UNIVERSIDADE DE SÃO PAULO
ESCOLA DE ENGENHARIA DE SÃO CARLOS

DANIEL DOS SANTOS JÚNIOR

Implementação de Processo de *Software* Para Teste de Equipamentos Aeroespaciais

São Carlos
2007

DANIEL DOS SANTOS JÚNIOR

Implementação de Processo de *Software* Para Teste de Equipamentos Aeroespaciais

Dissertação apresentada à Escola de Engenharia de São Carlos da Universidade de São Paulo, como parte dos requisitos para a obtenção do Título de Mestre em Engenharia Elétrica.

Orientador: Prof. Dr. Ivan Nunes da Silva

São Carlos
2007

Ficha catalográfica preparada pela Seção de Tratamento da Informação do Serviço de Biblioteca – EESC/USP

Santos Júnior, Daniel dos
S237i Implementação de processo de *software* para teste de equipamentos aeroespaciais / Daniel dos Santos Júnior ; orientador Ivan Nunes da Silva. -- São Carlos, 2007.

Dissertação (Mestrado) - Programa de Pós-Graduação em Engenharia Elétrica e Área de Concentração em Sistemas Dinâmicos -- Escola de Engenharia de São Carlos da Universidade de São Paulo.

1. Engenharia de software - desenvolvimento ágil. 2. Engenharia aeroespacial. I. Título.

FOLHA DE JULGAMENTO

Candidato: Engenheiro **DANIEL DOS SANTOS JÚNIOR**

Dissertação defendida e julgada em 04/05/2007 perante a Comissão Julgadora:

APROVADO

Prof. Associado **IVAN NUNES DA SILVA (Orientador)**
(Escola de Engenharia de São Carlos/USP)

APROVADO.

Prof. Dr. **EVANDRO LUIS LINHARI RODRIGUES**
(Escola de Engenharia de São Carlos/USP)

APROVADO

Prof. Dr. **JOSÉ DEMISIO SIMÕES DA SILVA**
(Instituto Nacional de Pesquisas Espaciais/INPE)

Prof. Associado **GERALDO ROBERTO MARTINS DA COSTA**
Coordenador do Programa de Pós-Graduação em
em Engenharia Elétrica e
Presidente da Comissão de Pós-Graduação da EESC

Dedicatória

À Fabíula, meu ponto de apoio durante a realização deste trabalho, que demonstrou paciência, compreensão e carinho em todas as horas difíceis, a quem eu amo profundamente!

Agradecimentos

Ao meu orientador, Prof. Dr. Ivan Nunes da Silva, pela confiança em mim depositada, pelo apoio durante a condução do trabalho e pelas observações valiosas que enriqueceram este documento.

À Universidade de São Paulo, onde aprendi não só engenharia, mas inúmeras lições de vida.

À Opto Eletrônica, em especial ao seu presidente, Prof. Dr. Jarbas Caiado de Castro, pela oportunidade de realizar o curso de mestrado, possibilitando ainda a apresentação do trabalho realizado nas dependências da empresa.

Ao Dr. Mario Antonio Stefani, diretor de P&D da Opto Eletrônica, pelo apoio constante e por todos os ensinamentos no decorrer dos últimos sete anos.

Aos amigos Ânderson Castellar, Henrique Pazelli e Lucas Supino, companheiros na realização deste projeto.

Ao grande amigo Rodrigo Modugno, de importância fundamental neste trabalho. É muito bom ter em quem sempre confiar.

Ao Moisés Satílio da Silva, que além de seu trabalho sério e amizade, contribuiu com parte das fotos aqui apresentadas.

A todos os demais pesquisadores (e alguns ex-pesquisadores) da Opto Eletrônica: Alan Garcia, Alessandro Damiani, Alex Affonso, Alexandre Soares, Alexandre Malavolta, Ânderson Santos, André Dias, Aparecido de Arruda, Artur Gonçalves, Carlos Burato, Carlos Romano, Cassius Ramos, Danielle Ferrari, Eduardo Richter, Érica Gabriela, Fabiano dos Santos, Fátima Yasuoka, Fernando Pegoraro, Giuliano Rossi, Guilherme Castro, João Roriz, José Antônio Ottoboni, Larissa Torres, Lucas Santos, Lucimara Scadutto, Luiz Carlos

Ottoboni, Marc'Antônio Bezerra, Márcio Tayama, Mariana Carvalho, Mariano Moreno, Marisley Hirinéia, Maurício Mota, Ricardo de Oliveira, Rogério Scadutto, Sanderson Barbalho, Saulo Bombonato, Sérgio Evangelista, Tiago Sanches e Vítor Santos. Todos participaram da execução deste projeto.

Finalmente, aos meus pais, Daniel e Maria Fátima, por me ensinarem, através do exemplo, a trabalhar com ética, honestidade e seriedade e por me darem tantas oportunidades que a vida não pôde lhes oferecer, possibilitando-me alcançar mais este objetivo. Todas as minhas vitórias se devem a eles.

"Porque dele, e por meio dele, e para ele são todas as coisas. Glória, pois, a ele eternamente. Amém."

Romanos 11,36

"We will either find a way or make one."

Hannibal

"Ser homem é ser responsável. É sentir que colabora na construção do mundo."

Antoine de Saint-Exupéry

"Any intelligent fool can make things bigger and more complex... It takes a touch of genius - and a lot of courage to move in the opposite direction."

Albert Einstein

Resumo

SANTOS JÚNIOR, D. **Implementação de Processo de *Software* Para Teste de Equipamentos Aeroespaciais**. 2007. 190 p. Dissertação (Mestrado em Engenharia Elétrica) – Departamento de Engenharia Elétrica, Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2007.

O MUX-GSE é um conjunto de equipamentos ópticos, mecânicos e eletrônicos que verifica, através de testes automáticos, uma série de requisitos funcionais e auxilia na montagem e integração da câmera multiespectral dos satélites CBERS-3&4. Estes satélites, fruto de uma parceria entre Brasil e China, são instrumentos de sensoriamento remoto, produzindo imagens da terra para estudos em diversas áreas, principalmente as relacionadas à exploração sustentável dos recursos naturais. A câmera multiespectral é a primeira do gênero projetada e produzida no Brasil e tem grande importância dentro do Programa Espacial Brasileiro. O sucesso da câmera multiespectral está intimamente ligado ao bom funcionamento do MUX-GSE, conseqüentemente, os *softwares* que controlam estes equipamentos e realizam os testes automáticos precisam de alta confiabilidade. O processo de *software* é um conjunto estruturado, composto de modelo de desenvolvimento, atividades, métodos, ferramentas e práticas, que pode assegurar a qualidade necessária a um produto de *software*. A definição deste processo é o primeiro passo para a execução de um projeto complexo. O objetivo deste trabalho é a implementação de um processo que permita a construção dos aplicativos do MUX-GSE em curto espaço de tempo, com uma equipe pequena e com o nível de confiabilidade necessário. A solução proposta, detalhada no texto, é baseada nos métodos ágeis, que definem práticas simples, mas que permitem assimilar mudanças ocorridas em qualquer fase do desenvolvimento. A implementação desta metodologia permitiu a produção dos aplicativos necessários a despeito de uma série de problemas enfrentados e sem atrasos

que impactassem o desenvolvimento da câmera multiespectral. Este trabalho mostra que um conjunto de técnicas relativamente simples pode ser mais adequado que aquelas tradicionalmente aplicadas na engenharia de *software*, mesmo em projetos complexos, cabendo à equipe de desenvolvimento a análise e seleção do melhor método. Mostra ainda que estas técnicas permitem manter ou até superar a confiabilidade obtida através de métodos tradicionais.

Palavras chave: desenvolvimento ágil. engenharia de software. engenharia aeroespacial

Abstract

SANTOS JÚNIOR, D. Software Process Implementation for Aerospatial Equipment Testing. 2007. 190 p. Thesis (Master degree) – Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2007.

The MUX-GSE is a set of optical, mechanical and electronic equipments that verifies, through automatic tests, a series of functional requirements and assists during the assembly and integration phases of the satellites CBERS-3&4's multispectral camera. These satellites, fruit of a partnership between Brazil and China, are remote sensing instruments, producing images of the Earth for studies in several areas, mainly the ones related to the sustainable exploitation of natural resources. The multispectral camera is the first of its gender fully projected and produced in Brazil and is of great importance for the Brazilian Space Program. The success of the multispectral camera is intimately related to the good operation of the MUX-GSE, therefore, the necessity of reliable software to control these equipments and to accomplish the automatic tests. The software process is a structured set composed of development model, activities, methods, tools and practices, that can assure the necessary quality to a software product. The definition of this process is the first step in the execution of a complex project. The objective of this work is to implement a process that allows the construction of the MUX-GSE's software in short term, by a small team and with the required reliability level. The proposed solution, detailed in the text, is based on agile methods, which define simple practices, but that allow to assimilate changes in any development stage. The implementation of this methodology provided means to produce the necessary applications, in spite of several faced problems, without delays that could possibly prejudice the multispectral camera development schedule. This work shows that relatively simples techniques can be more appropriate than those traditionally applied in software engineering, even in complex projects,

being the development team on charge of the analysis and decision of the most suitable development method. It also shows that the employed technique can reach, or even surpass, the software reliability achieved through traditional methods.

Keywords: agile development. software engineering. aerospatial engineering.

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas e Siglas

AEB	Agência Espacial Brasileira
ASD	*Adaptive Software Development* – Desenvolvimento Adaptativo de *Software*
B	*Blue* – Azul
CBERS	Satélite Sino-Brasileiro de Recursos Terrestres
CCD	*Charge-Coupled Device* – Dispositivo de Carga Acoplado
CCD	*High Resolution CCD Camera* – Câmera Imageadora de Alta Resolução
CGSE	*Complete Ground Support Equipment* – Equipamento Completo de Suporte em Terra
CLA	Centro de Lançamento de Alcântara
CLBI	Centro de Lançamento da Barreira do Inferno
CMOS	*Complementary Metal-Oxide Semiconductor* – Semicondutor metal-óxido complementar
CPLD	*Complex Programable Logic Device* – Dispositivo Lógico Programável Complexo
CRC	Centro de Rastreio e Controle de Satélites
DAQ	*Data Acquisition* – Aquisição de Dados
DDR	*Digital Data Recorder* – Gravador de Dados Digitais
DLL	*Dynamic Link Library* – Biblioteca de Vínculo Dinâmico
DSDM	*Dynamic Systems Development Method* – Método de Desenvolvimento de Sistemas Dinâmicos
G	*Green* – Verde
GOCNAE	Grupo de Organização da Comissão Nacional de Atividades Espaciais
GPIB	*General Purpose Interface Bus* – Duto de Interface de Uso Geral
HSDIO	*High-Speed Digital Input/Output* – Saída/Entrada Digital em Alta Velocidade
INPE	Instituto Nacional de Pesquisas Espaciais
IRMSS	*Infra-Red Multispectral Scanner* – Imageador por Varredura de Média Resolução
I/O	*Input/Output* – Entrada/Saída
LAN	*Local Area Network* – Rede de Área Local
LIT	Laboratório de Integração e Testes
LVDS	*Low Voltage Differential Signaling* – Sinalização Diferencial de Baixa Tensão

MECB	Missão Espacial Completa Brasileira
MIR	*Medium Infra-Red* – Infravermelho médio
MTF	*Modular Transfer Function* – Função de Transferência de Modulação
MUX	Câmera Multiespectral
MUXCAM	Câmera Multiespectral
MUX-GSE	*Mux Ground Support Equipment* – Equipamento de Suporte em Terra da MUX
NIR	*Near Infra-Red* – Infravermelho próximo
PAN	Pancromática
PANMUX	Câmera Pancromática
PCI	*Peripheral Component Interconect* – Interconector de Componentes Periféricos
PDR	*Preliminary Design Review* – Revisão do Projeto Preliminar
P&D	Pesquisa e Desenvolvimento
PNAE	Programa Nacional de Atividades Espaciais
PNDAE	Política Nacional de Desenvolvimento das Atividades Espaciais
QAS	Qualidade Assegurada de *Software*
R	*Red* – Vermelho
RADAR	Radar de Abertura Sintética
SCD	Satélite de Coleta de Dados
SEI	Sistema de Exibição de Imagens
SeRe	Sensoriamento Remoto
SGSE	*Simplified Ground Support Equipment* – Equipamento Simplificado de Suporte em Terra
SSR	Satélites de Sensoriamente Remoto
SPOT	*Systeme Pour l'Observation de la Terre*
SWIR	*Short-Wave Infra-Red* – Infravermelho de ondas curtas
TC	Telecomando
TM	Telemetria
TH	*Thermal* – Infravermelho termal
UCA	Usina de Propelentes Coronel Abner
V&V	Verificação e Validação
VI	*Virtual Instrument* – Instrumento Virtual

VLS	Veículos Lançadores de Satélites
WFI	*Wide Field Imager* – Imageador de Amplo Campo de Visada
XP	*eXtreme Programming* – Programação Extrema

Lista de Símbolos

km	quilômetros
kg	quilogramas
US$	dólares
m	metros
%	por cento
µm	micrometros
°	graus
W	Watts
’’	polegadas
U	unidade de *rack*
mm	milímetros
Gbit/s	gigabits por segundo
Mbit/s	megabits por segundo
Mbit	megabits
s	segundos
fps	f*rames per second* – quadros por segundo

Sumário

1 Introdução

1.1 Motivação e Relevância do Trabalho

1.1.1 O Programa Espacial Brasileiro

O programa espacial brasileiro teve início na década de sessenta quando foi fundado o Grupo de Organização da Comissão Nacional de Atividades Espaciais (GOCNAE). Eram objetivos iniciais: construção de um campo de lançamento de foguetes, preparação de equipes especializadas em lançamento de foguetes e estabelecimento de programas de sondagens meteorológicas e ionosféricas em cooperação com instituições estrangeiras.

Os primeiros resultados foram obtidos logo em 1965, com as primeiras campanhas de lançamento de foguetes de sondagem com carga útil, a partir do Centro de Lançamento da Barreira do Inferno (CLBI), localizado em Natal, RN.

Em 1971 o GOCNAE tornou-se Instituto de Pesquisas Espaciais, hoje chamado Instituto Nacional de Pesquisas Espaciais (INPE). Em 1979 foi aprovada a Missão Espacial Completa Brasileira (MECB), cuja meta era alcançar "[...] completa autonomia no projeto e construção de satélites de pequeno porte, e lançadores associados." (SOCIEDADE BRASILEIRA PARA O PROGRESSO DA CIÊNCIA, 2001).

Em 10 de fevereiro de 1994 foi criada a Agência Espacial Brasileira (AEB), através da lei n° 8.854 (BRASIL, 1994a). Em 08 de dezembro de 1994, pelo decreto n° 1332 (BRASIL, 1994b), foi atualizada a Política Nacional de Desenvolvimento das Atividades Espaciais (PNDAE), que apresenta as diretrizes a serem observadas na tentativa de se alcançar seu objetivo geral: "[...] promover a capacidade do País para, segundo conveniência e critérios

próprios, utilizar os recursos e as técnicas espaciais na solução de problemas nacionais e em benefício da sociedade brasileira".

O PNDAE orienta a criação do Programa Nacional de Atividades Espaciais (PNAE). Em 1996 foi aprovada a primeira edição do PNAE, cujos objetivos atuais, segundo a própria Agência Espacial Brasileira (2005, p. 13), são "[...] capacitar o país para desenvolver e utilizar tecnologias espaciais na solução de problemas nacionais e em benefício da sociedade brasileira [...]".

Ainda segundo a Agência Espacial Brasileira (2005, p. 7): "Aqui não aportarão tecnologias estratégicas por deferência de terceiros". E mais: "Serão estes os países [os que dominarem a tecnologia espacial] em condição de sustentar posições e argumentar nas mesas de negociação diplomática".

O mesmo documento ressalta que o PNAE consolidou no Brasil uma comunidade científica reconhecida internacionalmente, composta por profissionais de engenharia e tecnologia espaciais e por pesquisadores especializados em ciências espaciais, sensoriamento remoto e meteorologia por satélite. A infra-estrutura implantada, o pessoal, as ferramentas e metodologias desenvolvidas, foram incorporadas ao cotidiano de atividades em diversas áreas.

No desenvolvimento de satélites, dois programas destacam-se pelo sucesso alcançado: o Satélite de Coleta de Dados (SCD) e o Satélite Sino-Brasileiro de Recursos Terrestres (CBERS), que será melhor detalhado na próxima seção. São destaques ainda: o desenvolvimento de Foguetes de Sondagem e de Veículos Lançadores de Satélites (VLS-1), apesar destes últimos não terem obtido êxito nas três campanhas de lançamento realizadas até o momento.

Quanto à infra-estrutura, são referências os Centros de Lançamento de Alcântara (CLA) e da Barreira do Inferno (CLBI), o Laboratório de Integração e Testes (LIT – Figura

1.1), o Centro de Rastreio e Controle de Satélites (CRC) e a Usina de Propelentes Coronel Abner (UCA), além de numerosos laboratórios de pesquisa.

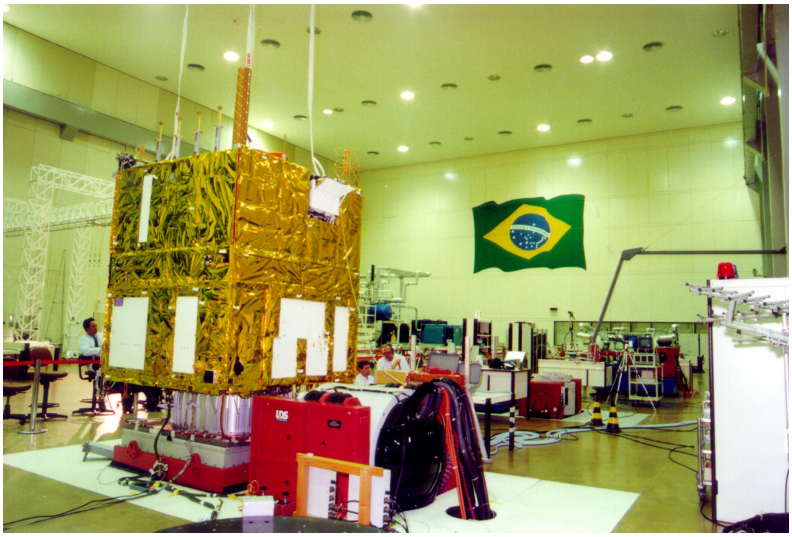


Figura 1.1: Vista do LIT-INPE: CBERS-2 durante teste de vibração
Fonte: Instituto Nacional de Pesquisas Espaciais (2006d).

1.1.2 O Programa CBERS

Entre as atividades espaciais, a observação da terra é uma das que apresenta grande número de aplicações já desenvolvidas. Os satélites de observação permitem a obtenção de dados sistematicamente e possibilitam o acompanhamento de grandes áreas territoriais em qualquer ponto do planeta, mesmo os de mais difícil acesso.

Podemos citar agricultura, pecuária, cartografia, hidrologia, demografia, entre outras atividades, como usuárias efetivas das imagens obtidas pelos satélites brasileiros. Estas imagens permitem realizar estudos ambientais, planejamento urbano, projetos agrícolas, estratégias de defesa territorial, reações a desastres naturais, avaliação de mananciais, gerenciamento de crises, etc.

Atualmente existem quatro missões de monitoramento e observação em andamento: Programa Sino-Brasileiro, Programa de Coleta de Dados, Programa Satélites de

Sensoriamento Remoto (SSR) e Programa Radar de Abertura Sintética (RADAR), sendo que os dois primeiros já possuem satélites em funcionamento.

O Programa Sino-Brasileiro teve início em 06 de julho de 1988, quando foi assinado um acordo de parceria entre os governos de Brasil e China com o objetivo de desenvolver dois satélites avançados de sensoriamento remoto de médio porte (1450 kg). Ele recebeu o nome de Satélite Sino-Brasileiro de Recursos Terrestres (*"China-Brazil Earth Resources Satellite"* – CBERS).

Como resultado deste acordo, dois satélites foram desenvolvidos, CBERS-1 e 2, que foram lançados respectivamente em 14 de outubro de 1999 e 21 de outubro de 2003. Seus projetos apresentaram características semelhantes aos satélites LandSat e SPOT (*"Systeme Pour l'Observation de la Terre"*), respectivamente americano e francês, referências científicas da época. Eles foram equipados com três câmeras, permitindo obtenção de imagens em nove faixas espectrais com máxima resolução espacial[1] de 20 m, um repetidor para o Sistema Brasileiro de Coleta de Dados Ambientais e um Monitor do Ambiente Espacial.

Pelo acordo, o Brasil assumiu 30% dos custos, contando os lançamentos. "Ficou responsável por vários subsistemas, como estrutura mecânica, suprimento de energia elétrica, câmera grande-angular e telecomunicações de serviço na banda S" (BRASIL, 1998, p. 20). Ao final do projeto, os custos totais dos CBERS-1 e 2 aos cofres brasileiros somaram US$ 118 milhões.

As câmeras dos CBERS-1 e 2 são: Imageador de Amplo Campo de Visada (*"Wide Field Imager"* - WFI), Câmera Imageadora de Alta Resolução (*"High Resolution CCD Camera"* – CCD) e Imageador por Varredura de Média Resolução (*"Infra-Red Multispectral Scanner"* – IRMSS). Algumas de suas características mais relevantes são apresentadas na Tabela 1.1.

[1] Resolução espacial é o tamanho que cada ponto da imagem (pixel) representa no mundo real.

Tabela 1.1 – Características das câmeras dos CBERS-1 e 2

Câmera	Bandas espectrais		Campo de visada	Resolução espacial	Largura da faixa imageada	Apontamento do espelho	Resolução temporal[2]
	Comprimento de onda	Região					
WFI	0,63-0,69 μm 0,77-0,89 μm	R NIR	60°	260 x 260 m	890 km	Não permite	5 dias
CCD	0,51-0,73 μm 0,45-0,52 μm 0,52-0,59 μm 0,63-0,69 μm 0,77-0,89 μm	PAN B G R NIR	8,3°	20 x 20 m	113 km	±32°	26 dias (3 dias – visada lateral)
IRMSS	0,50-1,10 μm 1,55-1,75 μm 2,08-2,35 μm 10,40-12,50 μm	PAN MIR SWIR TH	8,8°	80 x 80 m (160 x 160 m – TH)	120 km	Não permite	26 dias

PAN – Pancromática; B – Azul; G – Verde; R – Vermelho; NIR – Infravermelho próximo; MIR – Infravermelho médio; SWIR – Infravermelho de ondas curtas; TH – Infravermelho termal[3].
Fonte: Epiphanio (2005).

Segundo o sítio "CBERS – Satélites – As câmeras dos Satélites CBERS-1 e 2" (INSTITUTO NACIONAL DE PESQUISAS ESPACIAIS, 2006e), as câmeras dos CBERS-1 e 2, combinadas, permitem obter uma série de informações, "[...] resolver a grande variedade de escalas temporais e espaciais típicas de nosso ecossistema". A câmera CCD pode orientar seu campo de visada em ±32°, possibilitando a obtenção de imagens estereoscópicas, além de permitir focalizar qualquer fenômeno detectado pelo WFI em no máximo a cada três dias, para estudos mais detalhados.

O WFI possui bandas espectrais também cobertas pela CCD, permitindo a combinação dos dados obtidos pelos dois sensores. As imagens do IRMSS, com banda estendendo-se até o infravermelho termal, podem ser correlacionadas com as da câmera CCD, devido às características temporais similares.

Segundo Sausen (2006), o WFI permite a realização de estudos regionais (Figura 1.2), tais como o mapeamento de uma região inteira ou da área de um estado, enquanto o sensor

[2] Resolução temporal, período de revisita ou período de observação é o intervalo entre duas observações consecutivas de um mesmo ponto da superfície por uma câmera de satélite. Este intervalo é inversamente proporcional à largura de faixa imageada e está relacionado com as órbitas quase polares (de um pólo a outro a uma distância aproximada de 800 km da superfície da terra) descritas pelos satélites de sensoriamento.
[3] A banda infravermelha pode ser dividida em seções menores, mas esta divisão não é precisa, sendo empregada de maneira variada por diferentes autores. Neste documento consideramos as faixas empregadas pelas fontes consultadas, que se referem aos sensores dos satélites CBERS. Outros textos podem trazer classificações distintas.

CCD pode ser utilizado para estudos regionais/locais, como planejamento urbano-regional, estudos de áreas agrícolas em local ou média-escala. Assim, as diferentes resoluções espaciais contribuem para que as imagens encontrem utilização prática em uma maior variedade de aplicações. Uma prova disso é que somente no XII Simpósio Brasileiro de Sensoriamento Remoto (2005), mais de setenta dos trabalhos apresentados utilizavam imagens dos CBERS.

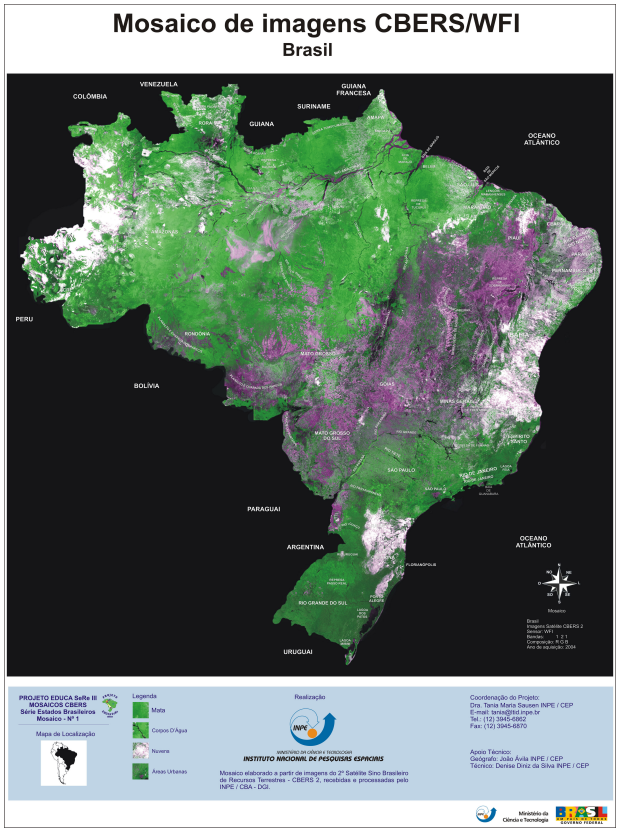
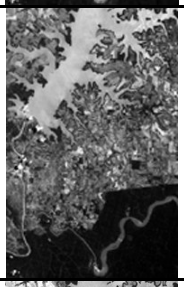
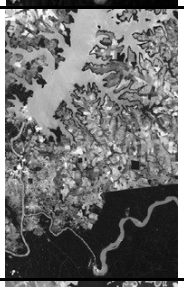
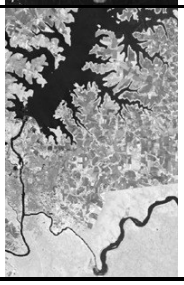


Figura 1.2: Brasil – Mosaico de Imagens do WFI
Fonte: Programa Educa SeRe – Elaboração de Material Didático para o Ensino de Sensoriamento Remoto Utilizando Imagens CBERS (2006)

O número de bandas espectrais dos sensores também contribui para a grande diversidade de aplicações. Isto pode ser verificado através da Tabela 1.2, que apresenta as bandas da câmera CCD dos CBERS-1 e 2, a imagem de uma mesma região obtida com cada uma delas e algumas informações que se consegue extrair destas imagens.

