV, V1.3. **CMMI® for Development**, Improving processes for developing better products and services, Version 1.3, 2010. 482p.

CORDET-2. **Activity 3 - Case study.** Disponível em: <http://cordet.gmv.com/activity3.htm>. Acesso em: 10 dez. 2016.

DE ALMEIDA, E. S.; *et al*. **C.R.U.I.S.E Component Reuse in Software Engineering,** C.E.S.A.R, 2007. Disponível em: < https://www.researchgate.net/publication/236660495_CRUISE_Component_Reuse_in_Software_Engineering>. Acesso em: 14 ago 2017.

DE JONGE, M. Developing Product Lines with Third-Party Components, **ScienceDirect - Elsevier**, Electronic Notes in Theoretical Computer Science, v.238, p. 63–80, 2009. Disponível em: < https://ac.els-cdn.com/S1571066109003958/1-s2.0-S1571066109003958-main.pdf?_tid=a8747c0e-4d70-48f4-a7b4-3299d6d1eedf&acdnat=1520473531_ed61a08fba910d6330ce40e9f1915ee8>. Acesso em: 15 abr 2017.

DOS SANTOS, W. A. **Adaptability, Reusability and Variability In Software Systems – A Case Study For Space On-Board Computing.** 2008. 234 p. Thesis of Doctor in Science – Course of Electronic Engineering and Computer Science. Area of Computer Science – Aeronautical Institute of Technology.

DOS SANTOS, W. A.; *et al***.** Workshop sobre Nanosats e suas Aplicações, Demonstração do modelo de Engenharia e Qualificação do UbatubaSat . Escola Tancredo Neves - INPE, 2014. 64 transparências. Disponível em: <http://www3.inpe.br/snct2015/arquivos/INPE_SNC&T_Nanosats_EngenhariaEspacial.pdf>. Acesso em: 15 jun. 2017.

DOUGLASS, B. P. **Real-Time Design Patterns: Robust Scalable Architecture for Real-Time Systems,** 1st Edition, Addison Wesley, 2002. 528 p. ISBN-10(0201699567).

DOUGLASS, B. P. **Real-Time Agility: The Harmony/ESW Method for Real-Time and Embedded Systems Development,** 1st Edition, Addison-Wesley Professional, 2009. 560 p. ISBN-10(0321545494).

DOWNS, R. R.; MARSHALL, J. J. DATA SCIENCE JOURNAL: **A Proposal On Using Reuse Readiness Levels To Measure Software Reusability**, 2010. p.73–92. Disponível em: <http://doi.org/10.2481/dsj.009-007>. Acesso em: 18 set. 2017.

DRD; DDR&E**.** Technology Readiness Assessment (TRA) Deskbook**, Department of Defense,** 2009. Disponível em: <

https://www.skatelescope.org/public/2011-11-18_WBS-SOW_Development_Reference_Documents/DoD_TRA_July_2009_Read_Version.pdf>. Acesso em: 5 set. 2017.

ECSS-Q-HB-80-01A. **Space product assurance Reuse of existing software**, 2011. Disponível em: < http://everyspec.com/ESA/ECSS-Q-HB-80-01A_47792/>. Acesso em 15 mai. 2017.

ECSS-E-ST-70-41C. **Telemetry and telecommand packet utilization**, 2016. Disponível em: http://www.ecss.nl/wp-content/uploads/2016/06/ECSS-E-ST-70-41C15April2016.docx>. Acesso em 15 abr. 2018.

ERENO, D.; RAMOS, L. PESQUISA FAPESP: Pequenos ganham o espaço, v. 219, 2014. p. 17-23. Disponível em: < http://revistapesquisa.fapesp.br/wp-content/uploads/2014/05/016-021_CAPA_nanosatelite_219-NOVO1.pdf >. Acesso em 6 jul. 2017.

EUROPEAN SPACE SOFTWARE REPOSITORY (ESSR)**. Available Projects**. Disponível em: <https://essr.esa.int/>. Acesso em: 14 nov. 2017.

FEDERAL AVIATION ADMINISTRATION (FAA). **AC 20-148:** Reusable Software Components. 2004. 30 p. Disponível em: < https://www.faa.gov/documentLibrary/media/Advisory_Circular/AC_20-148.pdf >. Acesso em 10 nov. 2016.

FIET, P.P; KINNAN, L. **Safety-Critical Software Development for Integrated Modular Avionics**, WindRiver, 2015. 12 p. Disponível em: < https://www.windriver.com/whitepapers/safety-critical-software-development-for-integrated-modular-avionics/wp-safety-critical-software-development-for-integrated-modular-avionics.pdf >. Acesso em: 30 mar. 2018.