Os CBERS-1 e 2 foram projetados com vida útil de dois anos, porém O CBERS-1 operou com sucesso até o dia 13 de agosto de 2003, quase dois anos a mais que o previsto, quando todas as transmissões de dados de imagens deixaram de funcionar. O CBERS-2, por sua vez, trabalha até hoje, apesar de algumas limitações.

Tabela 1.2 – Bandas da câmera CCD e suas aplicações

Banda espectral	Comprimento de onda	Região	Imagem	Possíveis aplicações
B1	0,45-0,52 μm	Azul		• Mapeamento de águas costeiras • Diferenciação entre solo e vegetação • Diferenciação entre vegetação conífera e decídua
B2	0,52-0,59 μm	Verde		• Mapeamento de vegetação • Qualidade da água
B3	0,63-0,69 μm	Vermelho		• Absorção de clorofila • Diferenciação de espécies vegetais • Áreas urbanas, uso do solo • Agricultura • Qualidade da água
B4	0,77-0,89 μm	Infravermelho próximo		• Delineamento de corpos de água • Mapeamento morfológico • Mapeamento geológico • Áreas de queimadas • Áreas úmidas • Agricultura • Vegetação

Fonte: Sausen (2006)

Ambos os satélites apresentaram alguns problemas técnicos, mas os resultados obtidos podem ser considerados muito relevantes. Entre os problemas, pode-se citar o mau funcionamento do WFI do CBERS-1, em maio de 2000, apenas sete meses após o lançamento, e o defeito em uma das baterias do CBERS-2, em abril de 2005, seis meses antes do final de sua vida útil projetada, que forçou as equipes em terra a desligarem permanentemente o WFI e o IRMSS, mantendo em operação somente a câmera CCD.

Em julho de 2004, devido à grande procura por imagens do CBERS-2, foi acordada a fabricação do satélite CBERS-2B, com investimento brasileiro estimado em US$ 15 milhões. Este satélite, réplica do CBERS-2, porém com a substituição do IRMSS por uma câmera pancromática de alta resolução espacial (2,5 m), está passando pela fase final de integração e testes no LIT, do INPE, e seu lançamento, inicialmente previsto para 2006 (BRASIL, 2004), foi transferido para maio de 2007. O lançamento deste satélite pretende cobrir o intervalo entre o encerramento da utilização do CBERS-2, que já ultrapassou seu tempo de vida projetado, e o lançamento do CBERS-3, evitando interrupção no fornecimento de imagens.

As imagens obtidas pelos satélites CBERS podem ser acessadas gratuitamente por usuários nacionais e são distribuídas internacionalmente com cobrança de uma taxa de operação, estratégia que fez do Brasil o maior distribuidor de imagens de satélite. Segundo o sítio “CBERS – Catálogo CBERS disponibiliza imagens da América do Sul” (INSTITUTO NACIONAL DE PESQUISAS ESPACIAIS, 2006b), até março de 2006, haviam sido distribuídas 200 mil imagens a usuários do território nacional.

“Com a falha dos satélites americanos LANDSAT-5 e LANDSAT-7 [...] diversos países têm demonstrado interesse no programa sino-brasileiro” (INSTITUTO NACIONAL DE PESQUISAS ESPACIAIS, 2006c). O interesse internacional tem sido tão grande que os Estados Unidos já realizaram com sucesso testes de recepção das imagens do CBERS-2,

através do U. S. Geological Survey (USGS), órgão do Departamento do Interior dos Estados Unidos responsável também pelo programa LandSat.

O sucesso e o valor do programa CBERS fizeram com que os governos brasileiro e chinês decidissem levá-lo adiante com a assinatura de um Protocolo de Cooperação, em novembro de 2002, para o desenvolvimento e lançamento de mais dois satélites, os CBERS-3 e 4. Desta vez a participação brasileira foi ampliada, chegando a 50% dos custos, o que significará algo em torno de US$ 200 milhões, levando em conta também o lançamento e a infra-estrutura de solo. As características dos novos satélites foram aprimoradas em relação aos anteriores, incorporando inovações tecnológicas e visando ampliar as possibilidades de aplicação das imagens.

A previsão inicial era de que o lançamento destes satélites ocorresse em 2008 e 2011, porém estas datas podem ser alteradas tanto por pequenos atrasos no desenvolvimento de seus próprios projetos quanto pelo atraso do lançamento do CBERS-2B. Como este último será colocado em órbita somente em 2007 (a previsão inicial era outubro de 2006) e possui vida útil projetada de dois anos, é muito provável que o CBERS-3 não seja colocado em operação antes de 2009.

Os CBERS-3 e 4 serão equipados com quatro câmeras (Tabela 1.3), uma a mais que os antigos satélites. Elas possuirão mais bandas espectrais, porém manterão algumas das bandas já utilizadas, para que as imagens sejam compatíveis com as aplicações atuais. A resolução espacial máxima será de 5 x 5 m, somente na banda pancromática da câmera PANMUX. Os novos WFI e IRMSS terão melhor resolução que seus antecessores.

Tabela 1.3 – Características preliminares das câmeras dos CBERS-3 e 4

Câmera	Bandas espectrais		Resolução espacial	Largura da faixa imageada	Apontamento do espelho	Resolução temporal
	Comprimento de onda	Região				
WFI	0,52-0,59 µm	G	73 x 73 m	866 km	Não permite	5 dias
	0,63-0,69 µm	R				
	0,77-0,89 µm	NIR				
	1,55-1,75 µm	MIR				
MUXCAM	0,45-0,52 µm	B	20 x 20 m	120 km	Não permite	26 dias
	0,52-0,59 µm	G				
	0,63-0,69 µm	R				
	0,77-0,89 µm	NIR				
IRMSS	0,76-0,90 µm	NIR	40 x 40 m	120 km	Não permite	26 dias
	0,76-1,10 µm	PAN	(80 x 80 m TH)			
	1,55-1,75 µm	MIR				
	2,08-2,35 µm	SWIR				
	10,40-12,50 µm	TH				
PANMUX	0,51-0,86 µm	PAN	10 x 10 m	60 km	±32°	5 dias
	0,52-0,59 µm	G	(5 x 5 m PAN)			
	0,63-0,69 µm	R				
	0,77-0,89 µm	NIR				

PAN – Pancromática; B – Azul; G – Verde; R – Vermelho; NIR – Infravermelho próximo; MIR – Infravermelho médio; SWIR – Infravermelho de ondas curtas; TH – Infravermelho termal.
Fonte: Instituto Nacional de Pesquisas Espaciais (2006f).

A Câmera Multiespectral (MUXCAM ou MUX) será similar à câmera CCD dos satélites anteriores em resolução e nas quatro bandas espectrais B, G, R e NIR. A grande novidade em relação à MUX está no fato de ela ser totalmente desenvolvida e produzida no Brasil. É a primeira vez que a indústria nacional desenvolve uma câmera deste gênero (Seção 1.1.3).

Assim como nos CBERS-1 e 2, a carga útil dos satélites contará também com o repetidor para o Sistema Brasileiro de Coleta de Dados e o Monitor de Ambiente Espacial. As novidades apresentadas são relativamente pequenas. O objetivo principal é a continuidade do programa, mantendo as principais características já qualificadas da família CBERS e aproveitando o conhecimento adquirido durante a produção dos modelos anteriores.

Apesar disso, a tecnologia empregada foi atualizada. Um exemplo é a utilização de um Gravador de Dados Digitais (*"Digital Data Recorder"* – DDR) com tecnologia de estado sólido para armazenar as imagens das câmeras e permitir sua posterior transmissão às estações

em terra (nos CBERS-1 e 2 foi utilizado um gravador experimental de fita de alta densidade). Outro fato relevante é o aumento na vida útil projetada, que passou para três anos.

O governo brasileiro tenta evitar a repetição de erros estratégicos ocorridos na primeira etapa do programa. Um dos esforços é a proteção das poucas indústrias (a maioria pequenas e médias) que participam das atividades espaciais. Segundo a Agência Espacial Brasileira (2005) é admissível que muitas delas mantenham dependência estreita do Governo, em muitos casos seu principal cliente. Assim, as dificuldades orçamentárias governamentais prejudicam fortemente seu desempenho financeiro e atualização tecnológica.

Esta preocupação é válida, pois o fechamento de uma empresa, em geral, provoca dispersão das equipes de desenvolvimento e de informações armazenadas e em conseqüência desfaz-se de maneira irrecuperável o conjunto que detém o conhecimento adquirido em anos de investimento. Mesmo que a mão-de-obra qualificada mantenha-se na área de atuação, o que nem sempre ocorre, diversas informações, métodos produtivos, procedimentos de trabalho, resultados, etc. perdem-se. O tempo e o investimento necessário para que outra empresa atinja o mesmo nível tecnológico são desperdícios que devem ser evitados.

Além de proteger a base industrial já consolidada, o governo pretende estimular outras empresas a participarem da área tecnológica espacial. Isto é importante não só por gerar empregos e reduzir o fluxo de capital para o exterior, mas principalmente por reduzir a dependência de produtos e serviços externos nesta área. Este foi um dos motivos para aumentar a participação brasileira nos custos de desenvolvimento dos CBERS-3 e 4, ou seja, permitir contratar junto à indústria brasileira a fabricação de mais subsistemas dos satélites, sustentando e fomentando a expansão de empresas capazes de atuar neste tipo de projeto.

1.1.3 Câmera Multiespectral

Nos CBERS-1 e 2, apesar do Brasil ter ficado responsável pelo desenvolvimento da estrutura do satélite, houve dificuldade para encontrar um fornecedor brasileiro, pois a Embraer, empresa teoricamente mais capacitada para isso, não demonstrou interesse no projeto. Assim, o subsistema foi quase que totalmente contratado junto à China. Em contrapartida, os Computadores de Bordo e o Transmissor de Dados, que ficaram a cargo da China, foram desenvolvidos por empresas brasileiras (MILESKI, 2003). Assim, se observarmos os dados da Tabela 1.4, veremos que somente estrutura e MUX são projetos inéditos para a indústria nacional.

Tabela 1.4 – Divisão do trabalho de desenvolvimento

	Subsistema	CBERS-1 & 2		CBERS-3 & 4
		Responsável	Desenvolvedor	Responsável
Módulo de Serviço	Estrutura	Brasil	China	Brasil
	Controle térmico	China	China	China
	Controle de atitude e órbita	China	China	China
	Fonte de alimentação	Brasil	Brasil	Brasil
	Computador de bordo	China	Brasil	Brasil
	Telemetrias	Brasil	Brasil	Brasil
Carga útil	Câmera PANMUX	Não se aplica		China
	Câmera CCD / MUXCAM	China	China	Brasil
	IRMSS	China	China	China
	WFI	Brasil	Brasil	Brasil
	Transmissor de dados	China	Brasil	Brasil
	Coletor de dados	Brasil	Brasil	Brasil

Fontes: Instituto Nacional de Pesquisas Espaciais (2006g), Mileski (2003) e Silveira (2006).

A MUX foi especificada com quatro bandas (azul, verde, vermelho e infravermelho próximo), faixa imageada de 120 km e resolução espacial de 20 m. Seu desenvolvimento e produção ficaram a cargo da Opto Eletrônica, empresa vencedora da licitação correspondente. Suas características representam desafios tecnológicos a serem vencidos em mecânica, óptica, eletrônica, computação, térmica, entre outras áreas do conhecimento.

Alguns obstáculos enfrentados são:

- A MUX será submetida a grandes variações de temperatura e grandes diferenciais térmicos[4]. Isto exige controle ativo de temperatura na Câmera e compensação de deformações (dilatações e contrações mecânicas) para evitar que a imagem seja formada fora do ponto focal e para que o sistema óptico não seja danificado.
- A alimentação é feita através das baterias do satélite, que por sua vez são carregadas pela energia do sol captada através de um painel solar. Assim, a quantidade de energia disponível é limitada, exigindo que todos os subsistemas sejam projetados para serem muito econômicos. Este é um desafio especialmente difícil para a MUX, devido à necessidade de controle térmico, sistema que consome grande potência.
- O espaço disponível no satélite é limitado. A resolução espacial, 20 m, exige um sistema óptico grande (Figura 1.3), que ocupa quase todo o espaço reservado à MUX. Assim, tanto o *design* eletrônico quanto o mecânico devem ser muito compactos.

[4] O termo "variações de temperatura" refere-se à mudança de temperatura no tempo; diferencial térmico refere-se a diferentes temperaturas em pontos distintos da MUX, no mesmo instante de tempo.

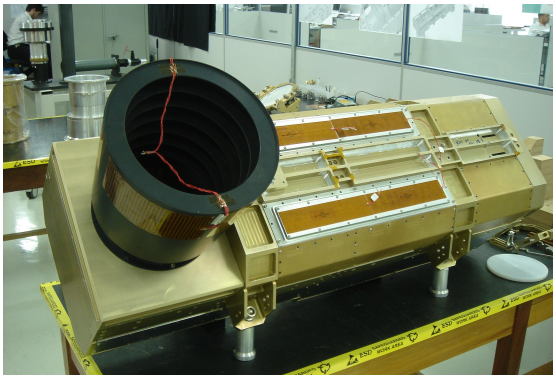
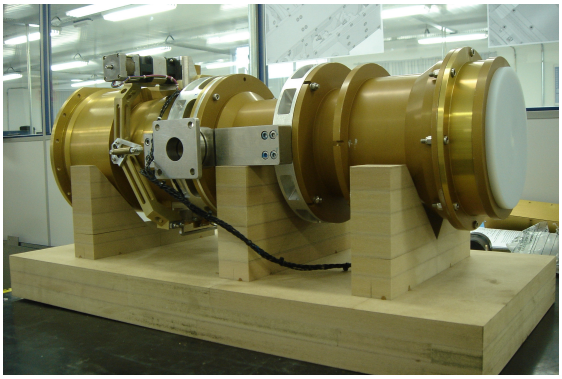



Figura 1.3: Câmera multiespectral (esq.) e seu canhão óptico (dir.) – modelo estrutural e térmico
Fonte: Opto Eletrônica

- Como o espaço, o peso também é limitado. Muito peso prejudicaria a colocação do satélite em órbita, além de deslocar seu centro de massa. Com isso, os componentes da MUX, em especial as peças mecânicas, devem ser produzidos com materiais leves e resistentes e com as menores dimensões possíveis, sem comprometer a resistência estrutural.
- O satélite é submetido a forças extremas, principalmente durante o lançamento, o que exige estrutura mecânica muito resistente. O mesmo vale para os componentes ópticos e seus suportes. Os circuitos eletrônicos e cabos devem ser selecionados cuidadosamente (componentes qualificados) e montados seguindo normas aeroespaciais.
- Em órbita a radiação recebida pelo satélite é muito grande. Seu efeito mais danoso é o envelhecimento precoce de alguns materiais usados na fabricação de componentes ópticos, eletrônicos e estruturais. Com isso, é preciso avaliar e, em alguns casos ensaiar, os problemas causados pela radiação em todos os componentes susceptíveis.
- O satélite é composto por grande número de circuitos eletrônicos que operam a freqüências variadas e podem interferir uns nos outros, se o *design* não for bem

> planejado. Assim, a emissão e a susceptibilidade eletromagnética são fatores de grande relevância, que devem ser considerados desde a idealização do projeto eletrônico e a seleção de componentes. O posicionamento dos circuitos, o comprimento e as características dos cabos, a necessidade de blindagens e diversos outros fatores devem ser observados para que se alcancem os índices especificados.

Por todas estas condições adversas, os requisitos da MUX são muito rígidos. Além disso, os custos envolvidos no projeto, a impossibilidade de manutenção após o lançamento e sua relevância dentro do PNAE e valor estratégico para a cooperação internacional Brasil-China, tornam fundamental identificar e eliminar possíveis defeitos, assegurando a qualidade do produto mesmo que isso implique em elevado investimento financeiro.

Para garantir o cumprimento de todas as especificações é gerada uma matriz de verificação de requisitos e um plano de qualidade e garantia que, aprovados pelo INPE, devem ser seguidos à risca. Este procedimento garante que todos os requisitos sejam verificados antes que a MUX seja colocada em órbita. Sua execução exige uma série de análises e testes de aceitação em várias etapas do desenvolvimento, culminando com a realização de uma bateria final de testes, após a integração ao satélite.

A realização dos testes dos parâmetros eletrônicos, mecânicos, ópticos e funcionais, a montagem e integração dos componentes da MUX exigem uma série de equipamentos de medição, suportes mecânicos, sistemas de alinhamento, *softwares* de avaliação, etc. Reunidos eles constituem o Equipamento de Suporte em Terra da Câmera Multiespectral (*"MUX Ground Support Equipment"*, ou MUX-GSE), que também foi especificado como parte integrante do desenvolvimento da MUX.

São funções do MUX-GSE:

- Gerar padrões ópticos para a calibração e testes.
- Realizar medidas quantitativas das características ópticas, mecânicas e eletrônicas especificadas, entre elas: função de transferência de modulação (*"Modular Transfer Function"* – MTF), plano focal, parâmetros radiométricos e distorção da imagem.
- Receber, decodificar, exibir em tempo real e armazenar as imagens obtidas pela MUX, verificando a formatação e taxa de dados.
- Auxiliar os ajustes ópticos durante a integração da MUX, fornecendo informações sobre o foco, colimação e centragem do sistema óptico e sobre o posicionamento do sensor CCD.
- Realizar e armazenar informações de calibração.
- Avaliar parâmetros mecânicos como: dimensões, massa, centro de massa e momento de inércia.
- Avaliar parâmetros elétricos como potência consumida, limites operacionais e características dos circuitos.
- Comandar e conferir o funcionamento da MUX em todos os seus modos de operação.
- Verificar o funcionamento de todas as telemetrias e telecomandos (TM&TC) e o controle de todos os sistemas que permitem ajustes.
- Suportar mecanicamente a MUX e suas partes durante a integração e a realização dos testes.

Para permitir a realização de todas as análises planejadas, foram solicitados dois tipos de MUX-GSE: Completo (*"Complete GSE-MUX"* ou CGSE), contendo complexo banco óptico para alinhamento e integração da MUX, e Simplificado (*"Simplified GSE-MUX"* ou

SGSE), que contém um sistema portátil para análise da imagem da MUX já integrada ao satélite.

A operação do MUX-GSE deve ser a mais automatizada possível, reduzindo a probabilidade de que o operador provoque erros nas medidas. As ações não automáticas devem ser indicadas numa seqüência de passos. Além disso, sempre que necessário, todos os equipamentos de medida devem ser operados manualmente, permitindo a execução de testes e medidas não originalmente previstos.

Por fim, o MUX-GSE deve ser projetado e construído levando em consideração a segurança da MUX: devem existir rotinas e procedimentos de teste que garantam seu funcionamento antes do acoplamento dos cabos para a realização de medidas. Devem existir circuitos de proteção, polarizadores nos conectores, indicações visuais, instruções claras e precisas e todas as outras medidas cabíveis para mitigar acidentes.

Em um sistema de testes tão complexo e com nível de automatismo tão elevado, os *softwares* têm importância destacada. Eles são responsáveis por comandar todos os equipamentos de medição, exibir instruções, avaliar todos os dados obtidos, realizar procedimentos de segurança, registrar comandos, erros e resultados, entre outras tarefas. Tudo isso com confiabilidade e através de uma interface simples e intuitiva.

O projeto e a criação destes *softwares* em prazo de tempo tão curto quanto o disponibilizado para este projeto e com o grau de qualidade necessário para o sistema, é uma tarefa bastante delicada. Exige o trabalho coordenado de uma equipe de desenvolvimento de *software* em perfeita harmonia com uma equipe multidisciplinar, responsável pelos outros sistemas do MUX-GSE. Exige compromisso de todos os envolvidos e a aplicação de uma metodologia confiável para que se tenha segurança sobre o funcionamento do código gerado.

É importante ressaltar que erros de *software* neste projeto podem provocar desde atrasos no lançamento do satélite até danos na MUX. Pior ainda, podem impedir a detecção de falhas,

aprovando erroneamente alguma característica, o que poderia levar ao espaço uma câmera que não atende aos requisitos exigidos.

1.2 Objetivos e Justificativas do Trabalho

Este trabalho contempla o desenvolvimento dos *softwares* do MUX-GSE, que ocorreu ao mesmo tempo em que os outros componentes do sistema foram projetados e produzidos. Os desafios apresentados no tópico anterior justificam a utilização de todo o conhecimento disponível para garantir o bom funcionamento e a conclusão do projeto no prazo estipulado.

O processo de *software* (Figura 1.4) pode ser definido como um conjunto estruturado de métodos, atividades, práticas e ferramentas que, quando corretamente aplicados, permitem o desenvolvimento de um produto de *software* com qualidade e baixo custo. O objetivo principal deste trabalho é implementar um processo de *software* que permita o desenvolvimento de *software* do MUX-GSE com qualidade, com uma equipe pequena e de modo que o desenvolvimento da MUX não seja prejudicado, mas sim auxiliado pelo MUX-GSE.

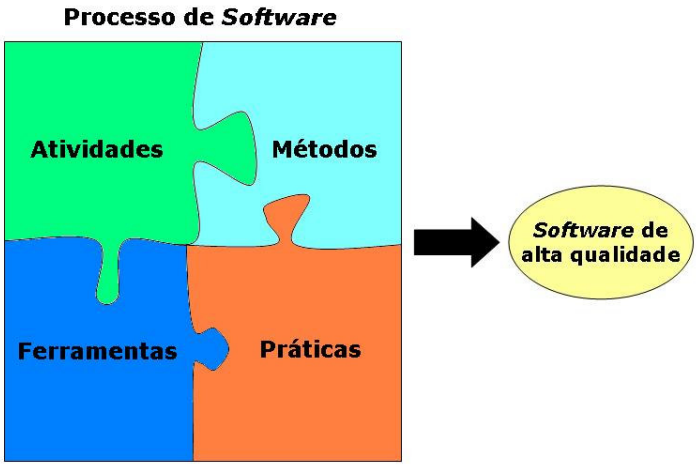


Figura 1.4: Definição de processo de *software*

Como o MUX-GSE é um sistema formado por componentes variados, que envolve uma equipe multidisciplinar, este trabalho se atém somente ao desenvolvimento de *software*, evitando informações sobre o restante do sistema, exceto aquelas que sejam relevantes para a compreensão do conteúdo, de modo que o texto não se alongue excessivamente.

O desenvolvimento de *software* será tratado desde seu início, considerando a escolha da equipe de programadores, da linguagem e plataforma de programação, do método de trabalho, as padronizações utilizadas, testes realizados e resultados obtidos. Serão apresentadas também as dificuldades encontradas ao decorrer do projeto e as soluções criadas para contorná-las.

Pode-se adiantar que o método proposto não é idêntico a nenhuma metodologia conhecida. É baseado no conhecimento adquirido durante o desenvolvimento de outros sistemas pelo Departamento de Pesquisa e Desenvolvimento (P&D) da Opto Eletrônica e pode ser classificado como ágil, pois segue todas as diretrizes definidas para estes métodos. Por isso, as práticas de desenvolvimento aplicadas serão apresentadas salientando as coincidências e diferenças em relação aos principais métodos ágeis, exaltando qualidades e apontando fraquezas identificadas.

Esta comparação servirá como base para permitir a localização de falhas na metodologia proposta, com o intuito de corrigi-las em trabalhos futuros, e para a caracterização detalhada do método utilizado, de modo que este possa ser extrapolado para outros projetos, desde que existam condições favoráveis para sua utilização.

Pretende-se ainda que este trabalho contribua mostrando que o desenvolvimento de *software* pode ser adaptado às necessidades e problemas de cada empresa, com flexibilidades para permitir adequações a cada projeto específico, de modo que o resultado prático, o *software*, seja obtido com qualidade, em tempo hábil e atenda às necessidades do cliente.

Por fim, acredita-se que este trabalho poderá servir como incentivo para que outros desenvolvedores, no início de um projeto de *software*, avaliem a viabilidade de aplicação de métodos ágeis. Isso possibilita a utilização da estratégia mais favorável ao trabalho que executam, ao invés da utilização do modelo tradicional simplesmente pelo fato de ser o mais difundido e, com isso, ajuda a reduzir os riscos de fracasso do projeto.

Cabe a seguinte ressalva: é importante notar que a utilização de um método ágil em um projeto ao qual ele não se enquadra perfeitamente pode trazer resultados desastrosos. A importância da difusão destes métodos é permitir aos responsáveis por projetos de *software* a opção pela solução que maximiza sua chance de sucesso, seja ela uma abordagem tradicional ou ágil.

1.3 Organização da Dissertação

Este documento é composto de cinco seções principais. A primeira delas, que termina aqui, trouxe um histórico resumido do Programa Espacial Brasileiro, mostrando o contexto em que se enquadra o trabalho atual e sua importância dentro do projeto MUX e em possíveis futuros trabalhos.

A segunda seção apresenta as metodologias de desenvolvimento mais conhecidas, ressaltando qualidades e defeitos e caracterizando particularmente os métodos ágeis, que são mais recentes e têm ganhado espaço muito rapidamente.

Na seção três é mostrado o processo de *software* utilizado para desenvolvimento do MUX-GSE, comparando as características com as dos métodos ágeis. Através dela devem ficar explícitos os pontos fortes e fracos da metodologia, e como as práticas propostas são implementadas, enquadrando-as nas características próprias da Opto Eletrônica.

A seção quatro apresenta o trabalho realizado, desde seu início, quando houve a definição da metodologia mais adequada e a justificativa desta escolha. Para cada atividade, há uma descrição, a apresentação dos resultados obtidos e conclusões específicas.

Por fim, a seção cinco apresenta as conclusões gerais do trabalho, os erros cometidos, as soluções apresentadas e idéias para tornar o método ainda mais efetivo, para possível aplicação em projetos futuros.

2 Metodologias Para Desenvolvimento de *Software*

2.1 Histórico

O *software* nasceu com os primeiros sistemas computacionais. Nesta fase inicial, com *hardware* bastante limitado, tinha sua importância reduzida. Era desenvolvido por um número pequeno de pessoas, que aplicavam metodologia própria e mantinham boa parte do projeto, senão todo ele, em suas próprias mentes.

Os primeiros métodos de desenvolvimento foram criados para aplicações militares e científicas, que utilizavam o que havia de mais moderno na época. Os programadores adotavam estratégias parecidas com as aplicadas no desenvolvimento do *hardware*, com as quais tinham familiaridade.

Com a evolução dos sistemas, especialmente com o surgimento dos dispositivos eletrônicos, acelerou-se a criação de *softwares* mais complexos, compostos de estruturas mais elaboradas e que, consequentemente, exigiam maior esforço de desenvolvimento. O avanço foi muito rápido (alguns autores consideram que houve uma nova revolução industrial ou uma terceira onda na história humana) (PRESSMAN, 2005), enquanto a disciplina de Engenharia de *Software* estava apenas se estruturando.

Nonemacher (2003) mostra que em uma segunda fase, surgiram as *"software houses"*, empresas especializadas na criação de *software*. O desenvolvimento passou a ser direcionado à distribuição e utilização em massa. Nesta época já existiam metodologias que se difundiram largamente. Em pouco tempo começaram a surgir críticas e alguns grupos isolados passaram a estruturar métodos alternativos.

Em uma terceira fase (NONEMACHER, 2003), o *software* ganhou mais relevância, sendo utilizado em uma vasta gama de equipamentos, desde satélites a eletrodomésticos. Sistemas distribuídos e redes ganharam dimensões mundiais, sistemas especializados passaram a ter grande valor agregado enquanto que o número de usuários domésticos cresceu rapidamente. O *hardware* perdeu importância, regredindo à condição de produto primário.

Ao mesmo tempo, os *softwares* cresceram em tamanho, funções, risco associado. Passaram a incorporar um número tão grande de estruturas, variáveis, classes, algoritmos e outros elementos que Brooks Júnior (1987, pág.11) afirma que "[...] são mais complexos [...] que qualquer outra construção humana, porque não possui duas partes iguais (pelo menos acima do nível de instrução)", diferentemente de computadores, prédios ou automóveis, que são formados por elementos repetitivos.

Neste cenário surgiu uma série de métodos alternativos de desenvolvimento. Com diversas características em comum, eles se derivam do conhecimento prático de desenvolvedores e pesquisadores experientes, desligam-se das características burocráticas do modelo tradicional, herança das técnicas militares, e buscam atingir a satisfação de clientes, empregadores e desenvolvedores em curto prazo, mas sem comprometer a qualidade do produto.

Por serem menos rígidos quanto à documentação e apresentarem características voltadas mais ao produto e às pessoas e menos ao processo, foram chamados de Métodos Leves, sendo posteriormente denominados "Métodos Ágeis". O termo "Métodos Tradicionais" passou a ser utilizado para designar as técnicas de desenvolvimento que seguem o modelo Cascata, mais antigo e conhecido, apresentado na subseção seguinte.

2.1.1 Modelo Cascata

A Engenharia de *Software*, em sua primeira fase, enfatizou o planejamento na esperança de tornar a codificação mais previsível. Assim, com definições bem feitas por analistas e projetistas, um programador poderia gerar o código descrito pelos diagramas, solucionando o problema apresentado da maneira pré-definida. "O objetivo desta abordagem é alcançar na área de *software* o mesmo nível de previsibilidade, determinismo e acerto presente em outros ramos da engenharia" (TELES, 2005, p. 48).

Estas idéias podem ser observadas no modelo mais difundido de desenvolvimento, conhecido como Cascata ou Clássico (Figura 2.1). Ele possui etapas bem definidas que ocorrem em seqüência, sendo que uma nunca se inicia antes que a anterior esteja completamente encerrada. A passagem de uma etapa à outra geralmente é marcada pela produção de um grande volume de documentos, que consolida os resultados obtidos até então.

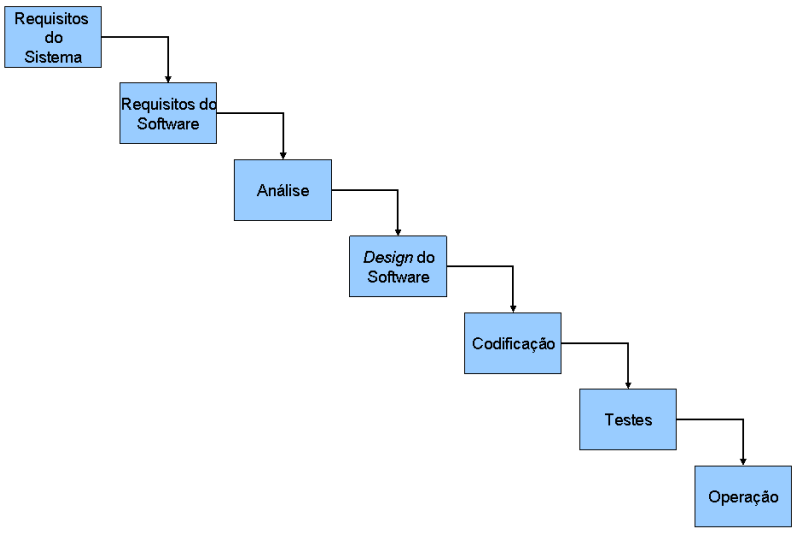


Figura 2.1: Desenvolvimento em Cascata

Segundo Vasconcelos (2004) e Racoon (1998), o modelo Cascata surgiu nos trabalhos realizados pelo Departamento de Defesa dos Estados Unidos no desenvolvimento de *hardware*[5]. Estes trabalhos foram adaptados ao desenvolvimento de *software* e se tornaram muito conhecidos a partir da Conferência da OTAN sobre Engenharia de *Software* (SOFTWARE ENGINEERING, 1968), considerada o marco de nascimento da disciplina de Engenharia de *Software*.

Segundo Teles (2004, 2005), o modelo Cascata remete aos trabalhos de Taylor[6], relacionando-se diretamente aos princípios da produção em massa, e Smith[7], pois prega a especialização dos trabalhadores em cada etapa. Pode-se então comparar o desenvolvimento de *software* com a construção de uma ponte ou a fabricação de um automóvel. Assim, o programador é considerado um profissional facilmente substituível, executando tarefas repetitivas, que não exigem muito raciocínio e cuja produtividade é diretamente proporcional ao tempo trabalhado, diferentemente do analista, que desenvolve trabalho intelectual.

Este modelo, apesar de ainda ser o mais utilizado, não é o ideal para a maioria dos projetos de *software* atuais. Ele possui diversas falhas bem conhecidas. É inadequado para situações em que os requisitos mudam durante o desenvolvimento, pois a documentação gerada a cada etapa torna-se obsoleta a cada alteração. Esta é uma característica muito prejudicial, pois "nos dias de hoje, eliminar mudanças cedo significa não reagir a condições de mercado [...]" (HIGHSMITH; COCKBURN, 2001, p.120). O modelo Cascata engessa o projeto, prejudicando adaptações indispensáveis, não só por falhas na especificação inicial, mas também por alterações nas necessidades do cliente.

---

[5] Muitos autores, entre eles Pressman (2005), atribuem equivocadamente a proposição do modelo Cascata a Winston W. Royce, que apresentou o trabalho "Managing the Development of Large Software Systems: Concepts and Techniques" na IEEE WesCon, em 1970. Neste trabalho, Royce mostra um método linear parecido com o Cascata, porém o critica, propondo um método iterativo.

[6] Frederick Wislow Taylor (1856 – 1915) desenvolveu teorias que visam maximizar a produção através da especialização e padronização do trabalho. *Princípios da Administração Científica*, de 1911, é um de seus trabalhos.

[7] Adan Smith (1723 – 1790). Em sua principal obra, *A Riqueza das Nações*, de 1776, defende a divisão do trabalho como forma de aumentar significativamente a produção.

É ineficiente também por não fornecer ao cliente uma visão clara do produto final durante um longo período de tempo (PRESSMAN, 2005). Os documentos das etapas de análise e *design* não permitem esta avaliação e a primeira versão utilizável só é entregue ao final de todo o ciclo. Com isto, a dificuldade de expressar as idéias do cliente através dos requisitos pode gerar *softwares* que não apresentem as características desejadas.

A falta de conhecimento dos analistas sobre o produto a ser desenvolvido e as falhas de comunicação entre eles e o cliente prejudicam também a definição da relevância das funções do *software*. Isto pode levar a equipe de desenvolvimento ao desperdício de tempo na criação de funções que serão muito pouco ou nunca utilizadas.

Outra parte do tempo é gasta na produção de funções, estruturas e outros elementos sem utilidade imediata, que servem para facilitar possíveis expansões futuras, já que modificações tardias são muito dispendiosas. Assim, analista, projetista e programadores empenham-se na criação de *software* que pode nunca ter utilidade, e que acaba tornando o produto ainda mais complexo (TELES, 2005).

Por fim, a estimativa de prazos neste modelo geralmente é falha. Os requisitos funcionais não são suficientes para a definição do tempo necessário à sua execução, nem mesmo para os desenvolvedores mais experientes. Além disso, a etapa de testes tem duração indeterminada, pois a detecção e a correção dos defeitos são totalmente imprevisíveis e, além dos erros de programação, podem existir problemas no *design*, identificados somente após a implementação do *software*.

Apesar de todas as críticas, não se pode descartar o modelo Cascata. Ele é adequado para algumas situações específicas e nestes casos deve ser utilizado. O fato é que a grande maioria das aplicações atuais pode ser desenvolvida com maior rapidez e com a mesma qualidade através de métodos baseados em outros modelos. Cockburn (1999) nos mostra que processos altamente disciplinados (em geral baseados no modelo Cascata) tendem a falhar em

aplicações comerciais, porém obtém sucesso em outros casos, como aplicações militares ou governamentais cujos requisitos são muito bem definidos e não se alteram durante o desenvolvimento e nos quais as equipes de trabalho são grandes.

2.1.2 Modelos Alternativos

Muitos dos problemas do modelo Cascata foram detectados ainda na década de 70, logo em seus primeiros anos de utilização mais ampla. Começaram, então, a surgir outras estratégias, cada uma com características que traziam benefícios em relação a determinadas falhas. Alguns modelos apresentaram idéias interessantes e conquistaram adeptos. Dois deles destacam-se, a Prototipagem Evolutiva e o modelo Espiral.

Estes modelos, além de serem bastante difundidos, são de especial interesse a este trabalho, pois foram utilizados diretamente no estudo de caso que será apresentado nos Capítulos 3 e 4, por isso serão apresentados resumidamente a seguir. Além destes, existem ainda diversos outros modelos que alcançaram sucesso, porém não tiveram a mesma abrangência e serão aqui omitidos.

2.1.2.1 Prototipagem Evolutiva

A Prototipagem Evolutiva, Evolucionária, ou simplesmente Prototipagem, é uma técnica empregada para ajustar e validar requisitos quando não se pode detectar a melhor alternativa para a solução de um problema nas fases iniciais. É, portanto, adequada quando não se consegue estabelecer com exatidão todos os requisitos do sistema. Pode ser empregada para avaliar a eficiência de um algoritmo, necessidades especiais do sistema ou detectar falhas

conceituais. Por isso, este modelo é muito utilizado no desenvolvimento de inovações tecnológicas, pois é difícil garantir os resultados antes que se realize uma bateria de testes.

Neste modelo, durante a definição dos requisitos, são selecionados resultados desejados, cuja solução não seja totalmente clara. Em seguida, um projeto simplificado é realizado e uma possível solução implementada. Este protótipo é utilizado para avaliar a solução e para que o cliente possa analisar claramente os resultados, solicitando as mudanças desejadas. Com as observações, os requisitos são refinados e o projeto refeito, incorporando as modificações pertinentes. O processo se repete várias vezes até que sejam obtidos resultados válidos.

Ao final desta etapa, os diversos protótipos gerados (para os diversos problemas selecionados) são utilizados para o desenvolvimento do sistema final. O ideal é criar o sistema utilizando outro modelo de desenvolvimento e os requisitos obtidos na Prototipagem, que serve como modelo complementar. É possível reunir os protótipos para que sirvam como base para o sistema final, desde que isto tenha sido definido no início do desenvolvimento, com todos os protótipos pensados e construídos para trabalharem em conjunto, porém isso prejudica a velocidade da execução das iterações. Caso contrário, corre-se o risco de utilizar a Prototipagem apenas como uma modificação da estratégia Codificar e Corrigir[8] (*"Code and Fix"*) (NATIONAL INSTRUMENTS, 2003).

Um dos principais defeitos da Prototipagem é a impressão deixada pelo protótipo de que o sistema já está funcionando (NATIONAL INSTRUMENTS, 2003). Isso pode levar o cliente a contar com o sistema completo antes que ele possa realmente ser implementado. Outro problema é o gasto excessivo de tempo na realização dos protótipos, por isso deve haver limites de tempo e/ou de iterações pré-definidos para evitar atrasos no projeto.

[8] Codificar e Corrigir é uma estratégia de desenvolvimento de software onde não existe planejamento. O programador vai gerando o código e corrigindo os defeitos até atingir os resultados desejados. Apesar de largamente utilizada, esta estratégia só obtém bons resultados em sistemas muito simples.

2.1.2.2 Modelo Espiral

O modelo Espiral (Figura 2.2), proposto em 1988 por Barry W. Boehm, atualmente está em evidência, pois se pode considerá-lo precursor dos modelos utilizados pelos métodos ágeis, apresentados na Seção 2.2. Entre as vantagens que apresenta estão adaptabilidade e realimentação (*"feedback"*) freqüente das opiniões do cliente. É muito vantajoso também no controle dos riscos oferecidos pelo sistema, uma vez que foi criado exatamente para este fim. Nele, as etapas de planejamento, modelagem, execução e distribuição se repetem diversas vezes nesta seqüência, gerando um *software* cada vez mais completo e buscando eliminar os riscos mais graves do projeto a cada iteração.

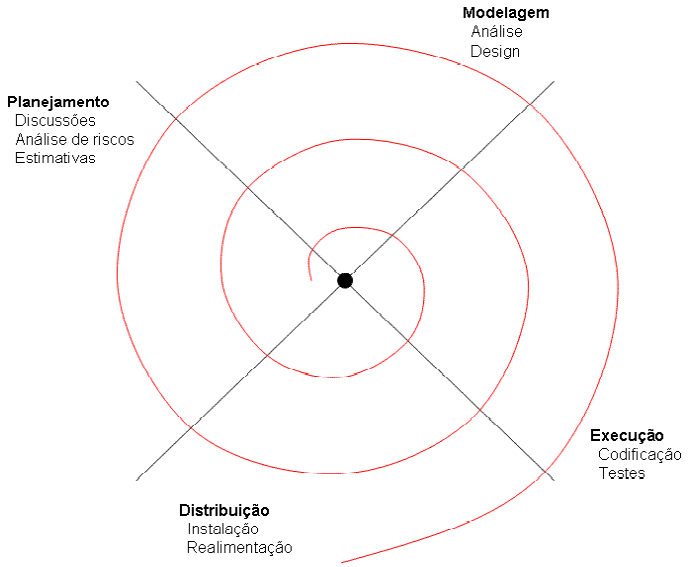


Figura 2.2: Modelo Espiral

A primeira etapa executada é o planejamento, que não tem a pretensão de definir todos os requisitos do sistema, nem de gerar uma agenda para o projeto. São definidos requisitos gerais do sistema e executada uma análise de riscos, utilizada para identificar os mais relevantes para execução imediata. Durante o planejamento também é avaliado o andamento do projeto e estimada a sua duração, a da próxima iteração e o número de iterações restantes.

A etapa seguinte é a modelagem. Somente os requisitos selecionados para iteração corrente são analisados. As soluções propostas visam criar um sistema executável que atenda aos requisitos atuais. A intenção é solucionar os riscos escolhidos sem prejudicar o sistema já desenvolvido até então.

A execução se divide em codificação e testes. A codificação visa incluir as soluções dos riscos selecionados, seguindo a modelagem do sistema. Os testes tentam assegurar a qualidade, tanto das novas funções implementadas quanto do *software* que já existia na anteriormente.

Ao final de cada iteração o *software* gerado é distribuído. Isto não significa necessariamente entregar uma versão completa (o que seria o ideal), mas ao menos apresentar o sistema ao cliente para testes e avaliação das funções desenvolvidas. Isto permite incorporar suas observações à próxima iteração na etapa de planejamento. Assim, a realimentação torna o sistema adaptável a novas condições de mercado e permite rápida detecção de erros na interpretação dos desejos do cliente.

2.2 Desenvolvimento Ágil

Nas décadas de oitenta e noventa, muitos métodos alternativos de desenvolvimento foram criados, cunhados a partir de práticas conhecidas, porém nem sempre muito aplicadas,

da Engenharia de *Software*. Estes métodos traziam uma série de características em comum e apresentavam forte contraste com a metodologia tradicional.

Em fevereiro de 2001, dezessete conhecidos desenvolvedores, consultores e pesquisadores que compartilhavam idéias semelhantes, reuniram-se para trocar experiências e discutir estratégias que pudessem aumentar a taxa de sucesso no desenvolvimento de *software*. O resultado desta reunião foi a assinatura do Manifesto Ágil (Figura 2.3), que propõe valorizar indivíduos e interações, *software* funcionando, colaboração do consumidor e respostas a mudanças mais que processos e ferramentas, documentação, negociação de contratos e planejamento rígido (BECK et al., 2001).

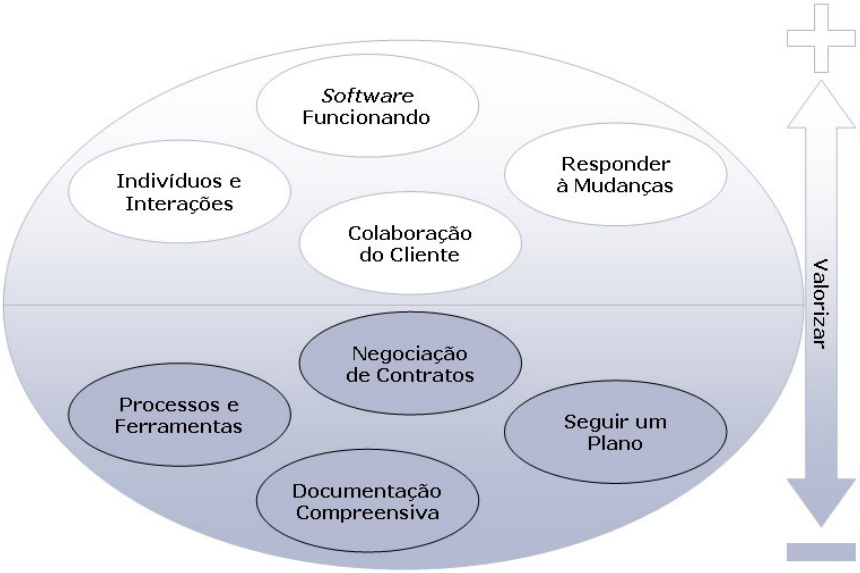


Figura 2.3: Representação das propostas do Manifesto Ágil

Os próprios signatários do Manifesto Ágil admitem a importância de todos os itens citados, porém o que propõem é uma maior valorização dos primeiros, ou seja, indivíduos, *software* funcionando, colaboração, resposta rápida a mudanças, etc. Este espírito, mesmo antes deste célebre encontro, já norteava as metodologias criadas e defendidas por estas personalidades, entre elas Programação Extrema (*"Extreme Programming"* ou XP), *Scrum*,

Método de Desenvolvimento de Sistemas Dinâmicos (*"Dynamic Systems Development Method"* ou DSDM), Desenvolvimento Adaptativo de *Software* ("*Adaptive Software Development"* ou ASD) e *Crystal*.

O movimento ágil, ao contrário do que possa parecer à primeira vista, não pretende que o desenvolvimento de *software* seja um processo caótico, guiado apenas pela vontade dos programadores, mas segundo seus próprios criadores, "[...] pretende restaurar a credibilidade da palavra 'metodologia'" (HIGHSMITH, 2001, grifo do autor).

Na verdade, com o desenvolvimento ágil, mais uma vez, a engenharia de *software* segue os passos de outros ramos da engenharia e passa a utilizar técnicas já empregadas com sucesso na produção industrial. Poppendieck (2001a, 2001b) mostra como todas as propostas mais importantes dos novos métodos se derivam da Manufatura Leve, proposta inicialmente pela Toyota, que levou a indústria japonesa a ultrapassar a norte-americana em diversas áreas. Estas propostas gerais, reafirmadas pelo Manifesto Ágil, serão apresentadas detalhadamente a seguir, com justificativas históricas e práticas que demonstram sua validade.

2.2.1 Indivíduos

Na metodologia tradicional, o trabalho de desenvolvimento de *software* era visto como repetitivo e braçal. A função do programador seria somente a de transformar em código o conteúdo dos diagramas e demais documentos gerados pelo analista e pelo projetista. Nesta abordagem, o processo é mais importante e as pessoas podem ser substituídas a qualquer momento. Ao contrário, "um processo ágil necessita de pessoas e organizações responsáveis. O desenvolvimento ágil foca-se nos talentos e habilidades dos indivíduos e molda processos para pessoas específicas [...]" (COCKBURN; HIGHSMITH, 2001, p. 131).

Teles (2004, 2005) mostra que o desenvolvedor é um trabalhador do conhecimento. Autonomia, responsabilidade sobre seu próprio serviço, oportunidade de inovar e contínuos aprendizado e ensino são fundamentais para garantir sua satisfação, produtividade e qualidade, enquanto que padronização, especialização, sincronização, concentração, maximização e centralização, fatores de produtividade do trabalho manual, podem não ter validade no desenvolvimento de *software*, ao contrário, podem prejudicá-lo.

Alistair Cockburn, um dos maiores estudiosos do fator humano no processo de desenvolvimento de *software*, em um de seus artigos mais conhecidos, afirma que praticamente qualquer metodologia pode funcionar em um projeto, porém os métodos leves (agora chamados ágeis) atingem o sucesso mais frequentemente (COCKBURN, 1999). O autor então conclui que isto não se deve à metodologia em si, mas às melhores condições de trabalho oferecidas pelos métodos ágeis. Os trabalhadores, com maior liberdade, tendem a mostrar suas virtudes e produzir mais por se sentirem felizes com o produto gerado.

Em um trabalho ainda mais completo, Cockburn avalia grande lista de referências, apresenta resultados obtidos em sua vasta experiência e conclui que "[...] podemos pensar na metodologia como 'as convenções de trabalho que a equipe escolhe seguir'" (COCKBURN, 2003, p. 68). Segundo ele, a quantidade de convenções necessárias cresce com o aumento do número de pessoas trabalhando no projeto.

Fowler (2000) afirma que não faz sentido o trabalho de um analista, projetista ou gerente definindo características do *software*, se os programadores são quem melhor entendem o que estão desenvolvendo. "Quando você quer contratar e manter boas pessoas, deve reconhecer que são profissionais competentes. Assim, elas são as melhores pessoas para decidir como conduzir seus trabalhos técnicos" (Fowler, 2000).

Mas a relevância e autonomia delegadas ao desenvolvedor não significam que a função do gerente esteja descartada. Blotner (2003) mostra que o gerenciamento bem feito

alivia os programadores de trabalhos que não lhes dizem respeito diretamente. Segundo ele, o gerente deve prover o correto ambiente de trabalho e ferramentas, transmitir progressos e problemas ao resto da empresa, e manter os programadores motivados e focados na criação do produto, sem deixá-los se esquecerem de prazos pré-estabelecidos. Ele defende que bons programadores são inerentemente perfeccionistas e o gerente precisa mostrar-lhes quando os resultados já são suficientes para o produto.

Segundo Fowler (2000), o pessoal técnico não pode resolver tudo sozinho, porém, o papel do gerente só tem valor quando ele tem muita proximidade com os desenvolvedores, permitindo e estimulando acesso contínuo ao conhecimento empresarial. Assim, a comunicação, tema da Seção 2.2.2, é o único modo de garantir a efetividade do trabalho gerencial.

Contudo, é importante lembrar que o gerente não deve nunca tomar as decisões técnicas e, portanto, deve dividir a liderança do projeto por igual com os programadores. Isto se deve à velocidade da mudança das tecnologias de desenvolvimento de *software*. Mesmo um programador, ao se tornar gerente, em pouco tempo perde suas habilidades técnicas, já que as tecnologias que domina tornam-se obsoletas rapidamente.

A importância dos programadores é corroborada por DeMarco e Lister (1990, p. 5, grifo do autor): "Os principais problemas de nosso trabalho não são de natureza *tecnológica* mas sim *sociológica*". Esta afirmação resulta de uma pesquisa realizada pelos autores com mais de 500 históricos de projetos, desde 1977, dos quais 15% falharam. Na grande maioria destes, segundo eles, "[...] *não havia um único aspecto tecnológico para explicar o fracasso*" (DEMARCO; LISTER, 1990, p. 4, grifo do autor). Enquanto os responsáveis pelos projetos se empenhavam na área técnica, esqueciam-se de seu bem mais valioso: as pessoas.

A Tabela 2.1 traz um resumo das diferenças nos enfoques tradicional e ágil em relação ao trabalho de desenvolvimento de *software*:

Tabela 2.1 – Trabalho de Desenvolvimento de Software – Abordagem Tradicional x Ágil

Característica	Abordagem Tradicional	Abordagem Ágil
Tipo de trabalho	Repetitivo, braçal	Criativo, mental
Programador	Facilmente substituível, constrói código previsto pelo analista	Autônomo, responsável, competente
Fatores de produtividade	Padronização, especialização, sincronização, concentração, maximização e centralização	Oportunidade de inovar, aprendizado e ensino contínuos, satisfação pessoal, qualidade de vida
Relevância das pessoas	Segunda ordem, linear	Primeira ordem, não linear
Natureza dos problemas	Tecnológica	Sociológica
Função do gerente	Coordenar o trabalho dos programadores	Fornecer as condições ideais de trabalho aos programadores

As empresas americanas há algum tempo já têm esta visão dos trabalhadores do conhecimento como parte do ativo da companhia. Eles não apenas carregam sua experiência, mas são os próprios meios de produção, por isso, cada vez mais empresas têm oferecido a seus programadores planos de premiação que incluem distribuição de ações e remunerações extra.

No Brasil, esta mudança é mais lenta, mas segue o mesmo rumo. A Pesquisa de Qualidade e Produtividade no Setor de *Software* Brasileira, realizada pelo Ministério da Ciência e Tecnologia, traz dados que comprovam o interesse das empresas de desenvolvimento de *software* no bem estar de seus funcionários, fórmula que reduz a rotatividade e aumenta a produtividade. A Tabela 2.2 traz algumas destas informações, relativas ao ano de 1999. O estímulo ao aprendizado contínuo, outro fator importante de redução na rotatividade, aumento de produtividade e auto-estima entre os trabalhadores do conhecimento, fica explicitado pelos dados da Tabela 2.3.

Tabela 2.2 – Benefícios oferecidos por empresas brasileiras de desenvolvimento de *software*

Benefícios	Todas as empresas de desenvolvimento	Empresas de *software* embarcado
Jornada de trabalho flexível (para todos os desenvolvedores)	51,7%	72,0%
Investimento em treinamento na área de engenharia/tecnologia de *software*	54%	64,0%
Realização de pesquisas formais de satisfação dos funcionários	24,3%	40,0%

Observação: Pesquisa realizada no ano de 1999, com 446 empresas entrevistadas.
Fonte: Ministério da Ciência e Tecnologia (1999)

Tabela 2.3 – Formas de atualização da força de trabalho em empresas de desenvolvimento de *software* brasileiras em 2001

Categorias		N° de organizações	%
Acesso à internet	Livre	334	75,4
	Restrito	91	20,5
Material especializado	Aquisição de publicações	339	76,5
	Assinatura de periódicos	312	70,4
Liberação para Congressos e afins	Com ônus	154	34,8
	Sem ônus	233	52,6
Liberação para cursos	Com ônus	187	42,2
	Sem ônus	272	61,4
Incentivo à pós-graduação	Com ônus	183	31,2
	Sem ônus	99	22,3
Incentivo à publicação de trabalhos técnicos e relatos de experiências		91	20,5
Treinamento interno		5	1,4*
Outras		5	1,1
Não adota		15	3,4
Base		**443**	**100**

Observação: o valor 1,4 consta, aparentemente de maneira incorreta, do documento referenciado. A tabela e os textos complementares não trazem errata e nem informações que justifiquem este número.
Fonte: Ministério da Ciência e Tecnologia (2001)

Estes números mostram que as empresas brasileiras vêm assimilando a idéia de que desenvolvedores são trabalhadores do conhecimento, seja através de conhecimento trazido de companhias estrangeiras, seja pela observação de seus próprios defeitos. Aceitar esta

definição e adaptar-se é fundamental para qualquer empresa que queira se manter no competitivo mercado de desenvolvimento.

Fowler (2000), engenheiro eletrônico que se especializou em *software*, afirma que pessoas brilhantes e capazes têm sido atraídas ao desenvolvimento de *software* pelo desafio da carreira e a possibilidade de grandes retornos financeiros. E completa:

> Se você espera que seus desenvolvedores sejam unidades de programação substituíveis, você não tenta tratá-los como indivíduos. Isso diminui o moral (e a produtividade). Os bons profissionais procuram por um lugar melhor e você termina exatamente com o que desejou: unidades de programação substituíveis. (Fowler, 2000).