FREDERICK, M. NASA Studying the Reuse of Spacecraft Software, **Space com**, 2006. Disponível em: <http://www.space.com/2304-nasa-studying-reuse-spacecraft-software.html>. Acesso em: 10 dez. 2016.

FREEMAN, E.; *et al*. **Head First - Design Patterns.** 1nd Edition, O'Reilly Media, 2004. 638 p. ISBN-10(0596007124).

GARCIA, V. C. **RiSE Reference Model for Software Reuse Adoption in Brazilian Companies,** 2010. 202 p. Tese de Doutorado em Ciência da Computação – Universidade Federal de Pernambuco. Disponível em: < https://repositorio.ufpe.br/bitstream/handle/123456789/1729/arquivo3101_1.pdf?sequence=1&isAllowed=y>. Acesso em 10 out. 2016.

GEORGAKOPOULOS, D.; PAPAZOGLOU, M. P. **Service-Oriented Computing**, Cambridge: The MIT Press, 2009. 30 p. Disponível em: <https://mitpress.mit.edu/sites/default/files/titles/content/9780262072960_sch_0001.pdf>. Acesso em 2 fev. 2017.

GRUBER, T. Ontology. In: LIU, L; ÖZSU, M. T. **Encyclopedia of Database Systems**, Springer-Verlag, 2009. Disponível em: <http://www.dainf.ct.utfpr.edu.br/~tacla/Ontologias/2013/a01-020-Ontology-definitionTomGruber.pdf>. Acesso em 15 mar 2017.

HOLMGREN, B. W. **Software Reusability: a study of why software reuse has not developed into a viable practice in the department of defense**, 1990. 158 p. Master of Science in Systems Management. School of Systems and Logistics of the Air Force Institute of Technology Air University. Disponível em: <http://www.dtic.mil/dtic/tr/fulltext/u2/a229551.pdf>. Acesso em 25 nov. 2016.

INCOSE. **Incose Systems Engineering Handbook: A Guide for System Life Cycle Processes and Activities**, 4th Edition,Wiley, 2015. 304 p. ISBN-10 (1118999401).

ISO / IEC / IEEE. **Std 12207:** Systems and software engineering — Software life cycle processes. 2008. 122 p.

ISO / IEC / IEEE. **Std 15288:** Systems and software engineering — System life cycle processes. 2008. 70 p.

ISO / IEC. **25010**: Systems and software engineering: Systems and software Quality Requirements and Evaluation (SQuaRE) - System and software quality models. 2011. 34 p.

JACOBSON, I. Reuse in Reality: The Reuse-Driven Software Engineering Business. Rational Software Corporation. 2800, San Tomas Expressway, Santa Clara, CA 95051.1996. 25 transparências. Disponível em: < http://community.wvu.edu/~hhammar/rts/adv%20rts/objex_ivar%20reuse%20presentation.pdf>. Acesso em 4 abr 2017.

JACOBSON, I.; *et al*. **Software Reuse: Architecture, Process and Organization for Business Success.** 1st Edition, Addison-Wesley Professional, 1997. 528 p. ISBN-10(0201924765).

JONES, M. Software Engineering: Are we getting better at it?**, ESA Directorate of Operations and Infrastructure**, v. 121, p. 52-57, 2005. Disponível em: <http://www.esa.int/esapub/bulletin/bulletin121/bul121g_jones.pdf>. Acesso em 12 dez. 2016.

KHANNA, V.; DEWALT, M. Reusable Software Component(RSC) reality show: Reusable SW components (RSC) in real life. Certification Services, Inc. and Federal Aviation Administration, Conference Norfolk, VA. 2005. 19 transparências.

KIEF, C.J.; Zufelt, B. Trailblazer, SPA and Radiation Testing Overview for CubeSat Workshop, 2011. 11 transparências. Disponível em: <http://mstl.atl.calpoly.edu/~workshop/archive/2011/Summer/24-Kief-Trailblazer.pdf>. Acesso em 2 dez. 2016.

KLAUS, P.; *et al*. **Software Product Line Engineering: Foundations, Principles and Techniques.** 2005 Edition, Springer Science & Business Media, 2005. 467 p. ISBN-10(3540243720).

KOSKELA, M.; et al. **Software development methods: SOA vs. CBD, OO and AOP**, Helsinki University of Technology, 2007. 16 p. Disponível

em:<http://www.soberit.tkk.fi/T-86/T-86.5165/2007/final_koskela_rahikainen_wan.pdf>. Acesso em 25 jun. 2017.

LARSON, W. J.; WERTZ, J. R. **Space Mission Analysis and Design**, 3rd Edition, Microcosm Press, 1999. 504 p. ISBN(1-881883-10-8).