Segundo DeMarco e Lister (1990, p. 122) "o custo total da substituição de cada pessoa é equivalente ao custo do empregado durante quatro meses e meio a cinco meses [...]", considerando somente as perdas mais óbvias, assim, o alto preço da rotatividade justifica o empenho das empresas em manterem seus funcionários.

2.2.2 Interações

Comunicação é uma das características principais de qualquer método ágil. A interação entre programadores, cliente, gerente, e todas as outras pessoas envolvidas em um projeto de *software* é o meio mais eficaz de fazer com que o produto gerado atinja as metas esperadas. É a ligação entre as etapas e iterações do projeto, o caminho pelo qual a realimentação chega aos programadores e o substituto do grande volume de documentos gerado nos métodos tradicionais.

A rede de interações permite a difusão do conhecimento entre os membros da equipe, fazendo com que o nível de trabalho de todos seja maximizado e tornando a substituição de um deles, quando inevitável, menos penosa, já que o programador que sai não carrega consigo dados não conhecidos pelos colegas e o novo membro tem acesso às informações e ao conhecimento de todos os demais.

Para que a comunicação seja clara e precisa (Figura 2.4) deve ser direta, presencial, de modo que estejam presentes elementos verbais, visuais e temporais. Entonação de voz, expressões faciais e corporais, possibilidade de discussão em tempo real, etc. fazem do diálogo o meio mais eficaz de se trocar informações (COCKBURN, 2002; TELES, 2004). É importante também que seja possível expressar idéias através de diagramas, desenhos ou representações. Um quadro negro é ideal para isso, mas uma folha de papel pode resolver o problema satisfatoriamente.

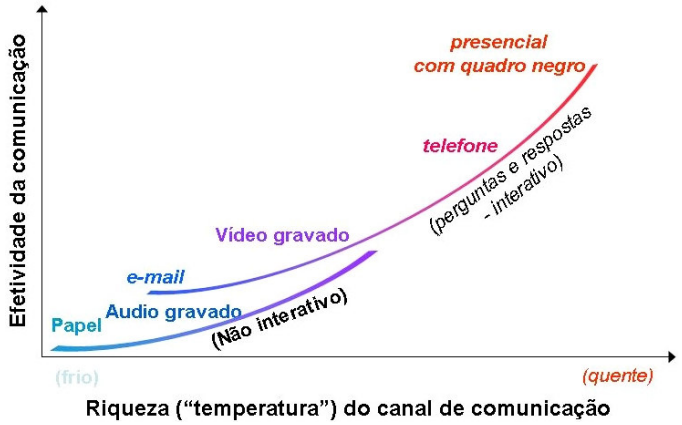


Figura 2.4: Efetividade da comunicação
Adaptado de Cockburn (2002)

A comunicação menos eficaz, o oposto do diálogo diante do quadro negro, é aquela que não permite perguntas e respostas instantâneas nem elementos emocionais, expressões visuais e sonoras, ou seja, a comunicação através do papel. Por isso a documentação gerada no modelo tradicional, mesmo quando bem feita e atualizada (o que é raro), é pouco efetiva na transmissão das informações. Por isso os métodos ágeis recomendam somente a produção de documentos realmente imprescindíveis (como mostrado na Seção 2.2.5).

Os métodos ágeis exigem postura colaborativa, o que só pode ocorrer baseado em constante comunicação e distribuição de conhecimento. Somente assim o ensino e aprendizado contínuos têm lugar e os programadores conseguem manter o mesmo nível e

estilo de código, gerando o melhor produto possível de acordo com a capacidade do grupo. O diálogo é a linha de comunicação que permite esta postura.

Uma maneira de se garantir o diálogo presencial é eliminar qualquer barreira física entre os programadores. Assim, o ideal é que todos dividam uma única sala, com espaço apropriado para a quantidade de pessoas e boa colocação das mesas, para que todos possam se distribuir conforme a necessidade (BECK, 1999a). As outras pessoas envolvidas no projeto, como o cliente e o gerente, devem estar disponíveis quando os programadores precisarem, em horários pré-programados ou em tempo integral.

É fundamental também que todos os desenvolvedores recebam as mesmas informações. Por isso, reuniões rápidas e freqüentes são estimuladas. Elas não devem ter longa duração para que não prejudiquem o andamento dos trabalhos e precisam ocorrer sempre visando evitar retenções de dados. Algumas devem ocorrer somente entre os programadores, outras devem contar com o gerente e outras ainda com o cliente.

Os pesquisadores salientam também a importância de reuniões informais. Segundo Cockburn (1999), IBM e Hewlett-Packard foram os primeiros a notar a importância de locais específicos para discussões de pequenos grupos, como áreas reservadas ao café ou lanche, onde as pessoas podem conversar livremente. Estas conversas permitem trocas de experiências e exposições de diferentes pontos de vista sobre o projeto, o que muitas vezes produz soluções inovadoras.

2.2.3 *Software* Funcionando

Os métodos ágeis propõem distribuições de *software* utilizável com freqüência e em curtos espaços de tempo, ou seja, "[...] que porções pequenas, mas completas de um sistema sejam desenvolvidas e entregues através de todo o ciclo de desenvolvimento [...]"

(POPPENDIECK, 2001a). Esta estratégia é benéfica para o consumidor e para a equipe de desenvolvimento e fundamental para garantir a reação rápida às mudanças de mercado e a detecção precoce de erros de especificação e implementação. Segundo Teles (2004), o lançamento freqüente de versões utilizáveis traz dois resultados importantes: geração de valor e gestão de riscos.

As distribuições devem consistir de *software* capaz de solucionar partes dos problemas especificados, ou seja, que tenham função prática, ainda que não todas as da versão final (Figura 2.5). A seleção da seqüência de implementação das funções cabe ao cliente, assim ele aponta o que é mais importante para que seja realizado nas primeiras distribuições, agregando o máximo valor ao *software* em curto espaço de tempo.

Figura 2.5: Comparação entre distribuição completa (abordagem tradicional) e distribuições freqüentes (abordagem ágil)

De posse de uma versão utilizável, o cliente pode disponibilizá-la ao usuário final, para que este avalie o *software* da maneira mais completa possível: a aplicação prática. Nenhum outro teste provê visão tão clara do sistema em desenvolvimento. O cliente e/ou

usuário final pode(m) utilizar o que já foi feito e ter boa noção do que ainda será desenvolvido, redirecionando o projeto conforme suas reais necessidades. Quando uma função é implementada de maneira diferente da desejada, seja por erros na especificação (problemas de comunicação) ou no *design*, a falha é rapidamente detectada e sua correção pode ser feita antes que implique em grandes modificações em um sistema complexo, reduzindo custos de manutenção.

Esta análise pode levar o cliente a rever o valor de algumas funções e modificar as especificações iniciais, solicitando a criação de procedimentos que tenham mais relevância na solução de seus problemas. Isto não deve ser mal visto pelos desenvolvedores, mas aceito como algo natural no processo criativo de um *software*. Dificilmente o cliente tem conhecimento suficiente para apresentar com precisão suas necessidades no início das atividades. O mesmo ocorre com os desenvolvedores, que passam a compreender melhor os problemas tratados com o decorrer dos trabalhos.

As informações precisas transmitidas pelo cliente, aliadas à compreensão mais apurada do produto por parte dos desenvolvedores e o conjunto das demais estratégias que asseguram respostas rápidas a mudanças, garantem a redução nos riscos do projeto. O cliente, mais satisfeito, recebe o resultado de seu investimento, enquanto que o empregador e os desenvolvedores têm confiança de estarem realizando exatamente o que se espera deles.

Contra-exemplos demonstram a importância desta prática: DeMarco e Lister (1990) relatam projetos com duração de mais de vinte e cinco anos, dos quais 25% nunca se completaram. Provavelmente, a maioria destes fracassos seria evitada se fossem promovidas distribuições freqüentes e mesmo que os projetos fossem cancelados, o esforço de desenvolvimento e o investimento não seriam totalmente perdidos.

Entretanto, para que as distribuições não prejudiquem o bom andamento do desenvolvimento, não podem existir muitas exigências de documentação e certificação de

versões, ou será necessário muito esforço na realização de testes e geração de papéis que terão que ser revistos a cada distribuição. Assim, o cliente deve aceitar esta condição para que possa desfrutar dos benefícios de ter versões funcionais em curto espaço de tempo e de direcionar o andamento do projeto.

2.2.4 Colaboração do Cliente

O cliente tem papel importante nos métodos ágeis, podendo ser considerado parte da equipe de desenvolvimento. É ele, junto do usuário final, o responsável pela realimentação do sistema, ou seja, pela avaliação do *software* gerado, apontamento de falhas e solicitação de modificações. Ele tem também a responsabilidade de definir a seqüência de implementação, ou seja, a prioridade, das funções. Sua participação ajuda a equipe a atingir os objetivos e conseqüentemente garante a sua própria satisfação. A Figura 2.6 apresenta resumidamente sua atuação em todas as etapas do processo de desenvolvimento.

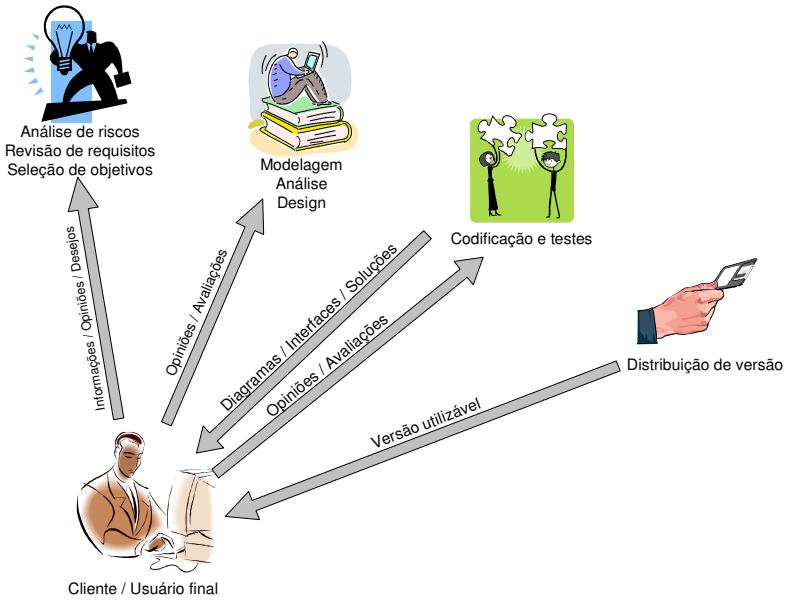


Figura 2.6: Participação do cliente no desenvolvimento ágil

A presença constante do cliente é uma fonte importante de informações sobre o produto. A interação entre ele e os desenvolvedores une diferentes experiências e conhecimentos, o que permite a criação de *software* de melhor qualidade, utilizando as ferramentas e estratégias mais adequadas e baratas (HIGHSMITH; COCKBURN, 2001). Reduz ainda consideravelmente a ocorrência de erros de especificação e *design*, pois cliente e desenvolvedores compreendem melhor as idéias uns dos outros.

Como mostrado na Seção 2.2.3, as avaliações do cliente se tornam ainda mais eficazes com as distribuições freqüentes do *software*, que permitem a detecção de erros e rápida correção, antes que isto se torne um processo complexo e caro. Entre as distribuições, o projeto deve ser apresentado de maneira clara, para que o cliente possa compreender o que foi realizado e acelere ainda mais a realimentação, apresentando opiniões consistentes antes que possa testar na prática o produto. Para completar, o relacionamento constante traz confiança a ambas as partes. Segundo Teles (2004, p. 61), tem-se: "a questão da confiança é uma das mais sérias em desenvolvimento de *software* e em qualquer prestação de serviço."

O contato com o cliente, para garantir todos estes benefícios, deve ocorrer rotineiramente, em reuniões presenciais, com a participação de todos os desenvolvedores. Os resultados alcançados devem ser apresentados, os próximos passos discutidos, as especificações globais revistas e os riscos do projeto avaliados. O cliente deve divulgar sua visão sobre o que já foi realizado e o que se pretende fazer e as mudanças observadas em seu negócio e em suas necessidades, além de decidir as próximas funções a serem implementadas. Reuniões gerenciais também podem contar com a presença do cliente, sem a participação dos programadores, porém sem a discussão de questões técnicas.

2.2.5 Documentação Leve

Boa parte da agilidade dos novos métodos se deve justamente à sua documentação, que se restringe ao necessário para o bom andamento do projeto, ou seja, os documentos utilizados para compreensão do *software*, permitindo sua manutenção futura. Parte dos diagramas e textos utilizados em excesso nos métodos tradicionais é substituída pela boa rede de comunicação e pelo profissionalismo dos desenvolvedores.

Isso é possível principalmente devido a uma diferença fundamental no enfoque das novas metodologias. Enquanto o modelo tradicional separa claramente o *design* da construção, com tarefas distintas para analistas, projetistas e programadores, nos métodos ágeis, todo o desenvolvimento de *software* é considerado uma tarefa de *design* e a produção compreende somente a compilação, montagem, empacotamento e distribuição do *software*. Assim, uma única equipe realiza o projeto e a codificação e não há necessidade de se gerar grande quantidade de informações para que outras pessoas, menos capacitadas, implementem o sistema (FOWLER, 2000).

Com isso, todos os documentos de *design* são muito simplificados. E mais, como é muito provável que os requisitos sofram alterações, os documentos não terão validade até o final do projeto. Sua importância restringe-se ao ciclo atual. Em alguns casos, não existe nem mesmo padronização para sua produção: cada programador é responsável por gerar sua própria representação, da maneira que lhe for mais conveniente, já que ela restringe-se à porção de código que ele desenvolve e não será reaproveitada no futuro.

Para que isso seja possível, o código deve ser muito claro, construído com boas práticas de programação. Comentários devem ser utilizados para facilitar a compreensão de partes que não sejam óbvias, porém sem exagero, evitando informações redundantes, que

poluem o arquivo. O código é sempre o documento principal de um projeto. Em última instância, é através dele que todos os programadores, agora e no futuro, devem compreender o sistema. Assim, deve-se atentar em sua geração, para que se mantenha um padrão, qualquer que seja, utilizado por todos os membros da equipe. "A adoção de um padrão ajuda a simplificar a comunicação [...] e a tornar o código coletivo." (TELES, 2005, p.91).

Outros documentos de projeto, como requisitos, resultados de testes, decisões do grupo, informações sobre o produto, etc. podem ser registrados de diversas maneiras, porém nunca devem ser considerados definitivos. Em um método ágil, tudo pode mudar com o tempo. Os registros utilizados devem permitir estas alterações sem representar obstáculo. Não se pode, entretanto, ignorá-los. Cockburn (2002) alerta que as informações deixadas após o encerramento do projeto devem ser suficientes para uma próxima equipe continuar os trabalhos, mesmo que sejam "[...] documentos incompletos e processos malfeitos [...]" (COCKBURN, 2002, p.175). Deve-se então, confiar na qualidade da nova equipe para compreender os dados e aproveitar o que for relevante, beneficiando-se da boa qualidade do código.

Manuais, ajuda (*"help"*), relatórios de desenvolvimento, testes de certificação, etc. são gerados ao final de cada distribuição, quando pertinentes. O volume destes documentos é minimizado, pois eles terão que ser revisados toda vez que uma nova versão utilizável do *software* for distribuída. Destes, os testes de certificação são os que mais variam em quantidade, e que mais podem gerar empecilho no lançamento de versões. Quando exigidos em excesso, aumentam os custos e reduzem a freqüência das distribuições, elevando os riscos do projeto, efeito oposto ao esperado. Assim, sua eficácia e necessidade devem ser cuidadosamente discutidas em cada projeto.

A geração excessiva de documentos, não só enrijece o projeto, como aumenta os custos de desenvolvimento, pois "[...] o tempo que se gasta documentando é um tempo que se poderia estar sendo usado para codificar" (TELES, 2004, p.210).

> Existem muitos gastos associados com esta documentação excessiva: o desperdício de tempo produzindo os documentos e revisando os documentos, e o trabalho desnecessário envolvido na mudança de solicitações e avaliações associadas, ajustes de prioridade, e mudanças no sistema. (POPPENDIECK, 2001a).

Além disso, a visão defensiva de que o armazenamento de todas as informações possíveis pode garantir o sucesso do projeto, tem sido já refutada há muito tempo. Segundo Teles (2004), estes documentos só teriam utilidade para isentar a equipe de alguma responsabilidade no caso de uma falha no projeto; como a intenção é atingir os objetivos, eles são desnecessários. DeMarco e Lister enfatizam: "**A documentação volumosa é parte do problema, não da solução**." (DEMARCO E LISTER, 1990, p. 135, grifo do autor).

Assim, a questão chave da documentação nos métodos ágeis é encontrar a quantidade adequada que deve ser produzida. Tanto o excesso quanto a escassez são prejudiciais e não existe fórmula para calcular o volume necessário. Segundo Cockburn (2002), o que é suficiente para uma equipe pode ser insuficiente ou excessivo para outras. Assim, mais uma vez, cabe à equipe de desenvolvimento definir o quanto de documentos deve produzir.

Do ponto de vista do cliente, muitas vezes, pode haver certa insegurança diante desta escassa documentação, principalmente se ele estiver habituado ao grande volume de informações geradas nos métodos tradicionais. É preciso convencê-lo dos benefícios desta estratégia: produto mais barato, efetivo, com menor risco e em menor tempo. A distribuição rápida de *software* utilizável (Seção 2.2.3) é fundamental neste aspecto e o contato permanente com os desenvolvedores (Seção 2.2.4) estimula a confiança mútua.

2.2.6 Responder a Mudanças

A necessidade de se adaptar durante o desenvolvimento é justamente o que levou diversos profissionais a buscarem alternativas ao desenvolvimento tradicional. Segundo Poppendieck (2001a), "práticas de desenvolvimento de *software* que mantêm os requisitos flexíveis [...] provêem uma vantagem competitiva. Em um ambiente comercial volátil, os usuários não podem prever precisamente suas necessidades futuras."

A Figura 2.7 apresenta uma comparação entre as duas abordagens. A ilustração do modelo tradicional como um sistema de controle de malha aberta pode ser um pouco exagerada, pois ele permite alguma realimentação. Porém, neste modelo, qualquer mudança é dificultada por características como o excesso de documentação, demora na realização de distribuições e equipes especializadas para cada etapa do processo, que tornam mudanças difíceis e caras. Com isso, a tendência é que qualquer alteração de requisitos seja evitada.

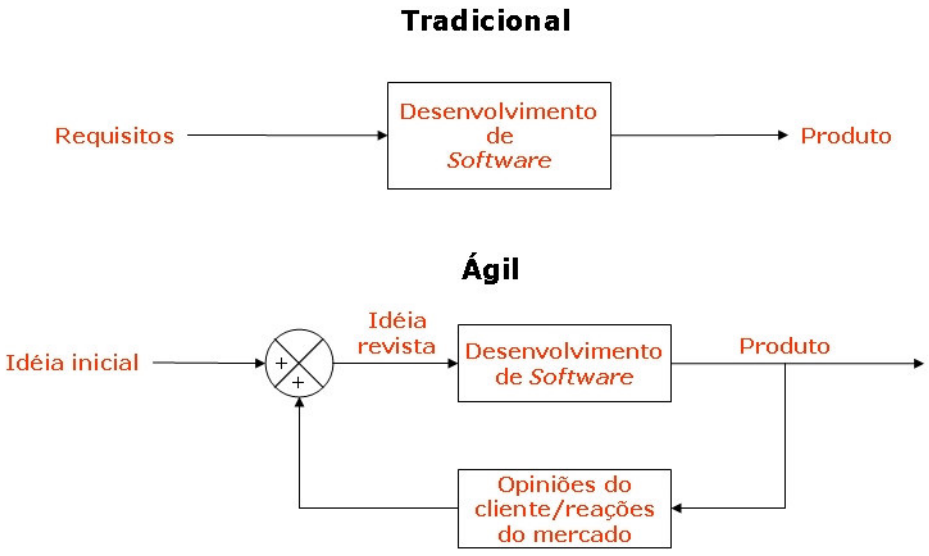


Figura 2.7: Metáfora comparando a abordagem tradicional e a ágil com sistemas de controle de malha aberta e malha fechada, respectivamente

Os métodos ágeis, por outro lado, não só aceitam modificações nas especificações a qualquer momento, mas as incentivam. Eles foram criados justamente com esta intenção. Todas as características apresentadas até aqui contribuem fortemente para isso, mas isoladamente suas ações são limitadas. Na verdade, é o conjunto delas que permite adaptação rápida e eficaz às mudanças.

A centralização nos indivíduos e o estímulo ao aprendizado e ensino constantes fazem com que a equipe esteja apta a lidar com situações inesperadas e possa buscar conhecimento para solucionar os problemas que surgem no decorrer do desenvolvimento. A interação constante entre desenvolvedores, gerente, usuário final e cliente garante a difusão das informações, a distribuição do conhecimento e proporciona alto nível e simplicidade nas soluções propostas.

A colaboração do cliente, facilitada pelo bom conhecimento do sistema proporcionado pelas distribuições freqüentes de *software* funcional e possibilitada pelas reuniões periódicas com a equipe de desenvolvimento, auxilia a compreensão de seus desejos pelos programadores, o que resulta em melhor direcionamento dos trabalhos, em busca de solucionar adequadamente os problemas apresentados, começando pelos mais relevantes. Da mesma forma, qualquer modificação apresentada pelo cliente é tratada rapidamente aproveitando os mesmos mecanismos.

A documentação visa permitir modificações rápidas. O código claro, comentado com ponderação e a ausência de relatórios, manuais, diagramas, etc. em excesso aceleram a execução de mudanças, já que os programadores preocupam-se apenas com o produto principal, sem a necessidade de revisar conteúdo inútil. Este enfoque só se sustenta com excelente comunicação entre os membros da equipe, como mostrado na Seção 2.2.5, e necessita da colaboração do cliente.

- **Contratos com Escopo Variável**

Apesar deste conjunto de fatores, problemas aparentemente simples podem ser de difícil solução e prejudicar a condução de um projeto ágil. A definição do contrato de prestação de serviços é um destes casos. Em geral, os contratos especificam escopo, custo e prazo, entretanto, como definir o escopo se os requisitos podem (e devem) mudar?

A solução mais aplicada é a definição de contratos com escopo variável, nos quais o custo e o prazo são definidos. Em geral, assina-se um contrato curto que é renovado se ambas as partes estão satisfeitas. Esta idéia pode parecer absurda à primeira vista, mas vários autores a defendem, entre eles Teles (2004), Fowler (2000) e Poppendieck (2002), e existem muitos estudos de caso comprovando sua eficácia. Segundo Beck e Cleal (1999), este tipo de contrato alinha os interesses de consumidores e fornecedores e tendem a evitar o sacrifício na qualidade que geralmente ocorre quando o projeto não segue os prazos pré-estabelecidos.

Porém, para a assinatura de um contrato deste tipo são necessárias a confiança e a colaboração do cliente, pois é preciso convencê-lo que a equipe trabalhará para lhe fornecer um produto de alta qualidade e que satisfaça suas necessidades. Sem este nível de colaboração, a implementação de um método ágil pode tornar-se inviável.

Vários trabalhos já foram apresentados sobre este assunto e workshops específicos[9] acontecem regularmente em eventos relevantes. A tendência é que contratos com escopo variável se tornem usuais, pois não existe até o momento nenhuma outra proposta que solucione aceitavelmente este empecilho.

[9] Por exemplo:
AGILE CONTRACTS WORKSHOP. In: XP 2003, 2003, Gênova, Itália.
AGILE CONTRACTS WORKSHOP. In: AGILE DEVELOPMENT CONFERENCE, 2003, Salt Lake City.
AGILE CONTRACTS WORKSHOP. In: OOPSLA 2003, 2003, Anaheim, EUA.

2.2.7 Características Específicas

Existem várias metodologias ágeis, todas compartilhando as premissas fundamentais já apresentadas, entretanto, cada uma tem características específicas. Alguns destes diferenciais referentes a três dos métodos ágeis mais conhecidos, Programação Extrema (*"Extreme Programming"* – XP), *Scrum* e *Crystal*, serão explicitados a seguir. O objetivo desta rápida apresentação não é descrever detalhadamente cada método, mas apenas apontar o que os distingue das idéias mostradas anteriormente.

2.2.7.1 Programação extrema (XP)

A XP é a metodologia de desenvolvimento ágil com maior número de adeptos. Sua difusão ocorreu principalmente a partir de 1999 com o lançamento do livro "*Extreme Programming Explained: Embrace Change*" (BECK, 1999a), de Kent Beck, que além de ser um dos criadores do método, ao lado de Ward Cunningham, Ron Jeffries e outros, é considerado o principal responsável pela sua grande aceitação, devido à liderança e persuasão que exerce e à divulgação que conseguiu promover.

XP baseia-se em valores e práticas que, seguidos pelas pessoas envolvidas no projeto (cliente e usuário final também participam), permitem grande nível de adaptabilidade e confiabilidade. É considerada ideal para projetos com requisitos que mudam ao longo do tempo e nos quais trabalham poucos programadores (até doze é uma quantidade considerada aceitável).

Os valores de XP, realimentação, comunicação, simplicidade, coragem e respeito, e suas práticas promovem a aplicação de todas as características ágeis apresentadas até aqui. É, na verdade, o método que adota estas características da maneira mais extrema, fazendo jus a seu nome. Além disso, XP propõe:

- **Programação em par**

  A programação em par (*"pair programming"*) ou programação pareada é uma técnica de desenvolvimento que consiste no trabalho simultâneo de dois programadores em um único microcomputador, gerando código em conjunto. É defendida por muitos autores, porém não é obrigatoriamente aplicada em todos os métodos ágeis.

  É uma técnica cuja implementação encontra grande resistência por parte de gerentes, diretores, etc., que dificilmente se convencem que suas vantagens compensam a utilização de dois programadores onde, à primeira vista, somente um seria suficiente. Seus defensores, entretanto, afirmam que a grande redução no número de erros tanto de *design*, que é continuamente construído, quanto de implementação, promovida por sua aplicação compensa os custos.

  Dois programadores trabalhando juntos produzem mais rapidamente e são mais objetivos, reduzindo a diferença de custo entre a programação em par e a tradicional. Cockburn e Williams (2001) mostram que o acréscimo nos gastos é em torno de 15%, o que não é muito considerando os prejuízos que a detecção tardia de um erro pode provocar.

- **Integração contínua**

Integração contínua refere-se a unir novas partes construídas de *software* ao programa principal em curtos intervalos de tempo. Sempre que ocorre este evento, são então realizados testes que verificam o funcionamento das novas funções inseridas e também das já existentes para assegurar que estas não foram prejudicadas pelas alterações.

Esta prática é comum a todos os métodos ágeis, a diferença em XP é a freqüência com que é executada: várias vezes ao dia. Os programadores são estimulados a integrar o código produzido sempre que uma nova funcionalidade é concluída. Para facilitar esta ação, deve haver um microcomputador exclusivo para integração, utilizado pelos programadores somente quando estão unindo seu código ao sistema e realizando os devidos testes. Isto evita que dois pares de programadores modifiquem o programa principal ao mesmo tempo.

- **Desenvolvimento orientado a testes**

Segundo Beck, “Em XP [...], duas mudanças nas estratégias convencionais de teste os tornam muito mais efetivos: os programadores escrevem seus próprios testes e eles escrevem estes testes antes de codificarem” (BECK, 1999b, p. 73). Isto significa que para qualquer funcionalidade produzida, deve-se em primeiro lugar criar uma rotina automática que assegure seu funcionamento, fornecendo as possíveis entradas e verificando a validade das saídas. Somente depois disto a função é implementada. O código de testes é mantido durante todo o projeto e serve como garantia de funcionamento da parte a que se refere.

Além destes, existem testes propostos pelo usuário/cliente a cada iteração, que verificam requisitos do sistema a serem implementados. Alguns dos programadores da equipe são responsáveis pela codificação destes testes sempre que o proponente não tiver conhecimento técnico para fazê-lo, sendo a grande maioria dos casos.

Todas as vezes que o *software* é integrado, todos os testes construídos são executados e devem resultar em 100% de aprovação. Se algum teste indicar erro, os programadores precisam localizar a falha e corrigi-la antes de continuarem o trabalho. Esta tarefa é facilitada pela simplicidade do código e pelo fato do teste referir-se somente a uma função do sistema, apontando claramente a parte do código que não está funcionando perfeitamente.

Apesar desta prática não ser padrão em outros métodos, é muito defendida e considerada um dos principais fatores de qualidade de *software*. Geras, Smith e Miller (2004), entretanto, mostram que para implementá-la, apesar dos benefícios, é necessário treinamento de pessoal, especialmente das pessoas responsáveis por codificar os testes propostos pelo consumidor, para evitar projetos mal feitos, que não prevêem todas as possibilidades de falha e que, portanto, podem aprovar *softwares* defeituosos.

- **Código coletivo**

Em XP, qualquer programador pode modificar livremente o código quando está implementando uma função ou quando detecta partes que podem ser melhoradas, independentemente de ter sido o criador ou não da porção modificada. Esta prática auxilia na padronização do código e em sua qualidade, pois todos os membros da equipe o revisam e a tendência é que ele sempre atinja a máxima otimização possível, dentro da capacidade de todos os programadores.

O grande problema desta prática é a possibilidade de um par de programadores prejudicar a execução de alguma função ao tentar otimizar o código. Isto é amenizado pela existência de testes para todas as funcionalidades do programa, que são 100% executados a cada integração. Assim, eles têm certeza do funcionamento de todo o código após suas modificações.

Existe também a chance de dois pares modificarem simultaneamente a mesma parte do código, porém este risco é reduzido pela proximidade dos programadores e a intensa comunicação entre eles. Mesmo assim, se isto ocorrer, como as integrações são freqüentes, não será perdido muito tempo de trabalho.

- ***Design* contínuo**

XP defende que o código seja revisto e melhorado constantemente, refinando o *design* e a implementação sempre que esta possibilidade é detectada. Esta prática, em XP, é chamada de refatoração (*"refactoring"*), porém outros autores a denominam *design* contínuo, emergente ou evolucionário. A idéia é manter o código sempre (SHORE, 2004):

a) Único: não devem existir partes repetidas.
b) Explícito: o código demonstra claramente seu objetivo, geralmente sem a necessidade de comentários.
c) Simples: as soluções são específicas, não genéricas. O *design* não prevê extensões.
d) Coeso: código e conceitos relacionados são agrupados.
e) Desacoplado: código e conceitos não relacionados podem ser trocados sem interferência.
f) Isolado: código de terceiros é isolado através de uma interface única.

O *design* contínuo é possível graças ao código coletivo e pode ser executado com segurança devido à integração contínua e ao desenvolvimento orientado a testes. A programação em par completa o ciclo, assegurando a qualidade tanto do *design* quanto do código, que são revisados em tempo real.

Todas estas características, reunidas, fazem do XP um método muito bem definido, que permite desenvolvimento ágil, capaz de reagir rapidamente a mudanças, e ao mesmo tempo muito seguro, com diversas técnicas intrínsecas de qualidade (Seção 2.2.8). Por outro lado, XP é o método ágil que exige maior disciplina de seus seguidores, porque "perde seu sentido quando aplicado esporadicamente." (COCKBURN, 1999). Esta é sua maior fraqueza, já que "perda de consistência é um *modo de falha* comum dos humanos" (COCKBURN, 1999, grifo do autor). Assim, o sucesso da implementação de XP depende de liderança forte e apoio de toda a equipe, com intensidade maior que a necessária para outros métodos.

2.2.7.2 Scrum

Scrum é o nome de uma formação do *rugby* na qual oito jogadores de cada equipe se reúnem de maneira organizada para permitir o reinício rápido, seguro e imparcial do jogo, após uma infração menor ou de uma interrupção Os oito jogadores trabalham em conjunto para mover a bola pelo campo. No desenvolvimento de *software*, o que se espera é que a equipe trabalhe de maneira semelhante: após a definição da nova função a ser produzida, os programadores devem se organizar para atingir o objetivo no prazo estipulado.

Dos métodos ágeis, *Scrum* é o que proporciona maior liberdade aos programadores. Isto, porque "ele não define nenhuma técnica específica de desenvolvimento de *software* para a fase de implementação" (ABRAHAMSSON et al., 2002), mas apenas apresenta estratégias

de gerenciamento que permitem que a equipe trabalhe de uma forma que garante a flexibilidade do sistema em um ambiente que sofre constante modificação.

Cada iteração (Figura 2.8) é composta por três fases: pré-jogo, desenvolvimento e pós-jogo. O pré-jogo é uma etapa de planejamento, na qual ocorrem reuniões com clientes, equipe de vendas, marketing, suporte técnico, desenvolvedores e todas as outras pessoas que podem contribuir com o projeto. As informações reunidas, em especial as solicitadas pelo cliente, definem as características desejadas e formam um registro do produto. Nesta etapa também são selecionadas as características que devem ser implementadas na iteração atual, o prazo para sua execução (geralmente de uma a quatro semanas) e a arquitetura inicial do sistema.

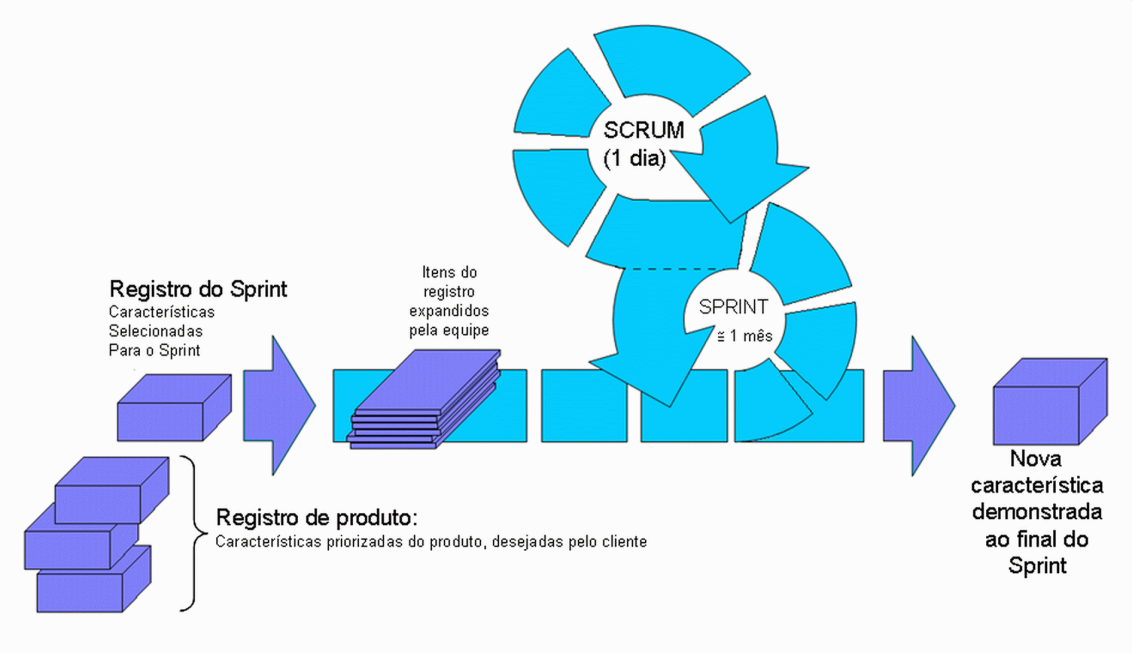


Figura 2.8: *Scrum* – Fluxo de processos
Fonte: Pressman (2005)

A partir daí, inicia-se a fase de desenvolvimento, chamada de *"sprint"*. Ela é dividida em *"scrums"*, com duração de um dia. A única atividade diária obrigatória é a reunião *scrum* (*"scrum meeting"*). Nela, todos os membros da equipe devem responder a três perguntas (RISING; JANOF, 2000):

- O que você realizou, relativo ao registro do projeto, desde a última reunião *scrum*?
- Quais as dificuldade encontradas na realização deste trabalho?
- O que você pretende realizar, em relação ao registro do projeto, até a próxima reunião *scrum*?

Esta reunião, com duração de quinze a trinta minutos, consegue focar os desenvolvedores nos objetivos do *sprint*, comunicar avanços e obstáculos, avaliar o progresso e reduzir riscos do projeto. Discussões e decisões mais complexas são deixadas para outras reuniões, que podem ser marcadas sempre que houver necessidade, para que a reunião *scrum* não se alongue excessivamente, reduzindo o tempo disponível para trabalho da equipe.

A equipe tem um gerente, chamado *"scrum master"*, que serve de ligação entre os desenvolvedores, o cliente e os demais departamentos da empresa durante o *sprint*. Sua principal função, entretanto, é prover à equipe de desenvolvimento todas as condições para que possa trabalhar livremente.

O *sprint* sempre termina no prazo agendado, mesmo que para isso algumas das características selecionadas para implementação tenham que ser ignoradas. Neste caso, o ritmo de desenvolvimento deve ser reavaliado para a próxima iteração. Em geral, após alguns *sprints*, os erros de estimação caem bastante.

Ao final do *sprint*, inicia-se a etapa pós-jogo, com a apresentação de uma versão funcional do *software* para avaliação do cliente. Uma reunião com todas as pessoas envolvidas no projeto avalia profundamente os resultados e ajusta as estimativas realizadas anteriormente. Nesta fase se decide também pela continuidade ou não do projeto. Caso ele prossiga, inicia-se um novo ciclo, repetindo todas as etapas.

Scrum foi criado para pequenas equipes, compostas por um máximo de dez pessoas. Em projetos maiores, divide-se o trabalho para várias equipes, sendo que cada uma é

independente das demais dentro dos *sprints*. Esta é a principal característica do *Scrum*: durante a etapa de desenvolvimento, a equipe é totalmente livre para definir seus métodos e conduzir seus trabalhos em busca do objetivo proposto. Qualquer mudança no projeto ocorre entre os *sprints*, nas reuniões do pré e do pós-jogo.

Além disso, outro grande diferencial do *Scrum* é que as estratégias de gerenciamento definidas permitem que grandes equipes trabalhem em pequenos grupos independentes, trabalhando em paralelo. Segundo Abrahamsson et al. (2002) esta vantagem tem sido em conjunto com XP para torná-lo viável em equipes maiores.

2.2.7.3 Crystal

Cockburn (2003) defende que os procedimentos devem adaptar-se às necessidades da equipe e não o contrário. Por isso, quando ele e Highsmith propuseram seu próprio método de trabalho, optaram por criar uma família de metodologias. Assim, ao iniciar um projeto, o chefe da equipe ou todos os membros em conjunto, devem avaliar a quantidade de desenvolvedores, o prazo disponível para entrega, as dificuldades técnicas, as implicações do produto, etc., definir um grau de risco e selecionar qual método *Crystal* é o mais adequado.

Como os cristais, cada método *Crystal* é caracterizado por uma cor e por sua "dureza". O método *Crystal Clear* é o mais simples, para projetos que oferecem menos risco. A seguir temos o *Crystal Yellow*, *Orange*, *Red*, etc. Cada um acrescentando mais "dureza", ou seja, rigidez, complexidade, às práticas de desenvolvimento. A Figura 2.9 mostra como o método mais adequado pode ser selecionado considerando o grau de risco e o número de pessoas na equipe de desenvolvimento. Outras características podem ser utilizadas na classificação, como obrigações legais ou produtividade.

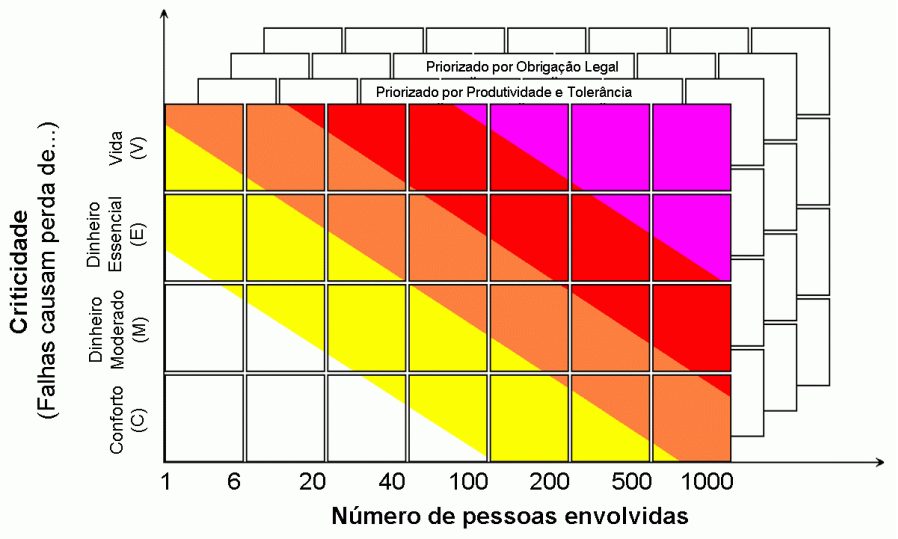


Figura 2.9: Esquema para seleção do método *Crystal* pela criticidade
Fonte: Cockburn (2003)

Todos os membros da família *Crystal* alinham-se com as idéias propostas pelo Manifesto Ágil e possuem um núcleo comum. Outras partes, processos, padrões, produtos de trabalho e práticas são diferentes para cada um deles (PRESSMAN, 2005).

Crystal ainda é recente e o número de adeptos é pequeno, mas mesmo assim tem recebido muita atenção pela inovação na apresentação de métodos mais ou menos complexos de acordo com o projeto a ser executado, e pelo respeito conquistado por seus criadores. O primeiro livro[10], contemplando o método mais "mole", *Clear*, foi lançado em 2004. *Crystal Orange*, *Orange Web* e *Red* já foram delineados, porém não existe ainda nenhuma publicação trazendo informações completas sobre eles.

[10] COCKBURN, A. **Crystal Clear:** A Human-Powered Methodology for Small Teams. Boston: Addison-Wesley, 2004. 336p.

2.2.8 Qualidade do *Software*

Pode parecer difícil garantir a qualidade de um *software* produzido através de métodos tão simplificados, com pouca documentação, tão dependentes de fatores humanos e nos quais nem mesmo os resultados desejados são totalmente conhecidos na fase inicial. Entretanto, como a qualidade é sempre um elemento fundamental no desenvolvimento de *software*, não faltam estudos sobre este assunto em metodologias ágeis.

Huo et al. (2004) apresentam um comparativo sobre as estratégias de teste utilizadas em cada uma das etapas da metodologia tradicional (Figura 2.10) e as aplicadas em métodos ágeis. Eles nos mostram que existem diferenças entre as duas propostas, porém pode-se atingir bom nível de confiabilidade em qualquer uma delas. Algumas das características dos métodos ágeis que substituem as estratégias de teste do modelo tradicional são apresentadas na seqüência.

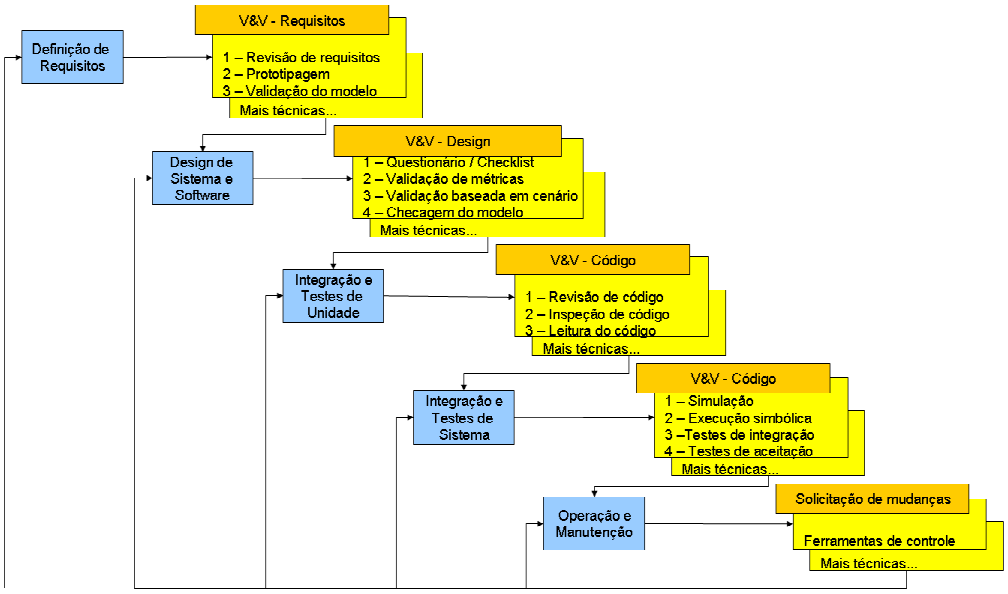


Figura 2.10: Modelo Cascata e suas técnicas de teste mais utilizadas
Observação: V&V – Verificação e Validação
Fonte: Huo et al. (2004)

- **Interação entre os Membros da Equipe**

O diálogo direto entre os programadores é um fator de qualidade importante em todas as etapas do ciclo de desenvolvimento de *software*. À primeira vista pode ser estranho aceitar esta condição, mas analisando as atividades exercidas, o valor da comunicação fica explicitado.

Sem bom diálogo, a equipe tem dificuldade em reunir as informações necessárias para a definição de requisitos, não consegue modelar, definir as melhores soluções e algoritmos, é incapaz de implementar o *design* contínuo e o código coletivo, tem problemas na etapa de integração e na avaliação e aplicação das idéias do cliente. Assim, a falta de diálogo é um problema grave em um ambiente de trabalho que tenta ser ágil.

Por outro lado, a sinergia de uma equipe que interage constantemente é capaz de influenciar fortemente na qualidade do trabalho, pois contribui tanto para um melhor planejamento das atividades, desde a definição de requisitos até a criação de algoritmos, quanto para a construção do *software* e as atividades de teste propriamente dito.

Em comparação com o modelo tradicional, os métodos ágeis são muito mais dependentes da boa comunicação entre a equipe de desenvolvimento, pois boa parte da documentação é substituída por contato direto entre as pessoas, porém os efeitos benéficos obtidos com implementação desta prática também são muito mais relevantes.

Apesar de ser muito difícil, ou talvez impossível, mensurar a quantidade de erros de especificação e implementação evitados somente pela comunicação, é certo

que o valor é significante. Assim, a interação tem que ser considerada também uma estratégia de qualidade.

- **Atuação do Cliente**

O cliente colaborativo (Seção 2.2.4), presente durante todo o processo de desenvolvimento, é importante na definição e ajuste dos requisitos de *software* e nos testes funcionais do sistema. Esta atuação constante é uma estratégia de correção sem precedentes nos métodos tradicionais, nos quais a consulta ao cliente só é realizada na definição dos requisitos e, algumas vezes, durante o *design* do sistema, porém sua contribuição é limitada pelos problemas de comunicação e compreensão, apresentados nas Seções 2.2.3 e 2.2.4.

- **Programação em Par**

A programação em par proporciona grande redução no número de erros tanto de *design* quanto de codificação, pois enquanto alguém cria o código, outro programador está avaliando a lógica e a implementação. Esta inspeção contínua tem efeito similar a todas as técnicas de Validação e Verificação (V&V) das etapas de *design* e codificação dos métodos tradicionais, mas com a vantagem de ocorrer continuamente, detectando e corrigindo falhas em tempo real.

- **Integração Contínua**

No desenvolvimento ágil, a integração do sistema é realizada repetidamente, em curtos espaços de tempo. Promover repetidas integrações significa também testar a integração do sistema diversas vezes, lidando com poucas modificações em cada uma delas. Assim, é muito mais fácil detectar falhas e corrigi-las. Este procedimento

incremental, baseado na premissa de sempre manter o *software* funcionando (Seção 2.2.3), é muito mais efetivo e previsível que os testes de integração dos métodos tradicionais.

- **Testes de Aceitação**

  Enquanto no modelo Cascata, os testes de aceitação ocorrem somente ao final do desenvolvimento, métodos ágeis propõem sua execução freqüente e em curtos espaços de tempo, sempre que uma versão é disponibilizada (Seção 2.2.3). Isto permite detectar erros mais cedo e corrigi-los sem grande prejuízo. O cliente pode solicitar mudanças constantemente, dirigindo o desenvolvimento do sistema (Seções 2.2.4 e 2.2.6), ao invés de esperar até a entrega da versão final.

- ***Design* Contínuo**

  O *design* contínuo (apresentado na Seção 2.2.7.1) não é uma estratégia de teste, mas é uma técnica de garantia de qualidade importante. Se bem conduzido, traz confiabilidade e reduz riscos associados às modificações e à manutenção do sistema, além de explicitar os possíveis erros existentes, já que o código otimizado é muito mais inteligível.

- **Código Coletivo**

  Assim como o *design* contínuo, código coletivo também não é uma estratégia de testes propriamente dita, mas uma técnica de garantia de qualidade, já que o código otimizado por um grupo de programadores é mais confiável que aquele criado e mantido por um único programador.

- **Desenvolvimento Dirigido a Testes**

  Esta estratégia, se bem conduzida, garante as entradas e saídas de todas as funções do *software*. É um método agressivo de testes que produz os mesmos resultados que a avaliação caixa preta de todas as funções de um programa desenvolvido pelo método tradicional.

Apesar das críticas e dúvidas, o que se comprova, mais uma vez, é que a metodologia não é o ponto fundamental, mas sim a aplicação que se dá das práticas propostas. Teoricamente, se todas as estratégias apresentadas anteriormente forem implementadas corretamente, o resultado será um *software* tão ou mais confiável que os desenvolvidos pelos métodos tradicionais, como pode ser observado na Tabela 2.4.

Tabela 2.4 – Práticas ágeis de qualidade de *software*

Etapa	Prática Ágil de Qualidade	Equivalência no Modelo Tradicional
Definição de requisitos	Interação da equipe Interação com o cliente *Design* contínuo	Revisão de requisitos Prototipagem Validação do modelo
Modelagem/Análise/*Design*	Interação da equipe Interação com o cliente Programação em par *Design* contínuo Código coletivo	Questionário/Checklist Validação de métricas Validação baseada em cenário Revisão do modelo
Codificação e testes	Interação da equipe Programação em par *Design* contínuo Código coletivo Desenvolvimento dirigido a testes	Revisão de código Inspeção de código Leitura do código
Integração e testes	Interação da equipe Interação com o cliente Programação em par Integração contínua Código coletivo Desenvolvimento dirigido a testes	Simulação Execução simbólica Testes de integração Testes de aceitação
Distribuição	Interação da equipe Interação com o cliente *Design* contínuo Código coletivo	Ferramentas de controle

Mesmo ignorando algumas das técnicas, como a programação em par e o desenvolvimento dirigido a testes, o produto ainda terá bom nível de confiabilidade se as demais forem aplicadas com correção. O ponto chave, não só em termos de qualidade, mas de eficiência em todos os aspectos, é definir o método mais adequado, considerando o tipo de produto, o tamanho da equipe de desenvolvimento, o prazo disponível, o cliente, os riscos envolvidos, etc. (DEMARCO; BOEHM, 2002; BECK; BOEHM, 2003).

3 Metodologia Aplicada

A implementação do processo de *software* para teste de equipamentos aeroespaciais proposto nesta dissertação foi executada na empresa Opto Eletrônica, sendo a mesma uma instituição genuinamente brasileira que desenvolve, fabrica e comercializa equipamentos de alta tecnologia desde 1985, participando desde a idéia do produto até a assistência técnica. Atua em diversas áreas, como médico-hospitalar, industrial, defesa e aeroespacial. A maioria de seus produtos demanda o desenvolvimento de *software* embarcado e alguns necessitam de interfaces gráficas que rodam em microcomputadores padrão.

Durante estes anos, a empresa vem crescendo sistematicamente e a complexidade de seus produtos segue a mesma direção. A única maneira de assegurar a qualidade e controlar os riscos associados aos projetos é definir métodos de desenvolvimento, usando o conhecimento geral e a experiência adquirida.

Os primeiros produtos não necessitavam de *softwares* complexos e apenas um ou dois programadores, livres para aplicar suas próprias estratégias, eram responsáveis por seu desenvolvimento. A qualidade do *software* era garantida por análise de riscos e pela execução de um conjunto de testes de aceitação.

Conforme os projetos passaram a incorporar maior capacidade de processamento os sistemas lógicos cresceram em tamanho, preço, relevância e risco, o que passou a exigir estratégias mais complexas. Os engenheiros de *software*, que sempre foram os responsáveis pela metodologia, definiram um conjunto de práticas, gerando não um método completo, mas regras gerais que devem guiar o desenvolvimento de qualquer produto.

Estas regras têm muito em comum com os métodos ágeis. Provavelmente isto ocorreu devido às características da maioria dos projetos da Opto Eletrônica, onde geralmente as equipes de desenvolvimento de *software* são compostas por poucas pessoas, de duas a seis, que dividem o tempo criando também o *hardware* em conjunto com outros engenheiros, e os produtos são voltados ao mercado. Portanto, os requisitos variam constantemente em resposta a desejos do consumidor, mudanças tecnológicas, equipamentos concorrentes, etc.

As características do método geral e as diferenças relativas ao produto estudado neste trabalho serão apresentadas detalhadamente na Seção 3.2. A seguir, serão descritas algumas características do equipamento de testes estudado, que influenciaram em decisões importantes tomadas ao longo de seu desenvolvimento.

3.1 Características do MUX-GSE

Como explicado na Seção 1.1.3, o MUX-GSE foi especificado como um conjunto de equipamentos cuja finalidade é realizar uma série de testes que auxiliam na montagem e permitem verificações de requisitos da Câmera MUX dos satélites CBERS-3 e 4. As principais funções previstas para o MUX-GSE foram apresentadas a partir da página 45.

Foi solicitada a produção de três unidades do MUX-GSE, de dois modelos diferentes: simplificado (SGSE, duas unidades) e completo (CGSE, somente um). O CGSE possui banco óptico completo, que auxilia na fabricação da MUX e permite sua calibração e a verificação de todos os seus requisitos. Os SGSE servem para assegurar que os parâmetros mais importantes não sofreram alterações durante o transporte e a integração ao satélite, permitindo uma verificação final que qualifica a MUX para o lançamento.

Cada MUX-GSE é composto por dois *racks* (Figura 3.1) padrão 19'', com 14U[11] de altura. Um deles é o Controlador GSE e o outro o Sistema de Exibição de Imagens (SEI). Cada *rack* contém um microcomputador industrial, que suporta a conexão de até 10 dispositivos PCI[12], gaveta de alimentação (responsável pela distribuição elétrica) monitor, teclado e mouse do tipo "*track ball"*. O Controlador GSE traz ainda freqüencímetro, fonte de alimentação controlada e uma gaveta de interfaces para conexão com a MUX. O SEI possui impressora laser, gaveta de alimentação e gaveta de interfaces, que recebe os dados digitais das imagens obtidas pela MUX.