LEONARDI, S. **NASA Goddard Releases Open Source Core Flight Software System Application Suite to Public**, NASA's Goddard Space Flight Center, Greenbelt, Maryland, 2015. Disponível em: <https://www.nasa.gov/press/goddard/2015/march/nasa-goddard-releases-open-source-core-flight-software-system-application-suite-to>. Acesso em: 4 Nov. 2016.

LEVESON, N. G. Intent Specifications: An Approach to Building Human-Centered Specifications**. Journal IEEE Transactions on Software Engineering** , v. 26 , n.1, p. 15-35, 2000. Disponível em: < http://sunnyday.mit.edu/16.355/levesonintent.pdf>. Acesso em 20 ago. 2017.

LEVESON, N.; TURNER, C. The Therac 25 - A case study in safety failure**, IEEE Computer**, v. 26, n. 7, p. 18-41, 1993. Disponível em: < https://pdfs.semanticscholar.org/7d7e/d3d44850d00c548eae45f2059a0f5e418956.pdf>. Acesso em 8 set. 2017.

LIVSCHITZ, J.; BLOMMESTIJN, R. OSSR ESA Open Source Software Repository, ESA, 2014. 13 transparências. Disponível em: < https://indico.esa.int/indico/event/53/session/8/contribution/37/material/1/0.pdf>. Acesso em 10 out. 2017.

LYKE, J.; CALIXTE-ROSENGREN, J. Overview of the Nanosatellite And Plug-and play Architecture (NAPA) Project Agreement, FMV / AAC Microtec, 2010. 34 transparências.

MCGREGOR, J. D. Software Product Lines**. Journal of Object Technology**, v. 3 , n.3, p. 65-74, 2004. Disponível em: <

http://www.jot.fm/issues/issue_2004_03/column6/column6.pdf>. Acesso em 15 abr. 2018.

MAXIM, B.R. Software Reuse and Component-Based Software Engineering, University of Michigan-Dearborn - Lecture Notes, 2013. 47 transparências. Disponível em: <groups.umd.umich.edu/cis/course.des/cis376/ppt/lec22.ppt>. Acesso em 10 out. 2016.

MR-MPS-SW**. Melhoria de Processo do Software Brasileiro MPS-BR,** SOFTEX, Versão 2016, 2016. 57 p. ISBN(978-85-99334-84-3).

MYERS, G. J.; SANDLER, C. **The Art of Software Testing**, 2nd Edition, John Wiley & Sons, Inc., Hoboken, New Jersey. 2004. 151 p. ISBN (0-471-46912-2).

NASA TRL. **Technology Readiness Level,** 2017, National Aeronautics and Space Administration. Disponível em: https://www.nasa.gov/directorates/heo/scan/engineering/technology/txt_accordion1.html>. Acesso em: 10 out 2017.

NASA/TP–2014–216648. **Small Spacecraft Technology State of the Art.** Mission Design Division, Ames Research Center. Revision 1, Moffett Field, California. 2014. 211p.

POULIN, J. S. **Measuring Software Reuse: Principles, Practices, and Economic Models**. 1st Edition, Addison-Wesley Professional,1996. 224 p. ISBN-10(0201634139).

PROKOP, L. NASA's Core Flight Software - a Reusable Real-Time Framework. Advanced Exploration Systems Core Flight Software Project Manager, NASA – Johnson Space Center (JSC), 2014. 61 transparências. Disponível em: <https://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/20140017040.pdf>. Acesso em 11 nov 2017.

PUMPKIN P/N710-00527. **Pluggable Processor Module (PPM) with Microchip® PIC24 for CubeSat Kit Motherboard**. Hardware Revision A, Datasheet. 2010. 14 p.

PUMPKIN P/N710-00484. **Single Board Computer Motherboard for Harsh Environments**. Hardware Revision D, Datasheet. 2009. 19 p.

RICHARDS, M. Software architecture patterns: Microkernel Architecture, **O'Reilly Media,** 2015**.** Disponível em: <https://www.oreilly.com/ideas/software-architecture-patterns/page/4/microkernel-architecture>. Acesso em: 1 abril 2017.

RTCA. **DO-178C:** Software Considerations in Airborne Systems and Equipment Certification. 2011. 144 p.

SCHNURR, R. CCSDS Standard On-Board Interfaces (SOIS). NASA Goddard Space Flight Center, MAPLD International Conference, Washington, D.C. 2006. 9 transparências. Disponível em: https://www.slideserve.com/garron/ccsds-standard-on-board-interfaces-sois>. Acesso em 18 out. 2016.

SERAFIM, G. Projeto CBD x SOA Serviços, processos de negócio e componentes que realizam os serviços, **Engenharia de Software 2**, Edição especial. Disponível em: <http://www.devmedia.com.br/artigo-engenharia-de-software-2-cbd-x-soa/9149>. Acesso em: 10 nov. 2016.