Figura 3.1: *Rack* Controlador GSE (direita) e Sistema de Exibição de Imagens (esquerda)
Fonte: Opto Eletrônica

Os *softwares* do MUX-GSE têm o objetivo de automatizar os testes previstos, sem impedir a operação manual dos equipamentos de medição, para a execução de procedimentos não programados inicialmente. Além disso, devem executar auto-testes antes da conexão com a MUX, visando garantir que a câmera não seja danificada em caso de defeitos nos

[11] U (unidade de rack) é a unidade padrão de altura de equipamentos em racks de 19''. 1U é equivalente a 44mm.
[12] PCI (*"Peripheral Component Interface"*) é um duto de dados padrão para conexão de dispositivos periféricos à placa mãe de um computador. É a interface mais utilizada nos micros modernos, que geralmente disponibilizam de duas a quatro portas PCI.

equipamentos. Devem ainda permitir gravação e impressão dos resultados dos testes e manter um arquivo de log[13] com todas as ações executadas pelo MUX-GSE, medições realizadas, erros de processamento, etc.

Para permitir máxima automatização dos procedimentos de teste, a equipe de desenvolvimento do MUX-GSE optou por utilizar instrumentos virtuais de medida (Figura 3.2), ou seja, placas de expansão PCI contendo *hardware* capaz de adquirir, digitalizar e transferir para o PC os mesmos sinais obtidos de instrumentos de medida reais. Com isso, é possível criar uma interface gráfica que representa os equipamentos de bancada, permitindo todos os comandos padrão e a programação de rotinas específicas, que não são comuns nos instrumentos reais.

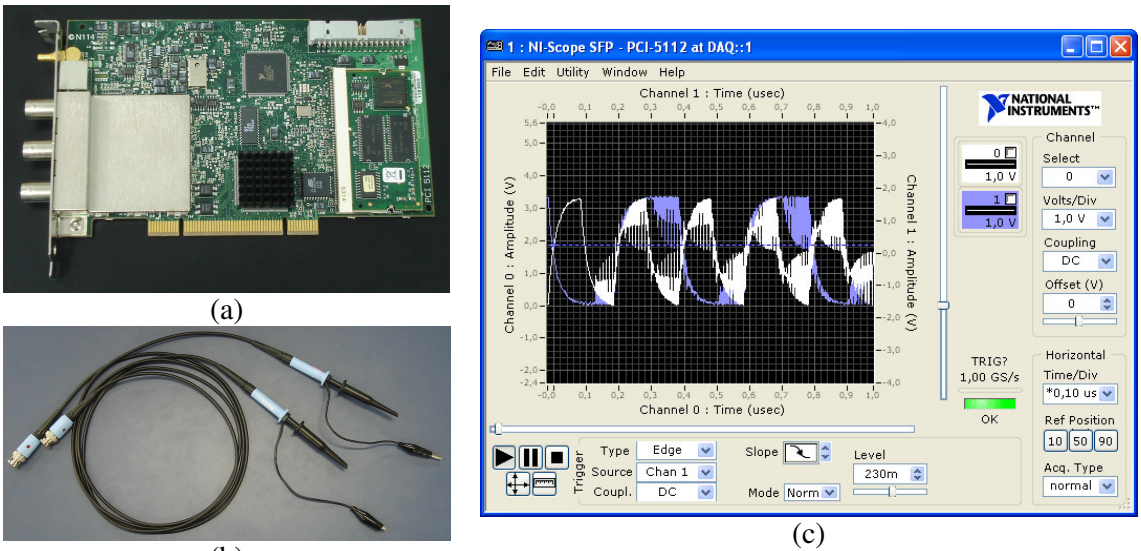


Figura 3.2: Osciloscópio virtual – (a) Placa PCI; (b) Pontas de prova; (c) Interface gráfica.
Fonte: ***National Instruments***

As interfaces entre o MUX-GSE e a MUX também foram implementadas através de placas PCI de entrada/saída (I/O) digitais e analógicas. Foram desenvolvidos circuitos de condicionamento para adequar os sinais fornecidos/recebidos por estas placas aos utilizados

[13] Log é o armazenamento seqüencial de informações, de modo que se possa manter um histórico de eventos relevantes.

na câmera para transmissão de telemetrias, recepção de telecomandos e comunicações de controle, além dos sinais de teste. Os circuitos foram construídos utilizando um barramento digital de endereçamento e uma placa de circuito impresso para cada conexão com a MUX, contendo diversos multiplexadores digitais e analógicos. A Figura 3.3 representa todas as conexões, internas e externas, do MUX-GSE.

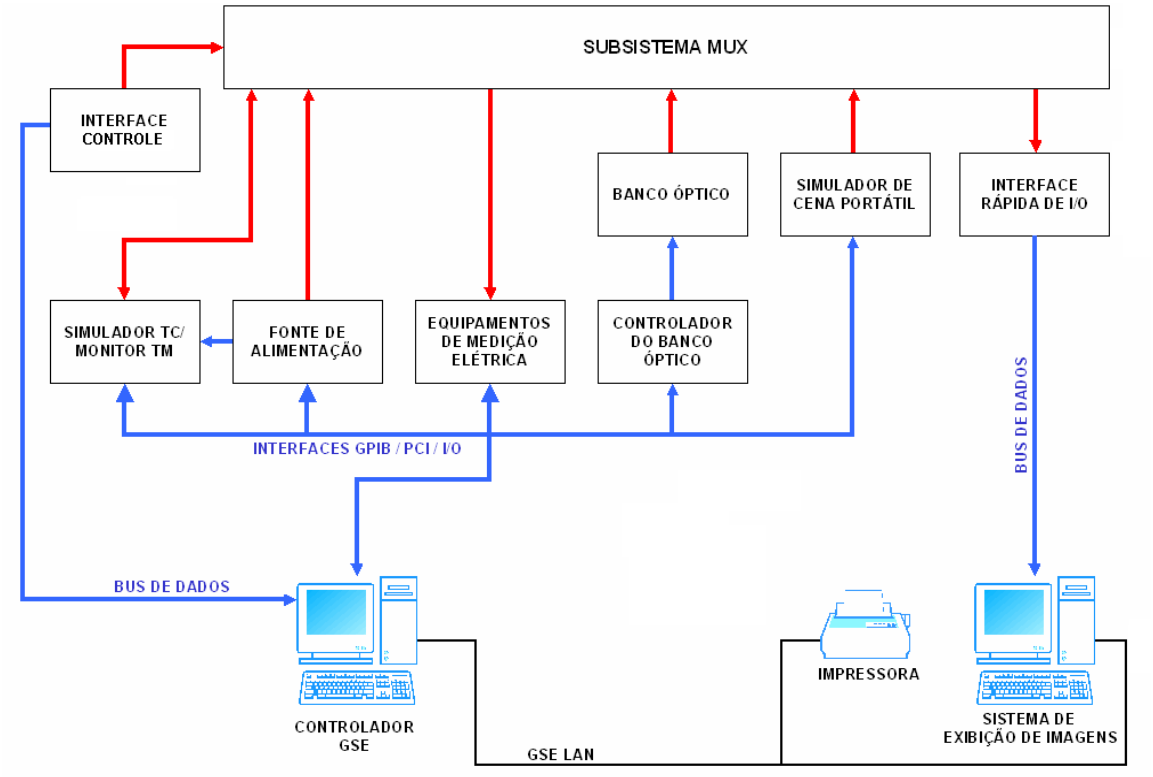


Figura 3.3: Diagrama de blocos do MUX-GSE
TC – Telecomandos; TM – Telemetrias; I/O – Entrada/Saída.
Fonte: Opto Eletrônica

Esta estratégia facilita a captura de dados para os testes automáticos e a realização de processamento e exibição de resultados em tempo real. Incorpora, também, maior flexibilidade ao sistema, pois é possível realizar muitas modificações em *software*, sem alterar o *hardware* utilizado. Alterações mais complexas podem exigir mudanças nas placas de interface, porém sua construção modular, facilita possíveis adequações a novos requisitos.

Os instrumentos virtuais instalados no Controlador GSE são: multímetro digital, osciloscópio, gerador de formas de onda digitais e analisador lógico, além da fonte de alimentação e do freqüencímetro, que são instrumentos de bancada com controle GPIB[14], o que permitiu a criação de interface gráfica e rotinas de controle via *software*. O fabricante das placas fornece *software* de boa qualidade para emulação do multímetro e do osciloscópio, porém os painéis para os demais equipamentos tiveram que ser desenvolvidos.

Os *racks* comunicam-se através de LAN (*"Local Area Network"*) com taxa de transmissão máxima de 1 Gbit/s e o Controlador GSE disponibiliza portas LAN para conexão via cabo e possibilita ainda acesso sem fio para micros executando um aplicativo cliente (parte dos *softwares* do MUX-GSE) que permite acesso aos resultados dos testes e informações armazenadas.

Com isso, foram definidos três *softwares* para microcomputadores diferentes: aplicativo do controlador GSE, do SEI e aplicativo cliente. Os três compõem a plataforma de *softwares* do MUX-GSE, juntamente com sistema operacional, pacote de *softwares* utilitários, anti-vírus e plataforma de desenvolvimento.

O aplicativo do Controlador GSE foi planejado como o responsável pelo controle de todos os instrumentos de medida virtuais e de bancada e suas respectivas interfaces gráficas, controle manual e automático do banco óptico, execução de todas as rotinas de testes com utilização também dos dados provenientes do SEI, gravação e impressão dos resultados, gravação de arquivo de log e transmissão de informações em tempo real, via LAN, para o aplicativo cliente.

Para o SEI, foi planejado o desenvolvimento de um aplicativo capaz de receber e exibir as imagens da MUX em tempo real, armazená-las e transmiti-las ao Controlador GSE.

[14] GPIB (*"General Purpose Interface Bus"*) é uma especificação de duto de comunicação padrão definida na norma IEEE-488, que foi originalmente criada para uso com equipamentos de teste automáticos.

Ele também permite a conexão do aplicativo cliente, para o qual transmite as informações exibidas na tela em tempo real.

O aplicativo cliente foi definido como capaz de conectar-se via LAN ao Controlador GSE ou ao SEI e exibir as telas de testes em execução, as imagens da câmera em tempo real, abrir informações de testes realizados e imagens anteriores mostrando-os com a formatação adequada. Entretanto, ele não foi projetado para enviar comandos a nenhuma máquina.

A maioria destas especificações genéricas iniciais foi derivada dos requisitos do sistema, que não traziam informações detalhadas referentes ao *software*, ou seja, os requisitos de *software* não eram muito bem definidos previamente, o que pesou fortemente na seleção do método de desenvolvimento, como mostrado na Seção 4.1.1.

3.2 Método de Desenvolvimento

O desenvolvimento de *software* na Opto Eletrônica é conduzido por engenheiros eletricistas (eletrônicos), que também são responsáveis pelos sistemas digitais produzidos. As regras gerais seguidas para criação do *software* foram definidas por estes engenheiros, em conjunto com o Departamento de Gestão e Qualidade. A utilização da experiência prática na sistematização do processo foi fundamental para garantir a aplicabilidade do método proposto. Além de não haver resistência em sua implantação, tinha-se certeza de seu funcionamento e da sua adequação aos produtos que seriam gerados.

Por coincidência, ou pelas características dos projetos executados (Tabela 3.1) e a influência direta dos desenvolvedores, as práticas adotadas na Opto Eletrônica têm muito em comum com as propostas do Manifesto Ágil (Tabela 3.2). Para evidenciar este fato, elas serão descritas a seguir, na mesma seqüência utilizada para apresentação do Desenvolvimento Ágil na Seção 2.2, comparando aquelas idéias com as utilizadas nos projetos de equipamentos

médicos, industriais e, principalmente, no equipamento de testes estudado neste trabalho. Os pontos fracos da metodologia aplicada também serão apontados; assim, este trabalho poderá servir como ponto de partida para discussão de modificações, tendo o intuito de fortalecer e aprimorar o método adotado.

Tabela 3.1 – Características dos projetos da Opto Eletrônica

Características dos Projetos da Opto Eletrônica que Justificam a Aplicação de Método Ágil
Equipes pequenas
Desenvolvedores dividem tempo com outras tarefas
Requisitos mudam constantemente
Projetos multidisciplinares
Produtos voltados ao mercado

Tabela 3.2 – Práticas gerais de desenvolvimento – Identificação da proposta ágil correspondente e comparação com os métodos *Scrum* e XP

Proposta Ágil	**Prática de Desenvolvimento**	**Práticas Similares**	
		Scrum	**XP**
Valorização dos indivíduos	Autonomia aos desenvolvedores	Ambos propõem estas práticas	
	Aprendizado e ensino constantes		
Interação da equipe	Equipe dividindo a mesma sala	Ambos recomendam	
	Reuniões solicitadas por membros da equipe a qualquer momento	Reunião *Scrum* (diária)	*"Stand-up meeting"* (diária)
Software funcionando	Distribuições freqüentes	Nova funcionalidade ao final de cada *Sprint*	Distribuições freqüentes
	Integração contínua	-[*]	Integração contínua
Colaboração do cliente	Contato freqüente com o cliente, mas sem período definido (direto ou através de pesquisas)	Reuniões com o cliente ao final de cada *Sprint* (o término do *Sprint* é pré-definido)	Reuniões freqüentes com o cliente, com período definido
Documentação leve	Cadernos de projeto e documentos adicionais[**]	Registro de produto e documentos adicionais[**]	Cartões de estórias[***] e documentos adicionais[**]
	Design contínuo	-[*]	*Design* contínuo
	Código coletivo	-[*]	Código coletivo
Responder a mudanças	Iterações curtas e revisão de requisitos a cada ciclo	Revisão do registro de produto a cada *Sprint*	Iterações curtas e revisão de requisitos a cada ciclo

[*] *Scrum* não define práticas para a fase de implementação

[**] Podemos citar, como documentos adicionais: manual do usuário, resultados de testes, informações de acompanhamento do projeto, etc.

[***] Em XP, cada estória corresponde a uma funcionalidade do *software* requisitada pelo cliente e é registrada em um cartão que ajuda a controlar o andamento do projeto.

3.2.1 Indivíduos

A valorização dos indivíduos, considerados elementos de primeira ordem de importância no método proposto, é evidenciada por aspectos ligados ao aprendizado e ensino contínuos, o bem-estar dos funcionários e a autonomia e responsabilidade sobre o próprio trabalho.

Como ferramentas de aprendizado, pode-se citar o acesso à internet e a periódicos científicos, o estímulo à realização de cursos de capacitação e de pós-graduação e, ainda, a possibilidade de participação em congressos e eventos científicos, com a apresentação de resultados obtidos nos trabalhos executados.

Para manter a satisfação dos funcionários, entre outras estratégias, é implementada a flexibilidade de horário. Esta prática, entretanto, produz um efeito colateral: como não existe limite para horas-extras, os prazos apertados podem levar a um ritmo excessivo de trabalho, o que reduz a qualidade dos produtos, aumenta os riscos dos projetos e provoca quedas de produtividade (DEMARCO; LISTER, 1990; BECK, 1999a).

Finalmente, o maior aspecto de valorização dos desenvolvedores é a liberdade que desfrutam na condução de seus trabalhos técnicos, com possibilidade de pesquisar, discutir e implementar soluções em todas as etapas do trabalho: análise, *design*, codificação, testes e distribuição.

3.2.2 Interações

Conforme constatado na grande maioria das situações, a comunicação entre os funcionários de P&D da Opto Eletrônica é muito estimulada. Não só os responsáveis pelo *software* dividem a mesma sala, mas todos os pesquisadores (Figura 3.4), uma vez que os produtos desenvolvidos geralmente integram sistemas mecânicos, eletro-eletrônicos, ópticos e lógicos, dependentes uns dos outros. Pessoas que atuam na mesma área têm as mesas mais próximas, porém todos estão disponíveis a qualquer momento. Isto estimula a troca de idéias e problemas, discussão de detalhes de integração e promove a descoberta de melhores soluções de um ponto de vista multidisciplinar.

O aspecto negativo observado desta disposição é o barulho, já que a discussão de um grupo pode atrapalhar a concentração dos demais pesquisadores. Mesmo assim, como estas interrupções não são freqüentes, a manutenção de um bom nível de interação compensa este prejuízo.

Figura 3.4: Vista da sala de projetos de Pesquisa e Desenvolvimento da Opto Eletrônica

O diretor de P&D coordena diretamente todos os projetos. Ele promove uma reunião semanal, exceto em casos excepcionais, com todos os pesquisadores, na qual o progresso do projeto é avaliado, os próximos passos são definidos e os principais problemas discutidos em nível de sistema. Os desenvolvedores de *software* podem apresentar suas necessidades, corrigir requisitos e prazos. Quando necessário, podem solicitar uma reunião específica.

O líder de projeto de *software* é um membro da equipe, geralmente o programador mais experiente, ou que melhor conhece o produto em desenvolvimento. Sua responsabilidade é manter o rumo do projeto, avaliando prazos, necessidades e qualidade. Os desenvolvedores podem organizar reuniões sempre que considerarem oportuno. Isto ocorre com maior freqüência em etapas de planejamento e menos durante a codificação, porém, como todos estão sempre muito próximos, o fluxo de informações nunca cessa.

Os consumidores não estão sempre disponíveis, como recomendado na maioria dos métodos ágeis. Os departamentos de Marketing e Vendas, entretanto, provêem informações sobre os produtos em desenvolvimento. Pesquisas de mercado, visitas de clientes e avaliações de protótipos ocorrem com freqüência para fornecer os dados necessários. Sempre que uma versão de *software* é distribuída, o pessoal de marketing avalia suas características e, se possível, alguns clientes são selecionados para testar e opinar sobre o equipamento. Quando existe interface com o usuário, ela é criada nas primeiras etapas do projeto e as idéias dos consumidores são incorporadas o mais cedo possível, porque mudar os comandos do sistema geralmente implica em modificações profundas de *hardware* e *software*, que são caras, mesmo para projetos ágeis.

3.2.3 *Software* Funcionando

Os projetos de *software* executados na Opto Eletrônica sempre objetivam a produção de uma versão utilizável em curto espaço de tempo, ainda que ela tenha capacidade limitada. As vantagens desta abordagem, apresentadas na Seção 2.2.3, incluem o rápido retorno de investimento ao cliente e a precisão das informações realimentadas à equipe de desenvolvimento.

Observa-se que, para a maioria dos produtos da empresa, existe mais um fator de peso na aplicação desta estratégia: devido à forte interação entre diferentes áreas do conhecimento, a possibilidade de se executar algumas funções do *software* serve de suporte ao desenvolvimento de outros subsistemas. Muitas vezes, é impossível testar certas operações sem que exista uma base de *software* disponível, que pode ser construída como protótipo ou nos ciclos normais de desenvolvimento, dependendo do caso.

A realimentação de usuários finais tem valor também para outras partes do sistema e permite aperfeiçoá-las, apesar da maior dificuldade em se modificar componentes mecânicos ou ópticos em comparação com o *software*. Ignorar as opiniões no desenvolvimento de equipamentos comerciais, entretanto, seria deixar de levar em conta desejos e necessidades de um grande número de possíveis consumidores, o que não é a estratégia mais adequada a um ambiente competitivo.

Em produtos exclusivos, feitos para um usuário específico, como é o caso do MUX-GSE, a opinião do cliente e/ou usuário final em relação a todos os componentes é ainda mais relevante, pois está intimamente ligada à sua satisfação, fator determinante do sucesso do projeto. Assim, todas as suas propostas são consideradas e implementadas sempre que viáveis. Para que isto seja possível, é fundamental o teste de versões funcionais.

3.2.4 Colaboração do Cliente

Sem o cliente colaborativo, a existência de versões utilizáveis perde muito de sua eficácia. A realimentação fica comprometida e o projeto fica dependente da sensibilidade dos desenvolvedores em relação ao produto a ser criado. Cresce a probabilidade de erros graves passarem despercebidos durante muito tempo, o que põe em risco o cumprimento das metas do projeto.

Conforme constatado, considerando a maioria dos projetos de P&D da Opto Eletrônica o cliente é a própria direção da empresa, enquanto que os usuários são potenciais compradores dos equipamentos desenvolvidos, ou seja, médicos, dentistas, engenheiros, etc., dependendo do tipo de produto. Alguns destes possíveis futuros usuários são procurados desde o início do projeto, para que possam expressar suas idéias e fornecer as opiniões que direcionam os trabalhos.

O MUX-GSE, entretanto, é um equipamento desenvolvido para um cliente externo, o Governo Brasileiro, representado pelo INPE, e cujo principal usuário é interno: a equipe de desenvolvimento da MUX. Esta diferença intensifica o relacionamento com o usuário final, porém prejudica os contatos com o cliente.

O usuário final está disponível em tempo integral, para comunicação direta. Isto tem aspectos muito positivos, pois a realimentação ocorre em tempo real. Existe também um lado negativo, pois nem sempre os desenvolvedores têm a privacidade necessária para conduzir seus trabalhos, porém os benefícios alcançados são muito mais relevantes.

Ter um órgão governamental como cliente, entretanto, traz certa burocracia ao projeto, devido ao volume de documentos exigidos, como será mostrado na próxima seção. Por outro lado, os pesquisadores do INPE designados para acompanhamento das atividades são

extremamente qualificados e conhecem as dificuldades inerentes a um projeto tão complexo. Além disso, estão tão interessados no sucesso do produto quanto os desenvolvedores, o que os torna grandes aliados.

O INPE fornece as informações que dispõe sobre o projeto, facilitando um pouco o trabalho a ser realizado. Infelizmente não existem muitos dados, uma vez que o Brasil nunca construiu uma câmera similar à MUX e seus equipamentos de suporte. A China, responsável pela produção das câmeras anteriores, não disponibilizou informações técnicas que pudessem auxiliar no desenvolvimento.

Apesar destas dificuldades, o INPE atua como parceiro, exigindo resultados positivos e fornecendo o suporte necessário para sua obtenção. Os responsáveis pela observação do MUX-GSE estão sempre disponíveis para comunicação via e-mail ou telefone e reuniões periódicas são utilizadas para trocar informações sobre o andamento do projeto, aprovar o que já foi produzido e definir os próximos passos.

3.2.5 Documentação Leve

Não é fácil reduzir o volume de documentos produzidos em um projeto cujo cliente é um órgão governamental. Mais difícil ainda quando o produto é um equipamento aeroespacial, que exige altíssimo nível de confiabilidade. Mesmo assim, é preciso convencer o cliente da importância desta abordagem.

Por mais que os pesquisadores do INPE sejam colaborativos, uma série de relatórios, documentos de planejamento e especificações é considerada obrigatória e não pode ser relevada durante o desenvolvimento dos *softwares* do MUX-GSE. Ficou então acordada a produção de um número mínimo de documentos capazes de satisfazer as exigências do projeto e sua revisão somente em datas (marcos) pré-definidas. A intenção foi reduzir o

esforço gasto na produção de documentos, aumentando o tempo empregado na geração do produto propriamente dito.

Mesmo com esta abordagem, uma parte considerável do tempo de trabalho é consumida na geração da documentação obrigatória. A estratégia utilizada é concentrar esforços no trabalho de desenvolvimento e escrever os relatórios o mais próximo possível das datas de entrega, evitando a execução de revisões intermediárias. O andamento dos trabalhos é registrado em cadernos de projeto mantidos por cada funcionário, seguindo normas previstas no sistema de gerenciamento de qualidade da Opto Eletrônica.

O código-fonte é considerado o principal documento do *software* e, para isso, é sempre construído de maneira padronizada, seguindo convenções pré-definidas, e modular, com funções específicas, curtas e objetivas. A redação de manuais de uso e manutenção foi agendada para a data de entrega oficial do MUX-GSE, aliviando bastante as dificuldades de implementação de modificações de uma versão para outra do *software*.

3.2.6 Responder a Mudanças

O processo de *software* adotado na Opto Eletrônica possui diversas características ágeis, que contribuem com a capacidade de reação a mudanças nos requisitos, soluções, tecnologia, etc. Como explicado na Seção 2.2.6, a agilidade do método deriva do conjunto destas características mais do que da ação de cada uma delas isoladamente.

A execução de iterações curtas, compreendendo etapas de planejamento, implementação e testes é a base que permite flexibilidade e atuação rápida sempre que mudanças ocorrem. A Figura 3.5 traz um fluxograma simplificado que resume as atividades de desenvolvimento, evidenciando a ocorrência de iterações no processo.

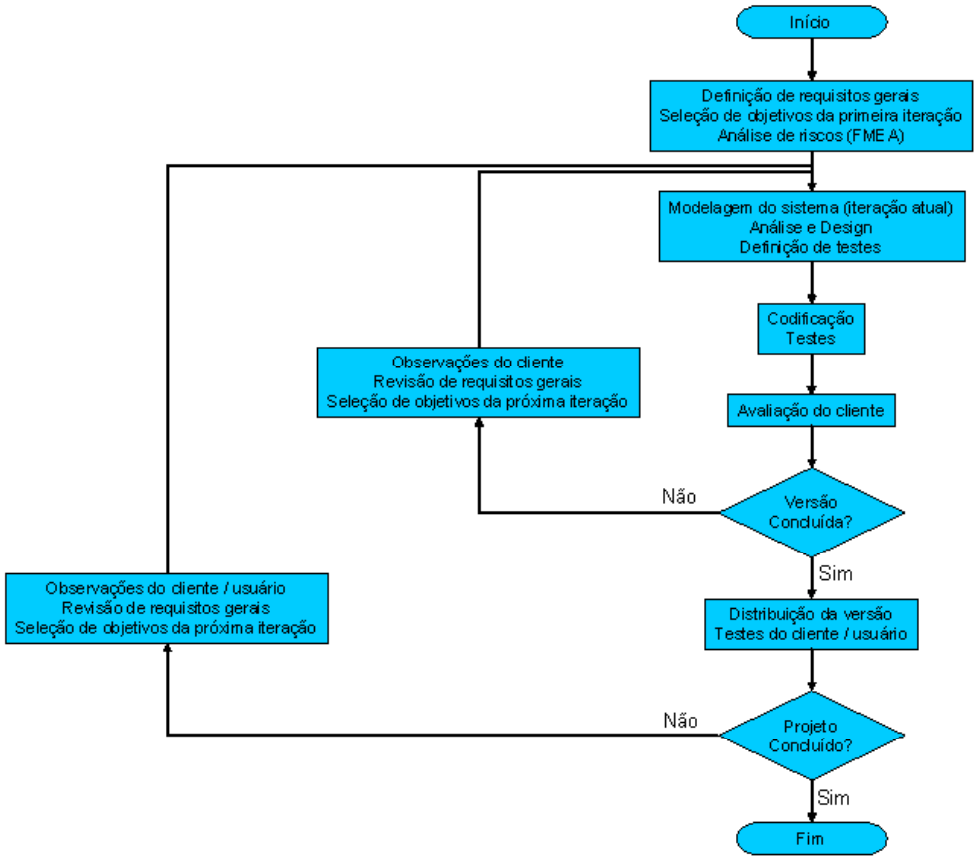


Figura 3.5: Fluxograma do processo de desenvolvimento de *software* da Opto Eletrônica

O modelo pode ser considerado uma adaptação da Espiral de Boehm (Seção 2.1.2.2), com iterações executadas em curtos espaços de tempo (de uma semana a um mês). Assim como em *Scrum* (Seção 2.2.7.2), não existe método de desenvolvimento pré-definido. Cabe aos desenvolvedores determinarem as técnicas mais adequadas ao trabalho e com as quais tenham familiaridade.

Esta liberdade é possível graças ao código coletivo, a integração e *design* contínuos, práticas que foram apresentadas na Seção 2.2.7 e cuja implementação na Opto Eletrônica será explicada na próxima seção. É sustentada ainda pela boa rede de interações (Seção 3.2.2) e pela confiança nos membros da equipe (Seção 3.2.1).

A padronização e coletividade do código e a utilização de cadernos de projeto (Seção 3.2.5) garantem a rastreabilidade das informações relevantes, sem comprometer a agilidade do método. Apesar de toda a flexibilidade obtida, a qualidade do *software* é mantida, como será mostrado na Seção 3.2.8.

A discussão de contratos não é um problema grave em grande parte dos projetos conduzidos pela Opto Eletrônica, já que o cliente é interno. Esta situação não vale para o caso estudado neste trabalho, porém a colaboração do INPE (Seção 3.2.4) permitiu a aplicação do método proposto sem entraves graves.

3.2.7 Outras Características

Algumas características específicas de um ou outro método ágil são adotadas na Opto Eletrônica. Elas contribuem com a qualidade do *software*, a agilidade na implementação e a interação entre as práticas propostas. Sua implementação ocorre como segue:

- **Integração contínua**

  Esta prática é adotada como forma de minimizar as chances de dois programadores realizarem modificações na mesma parte do código, situação que provoca a perda do trabalho de um dos dois. Além disso, toda integração é acompanhada de um conjunto de testes, que assegura o funcionamento global do programa, aumentando a qualidade do produto.

  O código-fonte completo é mantido em um local pré-definido e nunca é modificado diretamente. Cada desenvolvedor o copia para sua máquina de trabalho para que possa realizar as mudanças desejadas. Depois de concluí-las, faz uma cópia de segurança do código original, integra as alterações, testa o funcionamento do

programa e coloca o código integrado no local apropriado, para que todos possam acessá-lo.

Esta operação é feita em uma máquina reservada para este fim ou, quando a equipe é pequena, na própria máquina de trabalho do desenvolvedor, contudo, neste caso, ele precisa avisar todos os outros membros da equipe antes da integração. Para os *softwares* do MUX-GSE, com a equipe composta por apenas quatro pessoas, não foi utilizado microcomputador exclusivo para integração, porém foi definida uma estratégia de armazenamento dos arquivos e a comunicação direta foi utilizada para minimizar ainda mais as chances de modificações simultâneas em uma mesma parte do código.

Considerando ainda tal método, os testes executados na integração não compreendem todo o *software*, mas apenas as áreas afetadas pelas modificações recentes e os pontos mais críticos do sistema. Esta estratégia não é a mais segura e depende fortemente da perícia do desenvolvedor em detectar as falhas provocadas direta e indiretamente pelas mudanças efetuadas, mas, com os testes pré-definidos pela análise de risco e uma equipe tecnicamente bem preparada, é possível manter boa qualidade com grande velocidade nos procedimentos de integração.

- **Código coletivo**

Esta prática é necessária para que o *design* contínuo possa ser implementado. É importante também para manter a homogeneidade do código, tanto em termos de qualidade quanto de padronização, pois os programadores obrigam-se uns aos outros a manterem bom nível de qualidade.

Nos projetos implementados na Opto Eletrônica todos os desenvolvedores têm liberdade para realizar modificações em qualquer parte do código, sem pedir

autorização à pessoa que originalmente o construiu. Entretanto, é de praxe conversar com o programador e discutir as vantagens, desvantagens e implicações das mudanças que pretende implementar. Isto ajuda a disseminar o conhecimento e a detectar defeitos que as alterações podem introduzir.

- **Programação em par**

Conforme tem sido observado, esta não é uma prática adotada oficialmente, pois ainda não há consenso sobre sua validade na grande maioria dos casos. Entretanto, todos os desenvolvedores acreditam no seu potencial de reduzir o número de erros de *design* e implementação. Assim, rotinas mais complexas, consideradas críticas na análise de riscos ou que exijam otimização extrema, são selecionadas e programadas em par.

O objetivo desta estratégia é evitar a redução na produtividade imediata, que ocorre quando dois programadores trabalham juntos, mas mantendo bom nível de confiabilidade do *software* através da observação mais cuidadosa das partes que podem trazer maiores problemas.

- ***Design* contínuo**

A revisão constante do código-fonte, buscando mantê-lo simples, claro e otimizado, entre outras características, é executada e estimulada por todos os desenvolvedores da empresa. Esta prática é sustentada pelo código coletivo e a boa rede de interações. Assim, quando alguém se depara com uma função que pode ser melhorada, existe liberdade para fazê-lo. O criador da função pode ser consultado, o que geralmente ocorre, para avaliar mais profundamente as novas idéias e permitir a implementação da melhor solução.

3.2.8 Qualidade do *Software*

Os fatores mais relevantes que incorporam qualidade ao produto desenvolvido pela Opto Eletrônica serão apresentados na seqüência. Para melhor demonstrar o funcionamento do sistema de qualidade de *software*, a Tabela 3.3 mostra onde cada um destes fatores é aplicado e como se complementam, assegurando a confiabilidade necessária a um produto de alta tecnologia.

Tabela 3.3 – Práticas do Sistema de Qualidade de *Software*

Etapa	Prática de Qualidade	Equivalência no Modelo Tradicional
Definição de requisitos	Interação da equipe Interação com o cliente *Design* contínuo Prototipagem	Revisão de requisitos Prototipagem Validação do modelo
Modelagem/Análise/*Design*	Interação da equipe Interação com o cliente *Design* contínuo Código coletivo	Questionário/Checklist Validação de métricas Validação baseada em cenário Revisão do modelo
Codificação e testes	Interação da equipe *Design* contínuo Código coletivo	Revisão de código Inspeção de código Leitura do código
Integração e testes	Interação da equipe Interação com o cliente Integração contínua Código coletivo	Simulação Execução simbólica Testes de integração Testes de aceitação
Distribuição	Interação da equipe Interação com o cliente *Design* contínuo Código coletivo	Ferramentas de controle

Em comparação com a Tabela 2.4, notamos que duas práticas, de todas as propostas nas metodologias ágeis, não são implementadas integralmente em nosso processo: a programação em par e o desenvolvimento dirigido a testes. Elas são fatores importantes de garantia de qualidade, porém são também muito exigentes em termos de disciplina e empenho dos desenvolvedores. Talvez por isso, dos métodos ágeis mais difundidos, somente a programação extrema as defende.

O desenvolvimento dirigido a testes não é utilizado. Mesmo sem ele, o sistema é confiável, já que o código coletivo, o *design* contínuo e a interação entre os membros da equipe promovem a revisão constante de todas as partes do código por mais de um programador e a convergência para algoritmos simples e eficientes.

Como mostrado na seção anterior, somente as rotinas mais críticas são programadas em par. A não aplicação desta prática pode aumentar o número de erros de *design* e codificação, porém estes problemas são reduzidos devido à constante troca de informações entre os desenvolvedores, que discutem os algoritmos a serem implementados. No caso da codificação, a estratégia adotada é o teste cruzado, ou seja, o código é testado por um programador diferente daquele que o construiu, reduzindo significativamente o número de erros não detectados.

Todas as demais estratégias de teste indicadas na Seção 2.2.8 são implementadas como descrito nas subseções anteriores, que apresentam o método de desenvolvimento utilizado. Assim, comunicação direta entre os membros da equipe, atuação do cliente, integração contínua, *design* contínuo e código coletivo são sempre aplicados, assegurando a confiabilidade dos produtos gerados.

Por fim, assim como na metodologia tradicional, o processo de *software* apresentado utiliza-se da prototipagem como ferramenta para a definição de requisitos, com a vantagem que os protótipos podem ser aplicados em qualquer iteração do projeto, servindo-se também do *design* contínuo para permitir a introdução de novas funcionalidades a qualquer tempo.

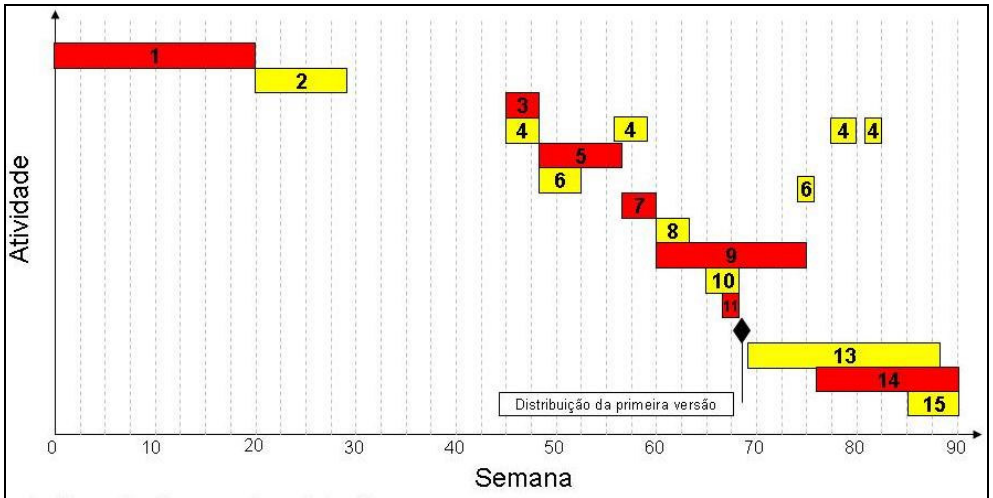
4 Desenvolvimento do MUX-GSE

Esta seção apresenta a implementação do processo proposto no desenvolvimento dos *softwares* do MUX-GSE. Foram contempladas as primeiras noventa semanas, a partir da data prevista para início do projeto, sendo que as quarenta e cinco últimas semanas foram de produção efetiva de *software*. Neste período, boa parte das funcionalidades especificadas foi construída e um grande número de problemas e situações inesperadas foi superado.

Apesar do trabalho se referir somente aos *softwares*, a forte característica multidisciplinar do MUX-GSE faz com que algumas etapas sejam diretamente interligadas a outros componentes do sistema. Assim, esta seção traz também informações sobre as partes do sistema que influenciaram decisões e/ou provocaram mudanças durante a condução do projeto.

Mostrando as situações enfrentadas, as decisões tomadas e as soluções implementadas, pretende-se comprovar a eficácia do processo de *software* proposto e detectar seus aspectos menos eficientes. Para atingir este objetivo, o projeto foi dividido em etapas, que serão apresentadas em cada subseção seguinte, trazendo, para cada caso, descrição, resultados e conclusões.

Para permitir melhor compreensão das atividades realizadas, buscou-se estabelecer uma relação temporal entre elas e definir de maneira clara o tempo gasto em sua execução. Com este intuito, o cronograma de desenvolvimento dos *softwares* do MUX-GSE é apresentado na Figura 4.1. Nele, as etapas são mostradas no prazo que ocorreram realmente e não naquele que estavam originalmente previstas.



1- Negociações para terceirização

2- Projeto Preliminar

3- Primeira iteração – Aprendizado, definição de estilo, estrutura de diretórios, estratégia de integração, etc.

4- Atualização da documentação do projeto

5- Painéis de equipamentos, tela principal do Controlador GSE, interfaces TCP/IP, GPIB e I/O digital

6- Prototipagem – TCP/IP, HSDIO e medida de MTF

7- Interfaces dos testes

8- Execuções de testes eletrônicos, funções de *log*, gravação e abertura de resultados

9- Testes e otimização da rotina de aquisição de imagens

10- Testes integrados *hardware/software*

11- Desenvolvimento de rotinas não originalmente previstas para testes da MUX

12- Distribuição da primeira versão

13- Depuração e otimização das rotinas de teste

14- Rotinas de auto-teste

15- Telas de controle do Colimador e do Simulador de Cena

Figura 4.1: Cronograma real de desenvolvimento dos *softwares* do MUX-GSE

4.1 Início do Projeto

O desenvolvimento do MUX-GSE teve um início difícil, pois se pretendia terceirizar o trabalho, entretanto, após vários meses de negociação, as propostas orçamentárias recebidas não eram satisfatórias. Assim, na data em que deveria começar o projeto completo, apenas os

físicos iniciaram suas funções, com o objetivo de caracterizar os equipamentos do banco óptico e selecionar marcas e modelos compatíveis.

O primeiro cronograma foi definido sem consulta à equipe de trabalho, uma falha grave, e reservava apenas cinqüenta dias para o desenvolvimento de todos os *softwares*. Dois meses depois, ainda sem encontrar propostas satisfatórias, o cronograma inicial sofreu as primeiras alterações. Novamente sem contato com a equipe, foram modificadas as datas para o desenvolvimento de *software*, com previsão de duração de somente cento e quarenta dias.

Mesmo assim, devido à expectativa de conseguir um fornecedor externo, os trabalhos só começaram cinco meses após a data proposta inicialmente. A princípio, foram escalados dois engenheiros eletricistas especializados em eletrônica. Outras duas pessoas com a mesma formação, porém menos experientes, seriam contratadas para compor a equipe, mas isso ocorreria somente em uma etapa posterior, após a apresentação do projeto preliminar. Além do desenvolvimento de *software*, estes engenheiros foram incumbidos da seleção de equipamentos para medidas eletroeletrônicas, sistemas eletromecânicos do banco óptico, plataforma de desenvolvimento, *racks*, interfaces de comunicação e todas as outras partes do sistema, excetuando apenas os dispositivos ópticos.

As primeiras reuniões foram utilizadas para colocar a equipe a par do que já vinha sendo feito pelos dois físicos responsáveis pelos testes ópticos. Também foram apresentados os marcos mais relevantes, prazos e exigências do trabalho. O cronograma de desenvolvimento de *software* foi considerado totalmente inviável. A opção foi pela distribuição da primeira versão utilizável, ainda não completa, mas funcional, em um prazo de sete meses, quando estava prevista a entrega do CGSE.

A princípio, a equipe concentrou esforços na análise dos requisitos do MUX-GSE, e definição das melhores soluções. Nos dois primeiros meses, o trabalho consistiu na redação de um pacote completo de documentos de projeto preliminar, que foi apresentado a uma banca

revisora do INPE como parte de um evento importante, a Revisão do Projeto Preliminar (*"Preliminary Design Review"* – PDR). Apesar do pouco tempo, a equipe conseguiu produzir especificações detalhadas dos equipamentos, circuitos, *softwares* e acessórios necessários.

Foi também redigida uma descrição funcional dos *softwares*, com explicações superficiais das funcionalidades pretendidas, informações sobre a definição da plataforma de desenvolvimento e implicações das características do *hardware*. Outra decisão importante foi a escolha da metodologia de desenvolvimento, que norteou todos os outros passos.

Estas informações compiladas para o PDR compreendem o primeiro bloco do fluxograma da Figura 4.2 (em vermelho), além de dados adicionais, necessários para satisfazer as exigências da licitação da MUX quanto aos documentos apresentados. As principais decisões desta etapa são resumidas nas próximas subseções.

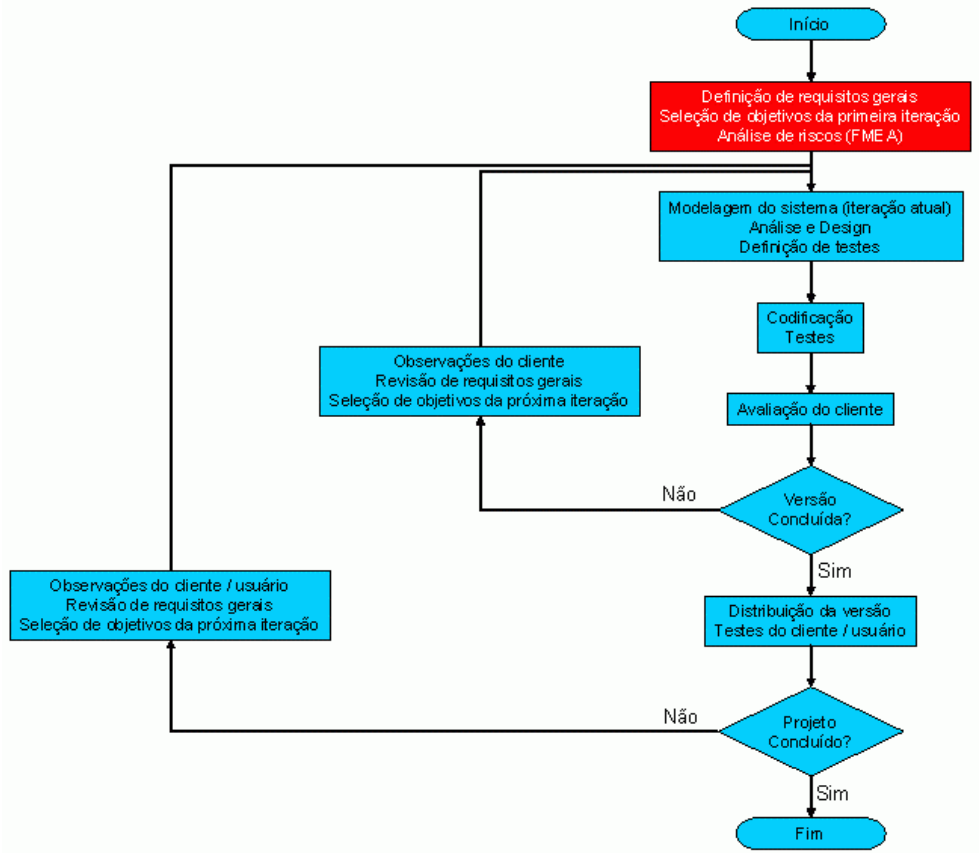


Figura 4.2: Fluxograma do processo de desenvolvimento de *software*, destacando o planejamento inicial

4.1.1 Definição da Metodologia

A discussão pelo método mais adequado de desenvolvimento (Figura 4.3) levou em conta características e requisitos do sistema que seria construído e da equipe disponibilizada para o trabalho. A princípio foi considerado o modelo clássico, já que o projeto tinha requisitos bem definidos e, por se tratar de um equipamento aeroespacial, com a rigidez típica deste tipo de trabalho, esta estratégia poderia obter sucesso.

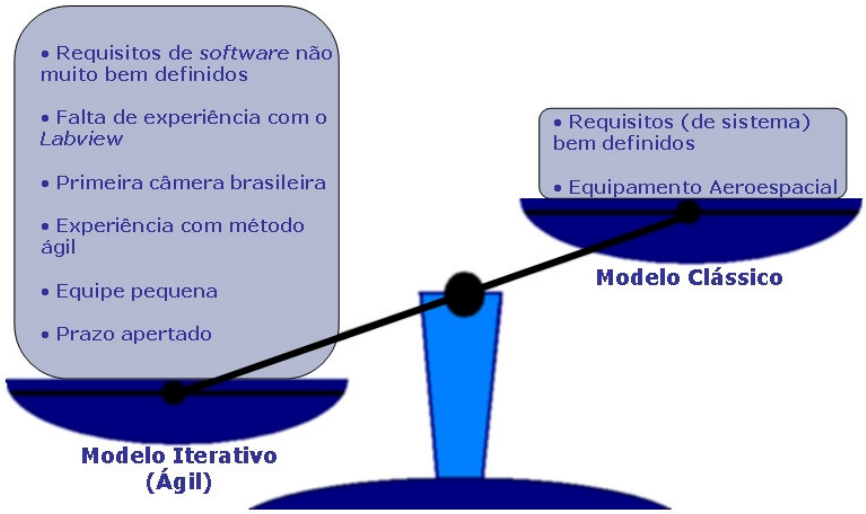


Figura 4.3: Seleção do modelo de desenvolvimento

Entretanto, foi constatado que os requisitos apresentados eram sobre o sistema, e não específicos para *software*. Além disso, não havia experiência prévia no desenvolvimento deste tipo de produto e o *Labview* nunca tinha sido utilizado na Opto Eletrônica para a criação de aplicativos profissionais de grande porte. E mais: tanto a MUX quanto o MUX-GSE são produtos experimentais, sem predecessores brasileiros, e poderiam necessitar de características diversas daquelas originalmente propostas.

Considerando todas estas informações, concluiu-se que um método iterativo seria mais adequado ao projeto e então, para evitar maiores fatores de risco que os já existentes, foi eleita a mesma metodologia (já apresentada na Seção 3.2) usada com sucesso no desenvolvimento de equipamentos médicos e industriais, com o mínimo possível de adaptações.

4.1.2 Opção pela Plataforma *Labview*

A definição da plataforma de desenvolvimento foi assunto de uma das primeiras discussões. A decisão, entretanto, só ocorreu depois que a equipe decidiu aplicar instrumentação virtual e selecionou diversas placas da *National Instruments* para emulação dos equipamentos de medida, aquisição de dados e I/O digital.

O *software* para estas placas pode ser desenvolvido em qualquer ambiente que aceite controles *ActiveX*[15], assim houve dúvida entre a utilização do *Labview* ou do C++ *Builder*, este último aplicado até então nos projetos que necessitavam interfaces gráficas, e que vinha sendo utilizado pelos dois engenheiros da equipe há alguns anos.

Na opção pelo *Labview*, foram consideradas:

- Sua rápida curva de aprendizado, que facilitaria o entrosamento dos membros que viriam integrar a equipe.
- Sua biblioteca com funções estatísticas, trigonométricas, processamento de matrizes, análise no domínio da freqüência e diversas outras operações necessárias ao MUX-GSE.

[15] Controles *ActiveX* são componentes reutilizáveis, baseados no Modelo de Objeto Componente ("*Component Object Model* – COM), da Microsoft. Eles são componentes encapsulados que podem ser utilizados por programas diversos. Várias plataformas, como *Visual* C++, *Delphi*, C++ *Builder* e outras, são compatíveis com controles *ActiveX* e permitem adicioná-los aos softwares desenvolvidos.

- A possibilidade de desenvolver rapidamente interfaces para as placas selecionadas, o que permitiria a criação de uma versão básica do *software* em muito pouco tempo.
- A vasta gama de programas-exemplo e aplicativos prontos disponibilizados no site da *National Instruments*[16].

Durante a execução do trabalho, foram detectadas algumas limitações do *Labview* que talvez nos fizessem preteri-lo para este projeto, entretanto, na fase inicial ele nos pareceu a opção mais adequada. Entre os principais problemas encontrados, podemos citar:

- Dificuldade em se produzir interfaces gráficas confiáveis, considerando todas as possibilidades do usuário: os painéis produzidos rapidamente são utilizáveis, porém ações indevidas geralmente provocam erros na execução do programa. Criar mecanismos que limitem a utilização dos controles para permitir somente ações válidas é possível, porém exige grande esforço de programação em comparação com outras plataformas de desenvolvimento.
- Lentidão na atualização da interface: mudar características de muitos itens visíveis geralmente é muito lento, tornando possível observar as modificações, o que empobrece o visual.
- Esforço necessário para a inclusão de novos comandos: em linguagens de texto, em geral, basta um *"Enter"* a e inserção de uma nova linha de código. Em *Labview*, muitas vezes é preciso arrastar uma série de blocos e linhas e reorganizar o diagrama para que se possa acrescentar um VI[17].

Apesar destes problemas, a facilidade em se trabalhar com matrizes, as poderosas funções de processamento disponibilizadas e o gerenciamento automático da memória

[16] www.ni.com
[17] VI (*"Virtual Instrument"* – Instrumento Virtual) é o nome dado no *LabView* tanto para aplicativos contendo interface (*"Front Panel"*) e diagrama de ligações, quanto para as funções definidas independentemente (que também podem ser chamadas de *SubVIs*). Pode-se fazer uma analogia a uma função ou procedimento de um software desenvolvido em linguagens tradicionais, como o C ou Pascal.

compensam o tempo gasto e nos levam a concluir que o *Labview* foi, no mínimo, uma boa opção para o projeto.

4.1.3 Funções do *Software*

Ainda nesta etapa inicial, foram apresentadas as funções pretendidas para os *softwares* do MUX-GSE e uma explicação básica de como se pretendia implementá-las. Como a equipe havia optado por uma abordagem ágil, não foi executado projeto detalhado das rotinas, apenas uma descrição funcional das mesmas. A Tabela 4.1 traz um pequeno resumo dos *softwares* desenvolvidos e as funções pretendidas para cada um deles.

Tabela 4.1 – Características dos *softwares* do MUX-GSE

Aplicativo	Principais funções	Interfaces necessárias	Funções adicionais
Controlador GSE	Auto-teste Interface para o banco óptico Interface para 6 equipamentos de laboratório 16 testes eletrônicos automáticos 11 testes ópticos automáticos 3 testes mecânicos pré-definidos	LAN I/O digital GPIB	Gravação e abertura de resultados Impressão de resultados Gravação e gerenciamento de log Calibração de instrumentos
Sistema de Exibição de Imagens	Auto-teste Captura de imagens em tempo real Decodificação de imagens Exibição em tempo real Transmissão em tempo real	LAN I/O LVDS[18]	Gravação e abertura de imagens Impressão de imagens Gravação de log
Cliente	Autenticação de usuário Conexão com o Controlador GSE Visualização de dados de todos os testes	LAN	Status da conexão

Durante a produção deste documento inicial, a equipe discutiu como seria executada cada uma das rotinas, sem detalhes aprofundados, porém atentando para possíveis dificuldades técnicas. Esta análise preliminar de riscos permitiu identificar funções críticas, elegíveis para prototipagem, que necessitavam de *hardware* especial ou de *software*

[18] LVDS (*"Low Voltage Differential Signaling"* – Sinalização Diferencial de Baixa Tensão) é um sistema de transmissão de dados digitais que utiliza a diferença de tensão entre dois cabos para codificar a informação. Ele consegue atingir alta taxa utilizando cabos par-trançado, que são relativamente baratos. É padronizado pela norma ANSI/TIA/EIA-644-A, publicada em 2001.

extremamente otimizado. A equipe da MUX e os físicos envolvidos no projeto também participaram de algumas reuniões para que todos compreendessem os desejos e necessidades dos demais.

Para complementar o documento, foram criados Diagramas de Fluxo de Dados, DFD (PRESSMAN, 2005), de níveis 0 e 1 para os aplicativos do Controlador GSE e do SEI e somente de nível 0 para o aplicativo cliente (Figura 4.4 a Figura 4.8), permitindo visão global do trabalho que seria executado. Os diagramas não foram detalhados a outros níveis, pois eles não teriam valor prático em uma abordagem ágil.