SOFTWARE ENGINEERING INSTITUTE (SEI). **Product Line Engineering**: Key Benefits: Why Product Line Engineering?, BigLever Software, Inc. 2013. Disponível em: <http://www.productlineengineering.com/benefits/key-benefits.html>. Acesso em 20 nov. 2016.

SOFTWARE ENGINEERING INSTITUTE (SEI1). **Product Line Engineering**: Three Dimensions of Product Line Engineering. BigLever Software, Inc. 2013. Disponível em:

<http://www.productlineengineering.com/overview/three-dimensions.html>. Acesso em 20 nov. 2016.

SOFTWARE ENGINEERING INSTITUTE (SEI2). **Product Line Engineering**: What is Product Line Engineering?, BigLever Software, Inc. 2013. Disponível em: <http://www.productlineengineering.com/overview/what-is-ple.html>. Acesso em 20 nov. 2016.

SPECHT, E.; *et al*. Analysis of the Use of Declarative Languages for Enhanced Embedded System Software Development. In: 20TH ANNUAL SYMPOSIUM ON INTEGRATED CIRCUITS AND SYSTEMS DESIGN, 2007, Rio de Janeiro, **Proceedings…** Copacabana: SBCCI, 2007. p. 324-329. Disponível em: < https://www.researchgate.net/publication/220850665_Analysis_of_the_use_of_declarative_languages_for_enhanced_embedded_system_software_development >. Acesso em 10 dez. 2016.

STEVENS, D.; *et al*. FACE Master Class. The Open Group, IOA, London, England. 2016. 88 transparências. Disponível em:< http://www.opengroup.org/sites/default/files/contentimages/face_master_class_presentation_final_v3.pdf>. Acesso em 11 out. 2016.

STMP3510. **Sigmatel D-Majortm Mp3 Flash Player**. Application Brief. 2 p. Disponível em: <http://pdf.datasheet.live/ac47c00f/sigmatel.com/STMP3510.pdf>. Acesso em 21 out. 2016.

TAGAWA, G. B. S.; E SOUZA, M. L. O. An Overview Of The Integrated Modular Avionics (IMA) Concept. In: 10ª CONFERÊNCIA BRASILEIRA DE DINÂMICA, CONTROLE E APLICAÇÕES, 2011, São Paulo, **Anais**... Águas de Lindoia: DINCON, 2011. p. 277-280. Disponível em: https://pdfs.semanticscholar.org/580d/97a5cf859beb8c6f30261e17d29e1f7947d7.pdf>. Acesso em 20 mai. 2017.

TANENBAUM, A.S. **Redes de computadores**. 5nd Edition, Pearson, 2011. 600 p. ISBN(9788576059240).

THE OPEN GROUP. **SOA Source Book**. Disponível em: <http://www.opengroup.org/soa/source-book/intro/>. Acesso em: 15 jan. 2017.

VARNELL-SARJEANT, J.; ANDREWS, A. A. Comparing reuse strategies in different development environments. In**: Advances in Computers Volume 97**, 1st.Edition, Elsevier, 2015. 276 p. ISBN (9780128023419).

WEISS, K. A.; *et al*. Reusable Software Architectures for Aerospace Systems. In: **Aircraft Engineering and Aerospace Technology**, v. 75, n. 5, p. 461-469, 2004. Disponível em: <https://pdfs.semanticscholar.org/c56f/a70eea8e230a4cca91b567f3958f0d58ff6f.pdf>. Acesso em 3 mar. 2017.

GLOSSÁRIO

ativos – São artefatos e/ou documentos necessários para o desenvolvimento de uma aplicação de software, como : requisitos, funções, modelos, códigos, testes, etc

engenharia recorrênte – Traduz-se em retrabalho da engenharia de desenvolvimento.

***feature*-PLE –** São ativos da PLE que descrevem uma determinada capacidade / funcionalidade.

ontologia - Possui um significado muito maior, entretanto nesta pesquisa, seu uso restringe-se a identificação de ativos semelhantes dentro do domínio do Reuso.

produto-alvo - É o produto onde o reuso é aplicado.

produtos-PLE - são componentes de software, onde foi empregado a técnica PLE e possuem características de reutilização.

projeto-alvo - É o projeto onde o reuso é aplicado.

Portfólio - Termo de acordo com guia PMBOK 5a. Ed. Se a relação entre projetos for somente a de um cliente, vendedor, tecnologia ou recurso compartilhado, o esforço deve ser gerenciado como um portfólio de projetos e não como um programa.

táticas - São meios fundados no uso de técnicas, processos e abordagens, que neste caso, favorecem o reuso de software.