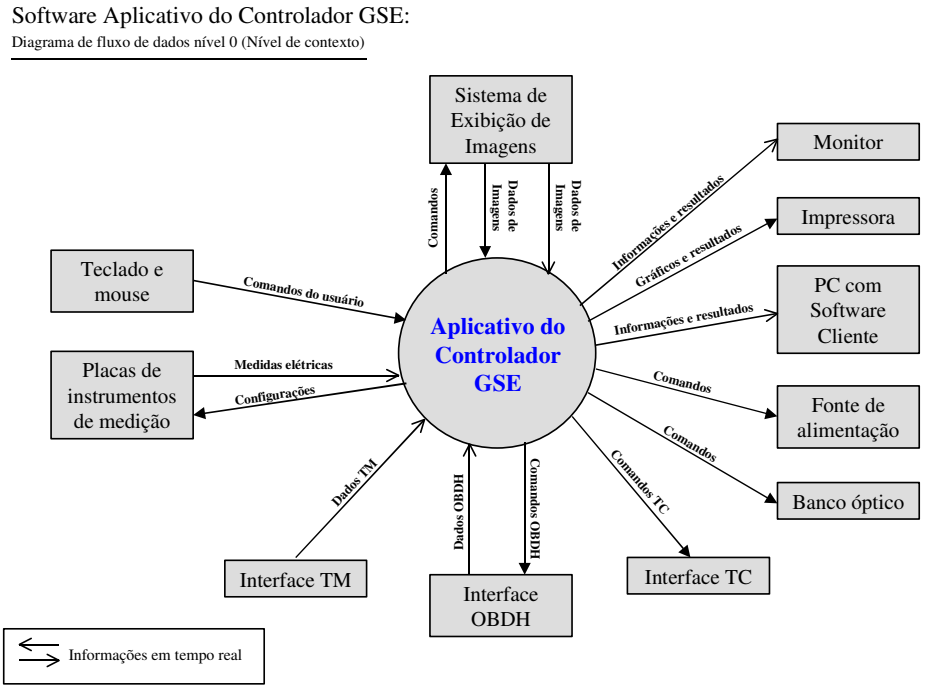


Figura 4.4: Diagrama de fluxo de dados de nível 0 do Controlador GSE

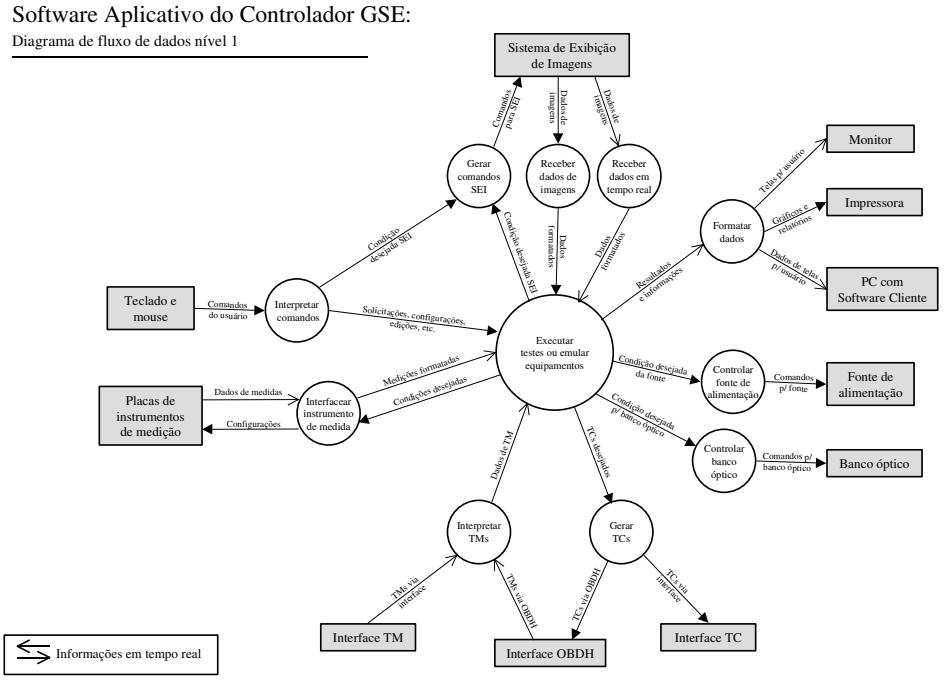


Figura 4.5: Diagrama de fluxo de dados de nível 1 do Controlador GSE

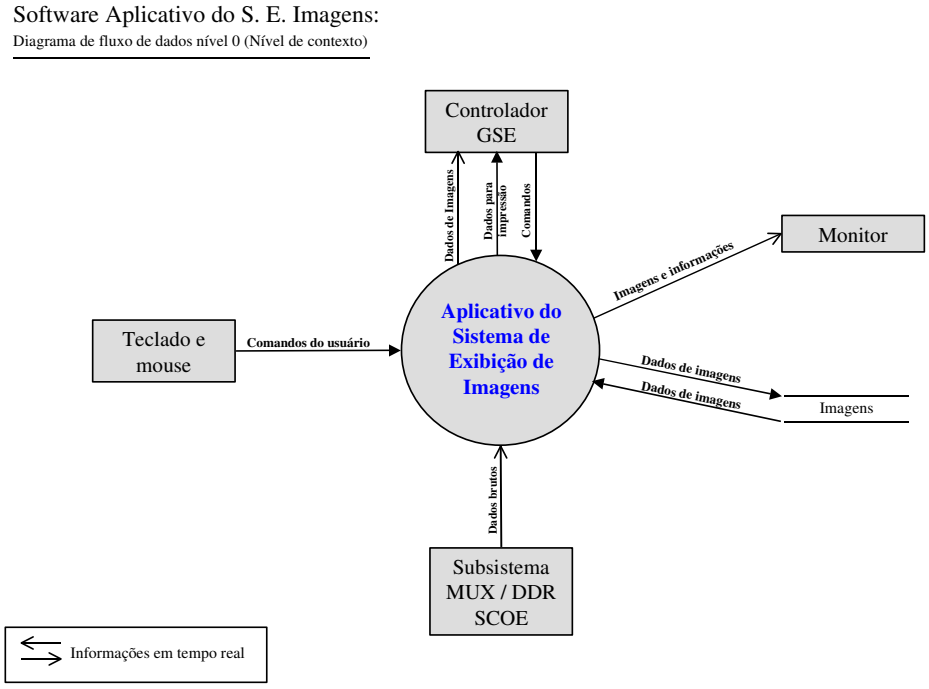


Figura 4.6: Diagrama de fluxo de dados de nível 0 do SEI

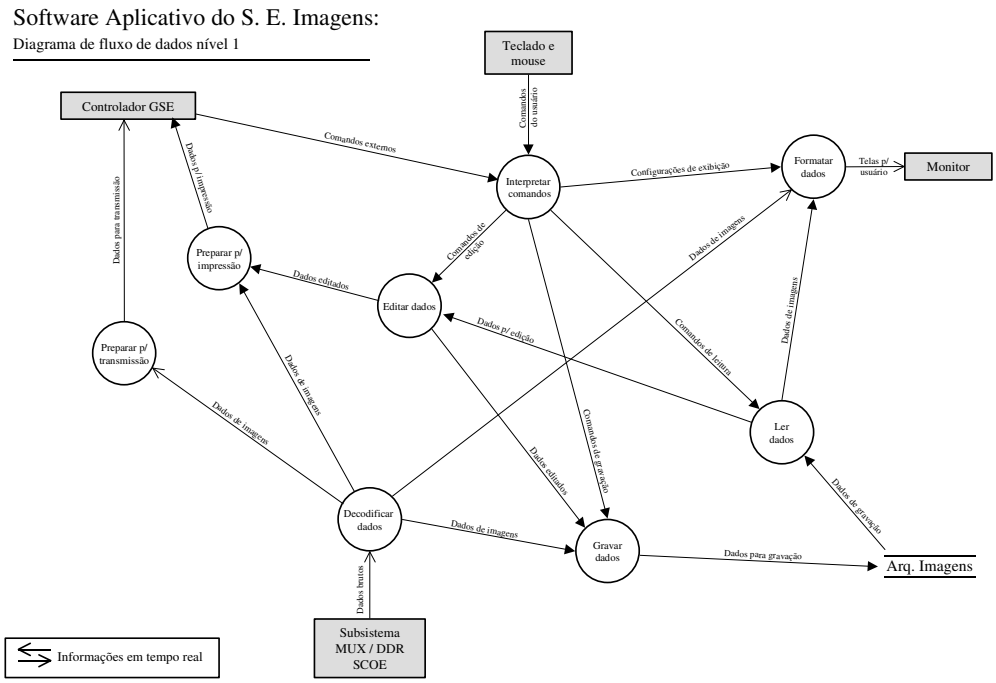


Figura 4.7: Diagrama de fluxo de dados de nível 1 do SEI

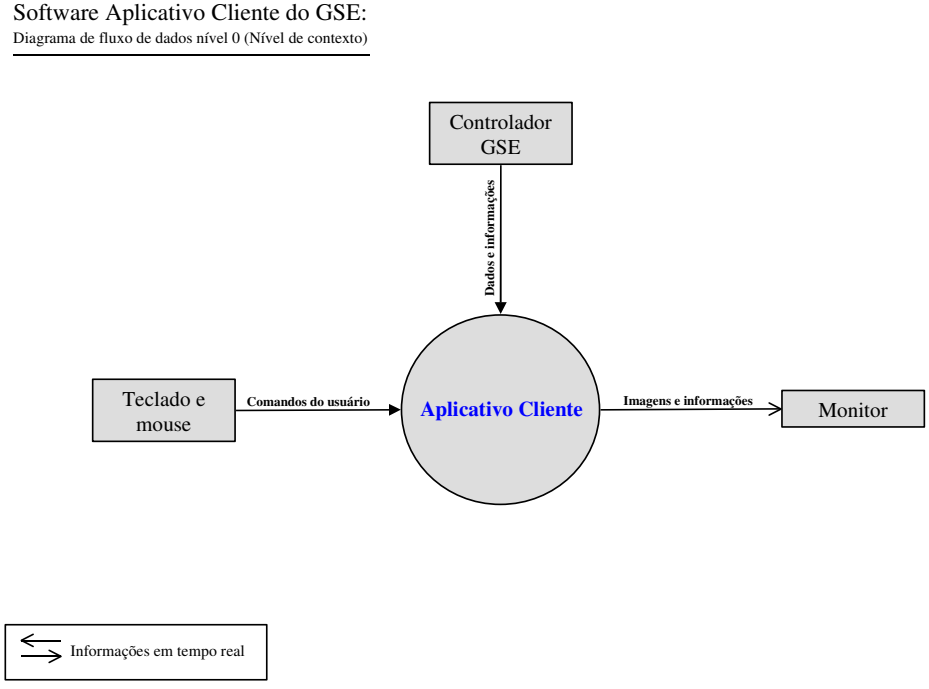


Figura 4.8: Diagrama de fluxo de dados de nível 0 do aplicativo cliente

Após a entrega do PDR, houve ainda algum tempo consumido em atividades de planejamento não diretamente relacionadas ao *software*. Ocorreu a apresentação do projeto preliminar aos especialistas do INPE, responsáveis por sua análise, solicitação de modificações e posterior aprovação, que propuseram a revisão de algumas informações.

Entre as solicitações estavam as seguintes: a avaliação detalhada dos erros de medida, para comprovação da capacidade do sistema em cumprir os requisitos propostos, e a expansão das funções de auto-teste, visando assegurar que em hipótese nenhuma o MUX-GSE pudesse danificar a câmera. Para isso, foi projetado um sistema para testar todos os pinos dos conectores destinados à MUX antes de efetuar sua ligação. Esta tarefa consumiu o tempo de trabalho da equipe por aproximadamente três semanas.

Na seqüência, os trabalhos foram interrompidos por algum tempo. Os engenheiros da equipe tiveram que resolver pendências de projeto que estavam envolvidos anteriormente. Isto ocorreu porque, como já foi dito, havia a intenção de terceirizar o desenvolvimento do MUX-GSE, e, portanto, não havia pessoas pré-escaladas para a condução do trabalho. Neste período, trabalhando apenas parte do tempo, eles apenas coordenaram a negociação e aquisição dos equipamentos eletrônicos selecionados.

Ainda nesta etapa, foram integrados os dois outros engenheiros à equipe. Em suas primeiras semanas de trabalho, dedicaram-se ao estudo dos documentos de projeto já existentes, para que pudessem compreender o trabalho a ser realizado.

Com a chegada de *racks*, computadores industriais, monitores e fonte de alimentação, a equipe passou a trabalhar também na montagem dos *racks*, integração dos equipamentos e solicitação de acessórios mecânicos necessários. Ao mesmo tempo, teve início o desenho eletrônico detalhado, com a construção de todos os esquemas elétricos dos circuitos de

interface, seguido do *design* das placas eletrônicas, que contou ainda com a ajuda de técnicos em eletrônica.

Logo que chegaram as placas de aquisição, I/O e emulação de equipamentos e o *software Labview*, a equipe realizou a montagem do *hardware* e buscou testá-lo através das ferramentas fornecidas pela *National Instruments*. Deu-se então início à primeira iteração no processo de *software*, sem interrupção do desenvolvimento das placas de interface. Com isso, o desenvolvimento começou oito meses após a data prevista, porém o atraso não foi tão grande se levarmos em conta que todo o projeto da MUX teve o cronograma revisto, devido a uma série de problemas. Nesta época, os prazos para o MUX-GSE já haviam sido redefinidos, principalmente por dificuldades nas negociações com fornecedores para o colimador[19] e os simuladores de cena[20].

4.2 Primeira Iteração

Ao iniciar o desenvolvimento, quatro meses após a definição dos documentos iniciais do projeto, a equipe reuniu-se e estabeleceu um objetivo inicial: aprofundar os conhecimentos de programação em *Labview*. Os quatro membros da equipe tinham alguma familiaridade com a plataforma, mas nenhum tinha boa experiência em sua utilização e, portanto, era preciso melhor preparação para os trabalhos. A *National Instruments* já havia sido consultada quanto à realização de cursos, porém a cotação apresentada não foi considerada satisfatória. Com isso, os programadores tiveram que buscar material e aprender as melhores práticas por conta própria.

[19] O colimador é um sistema óptico cujo objetivo é projetar alvos de testes a distâncias muito longas, consideradas infinitas. Ele é utilizado para avaliar a capacidade da câmera MUX de captar imagens da terra a partir do espaço, simulando as condições de operação. É parte do banco óptico do CGSE.

[20] O simulador de cena é um colimador simplificado, menor e mais leve, capaz de gerar um sinal óptico que permite a realização de testes menos complexos, apenas para garantir que as condições da MUX não se modificaram após o transporte e a integração com o satélite. Ele compõe o SGSE.

Para um primeiro contato com o *Labview*, foram selecionados quatro módulos independentes do *software* (comunicação LAN, painéis do gerador de formas de onda digitais, comunicação GPIB e de visualização das imagens da câmera) para construção utilizando a estratégia Codificar e Corrigir, com cada um dos membros da equipe trabalhando independentemente. Depois de duas semanas e meia, foram realizadas várias reuniões para discutir as possibilidades e problemas encontrados.

O documento "*Labview*: *Development Guidelines"* (NATIONAL INSTRUMENTS, 2003) foi discutido e definiu-se um estilo de programação usando suas informações. Foi montada uma estrutura básica, com um diretório para cada um dos aplicativos e com suas funções divididas em grupos, segundo características comuns. Foi selecionado o local de integração do *software* e a estratégia que seria adotada. O trabalho para o próximo ciclo também foi discutido e dividido entre os programadores. Os subitens seguintes sumarizam as decisões desta reunião.

4.2.1 Estilo de Programação

Ao definir um estilo de programação, a equipe de desenvolvimento buscou evitar muitas regras, reduzindo dificuldades na execução do trabalho, que tomam tempo sem agregar benefícios ao projeto. Por outro lado, buscou-se um estilo único, com código claro e inteligível, para dar suporte ao código coletivo e ao *design* contínuo.

As boas práticas apresentadas no *"Development Guidelines"*, da *National Instruments* (2003) foram adotadas, entre elas, podemos citar:

- Posicionar os fios (linhas que representam a transmissão de dados no *Labview*) de modo que possam ser vistos claramente, do início ao fim, sem cruzar por baixo de estruturas ou subVIs[21].
- Desenvolver VIs cujo código pode ser observado em uma única tela, sem a necessidade de movimentar as barras de rolagem. Quando isto não é possível, tentar posicionar todo o código de modo que somente uma barra de rolagem, vertical ou horizontal, seja utilizada.
- Utilizar as ferramentas de alinhamento, espaçamento, etc. para posicionar os componentes de maneira que valorize a observação do diagrama e a compreensão do código.
- Não utilizar fios da direita para a esquerda; assim, a lógica do programa é a execução da esquerda para a direita, guiada pela fiação e pelas estruturas.
- Usar comentários com moderação para explicar a função das estruturas, identificar o conteúdo de fios longos, a utilidade de constantes e para explicar a função do código quando ela não é óbvia.
- Evitar o uso de estruturas seqüenciais[22], baseando a seqüência de execução no fluxo de dados determinado pelos fios, especialmente pelos de erro.
- Como as linhas de erro são usadas para garantir a seqüência de execução, todos os subVIs criados devem possuir entrada e saída de erros.

[21] Em *Labview* utiliza-se o termo VI (*"Virtual Instrument"* – Instrumento Virtual) para denominar uma rotina, composta de um diagrama e um painel de interface (que pode ou não ficar visível ao usuário final). Os VIs auxiliares, utilizados para executar tarefas específicas dentro dos VIs principais são denominados SubVIs, em analogia às subrotinas de um *software* desenvolvido em linguagem de texto.

[22] Estruturas seqüenciais (*"Sequence Structures"*) são estruturas lógicas do *Labview* formadas por quadros seqüenciais usados para a inserção de código que deve ser executado em etapas, na ordem definida pelos quadros. São evitadas por quebrar o paradigma do fluxo de dados da direita para a esquerda.

- Todos os subVIs devem ser identificados por um ícone com seu nome escrito em fonte pequena. Cada grupo de funções, definido na estrutura de diretórios, é identificado por uma cor de fundo nos ícones.
- Evitar o uso de variáveis locais, utilizando fios, sempre que possível, ou nós de propriedades[23], que possuem entrada e saída de erros, facilitando a observação do fluxo de dados.
- Dividir todos os VIs em subVIs, até que cada um seja responsável por uma única tarefa (da mesma forma que cada função em linguagem de texto deve ter um objetivo único).

Estas práticas foram discutidas e aceitas em consenso. Algumas foram registradas no livro de projetos e outras apenas discutidas verbalmente. Todas devem ser aplicadas com bom senso e podem ser ignoradas quando conveniente, pois o importante é o funcionamento, a clareza e a manutenabilidade do *software*, e não a observância rígida de normas.

A Figura 4.9 mostra um exemplo de diagrama construído seguindo as práticas expostas acima. Todos os VIs do MUX-GSE têm aparência similar a este, facilitando a compreensão por parte de todos os membros da equipe e a integração de novos membros a qualquer momento.

[23] Nós de propriedade (*"Property Nodes"*) são estruturas do *Labview* que permitem o acesso às propriedades dos objetos para leitura ou escrita.

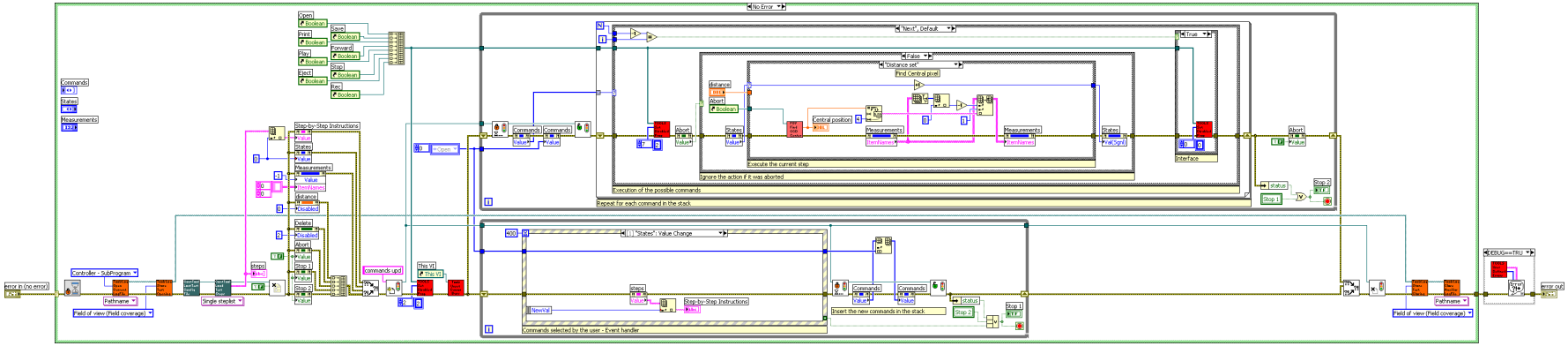


Figura 4.9: Diagrama construído seguindo as práticas propostas

4.2.2 Estrutura de Diretórios

A definição inicial de uma estrutura de diretórios foi realizada para facilitar a identificação dos VIs, de acordo com suas funções e permitir o trabalho dos quatro programadores com o mínimo risco de modificação simultânea de uma rotina. Assim, após dividir o trabalho, cada um modificará arquivos em certos diretórios, sem necessidade de acesso à maioria dos outros locais. Ela é importante para sustentar a integração contínua

A hierarquia seguiu a natureza dos equipamentos aos quais as funções se referem. Assim, existe um diretório do Controlador, contendo um diretório para cada instrumento de medida, além do de testes, que por sua vez é dividido para cada um dos testes propostos e assim por diante. Existem ainda diretórios para ferramentas genéricas e para funções de interface com as placas de aquisição. A Figura 4.10 mostra uma parte da estrutura utilizada.

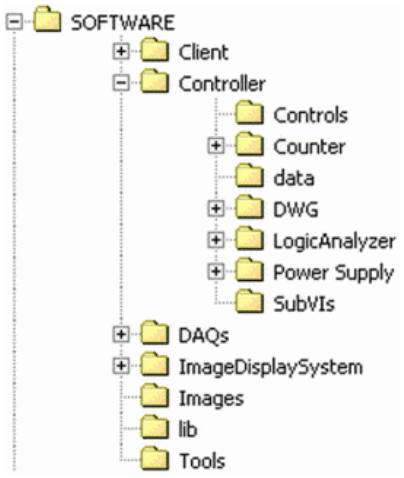


Figura 4.10: Parte da estrutura de diretórios definida para o projeto

4.2.3 Estratégia de Integração

Para permitir a implementação da integração contínua, além da definição da estrutura de diretórios, foi selecionada uma área na rede corporativa exclusiva para armazenamento da versão mais atualizada do *software*. Acordou-se que ali seriam adicionadas as modificações realizadas, tão logo elas concretizassem uma nova utilidade.

Para evitar integrações simultâneas, ficou estabelecido que antes de modificar o conteúdo desta área, o desenvolvedor deveria avisar todos os demais e só poderia liberar o diretório após cumprir os testes necessários para assegurar o perfeito funcionamento das mudanças e das demais partes do programa influenciadas por elas.

Por questão de segurança, propôs-se que ao realizar uma integração seria armazenada uma cópia da versão mais atual em um diretório específico, na mesma área da rede reservada à versão corrente. As cópias de segurança deveriam ser mantidas durante alguns dias, para que rotinas danificadas acidentalmente pudessem ser recuperadas.

4.2.4 Conclusões

Como visto, a primeira iteração compreendeu apenas uma reunião inicial e a atividade de codificação, ignorando a modelagem, análise e todas as demais partes. Mesmo assim, os objetivos propostos foram atingidos, pois a intenção não era gerar código, mas o aprendizado do *Labview* e a definição de técnicas profissionais para sua utilização.

As reuniões no final deste primeiro ciclo foram muito produtivas. A partir da primeira delas foram discutidas diversas idéias, até que, após alguns dias, a equipe convergiu para os resultados apresentados nas subseções anteriores. Na reunião de conclusão foram definidos os

objetivos e responsabilidades para a próxima etapa. Todos também concordaram que o código gerado até então era de muito baixa qualidade e necessitava de revisão ou, em alguns casos, descarte.

4.3 Painéis de Instrumentos

Após três semanas de aprendizado, teve início uma etapa de desenvolvimento normal, composta de vários ciclos, com o objetivo de produzir *software* de alta qualidade e em curto espaço de tempo. As iterações seguintes seriam completas, com atividades de planejamento, modelagem, codificação, testes e avaliação.

Na definição dos objetivos desta etapa, a equipe considerou prudente concentrar-se na produção de módulos independentes, que não exigissem muita interação entre diversas partes do *software*, deixando atividades mais complexas para quando todos tivessem adquirido segurança na utilização do *Labview* e as práticas definidas na etapa inicial estivessem consolidadas.

Foram então eleitos para desenvolvimento os painéis de comando dos equipamentos eletrônicos de laboratório: fonte de alimentação, analisador lógico, gerador de formas de onda digitais, freqüencímetro, osciloscópio e multímetro. Além do *software* para estes equipamentos, esta etapa compreendeu ainda a criação das telas principais (*design* gráfico e planejamento funcional) do Controlador GSE e do SEI e a realização de testes de comunicação via TCP/IP, através de prototipagem.

O osciloscópio e o multímetro não foram incluídos na lista de tarefas de desenvolvimento, pois a *National Instruments* fornece painéis prontos (Figura 4.11), que foram avaliados pela equipe e considerados apropriados para as necessidades do projeto.

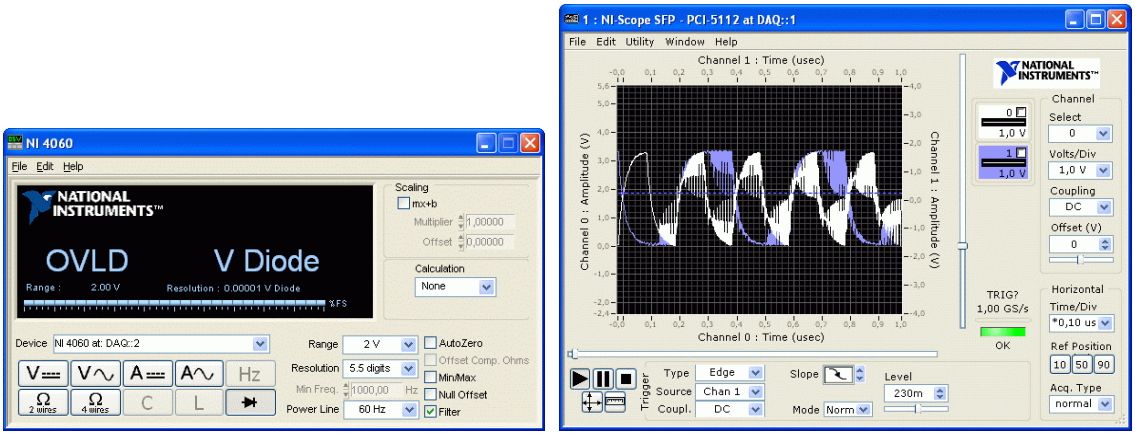


Figura 4.11: Painéis de controle do multímetro (esq.) e osciloscópio (dir.)

A fonte de alimentação e o freqüencímetro são equipamentos de bancada, com controle via GPIB. O *software* desenvolvido para estes instrumentos é uma interface gráfica com os controles do aparelho, que transmite comandos e recebe informações, exibindo-as ao usuário. Antes de sua construção, foram criados VIs para transmissão e recepção via GPIB.

Já o analisador lógico e o gerador de formas de onda são instrumentos virtuais. Além do desenvolvimento da interface, exigem construção de todo o código para controle da placa PCI usada em sua emulação, como, por exemplo, a construção na memória das formas de onda a serem geradas. Eles são emulados através de uma única placa com trinta e dois canais de I/O digital de alta velocidade. Antes de sua codificação, foram construídos protótipos para avaliar as capacidades da placa e as funções disponibilizadas mais adequadas às necessidades do MUX-GSE.

Durante o planejamento foi decidido que os painéis de equipamentos seriam construídos somente com os controles essenciais, sem opções que não fossem relevantes. Estes comandos poderiam ser acrescentados no futuro, caso se tornassem necessários. Definiu-se ainda que cada programador ficaria responsável por um módulo, trabalhando

isoladamente, mas discutindo constantemente suas atividades com os demais. Ao final da etapa, o trabalho havia sido dividido como mostra a Tabela 4.2.

Tabela 4.2 – Divisão do trabalho na segunda etapa de desenvolvimento

Atividade	Programador(es)
Interface do Controlador GSE	1
Protótipo para Acesso ao I/O Digital de Alta Velocidade	1
Painel do Gerador de Formas de Onda Digitais	1
Painel do Analisador Lógico	1
Acesso GPIB	2
Painel da Fonte de Alimentação	2
Painel do Freqüencímetro	2
Protótipo para Comunicação TCP/IP	3, 4
Acesso ao I/O Digital e Analógico	3
Interface de Telecomandos e Telemetrias	3, 4
Interface do Sistema de Exibição de Imagens	2, 3, 4
Acesso ao I/O Digital de Alta Velocidade LVDS	4

Como era de se esperar em um processo ágil, esta divisão de trabalho não aconteceu previamente, mas foi sendo definida a cada iteração. A troca de informações durante o desenvolvimento foi constante, com os programadores ajudando uns aos outros para manter o mesmo nível de qualidade. Eles também trocavam *software* com freqüência, para que pudessem testar o produto uns dos outros.

O projeto das telas de interface com o usuário sempre contou com a opinião de todos os programadores e, muitas vezes, membros da equipe da MUX foram chamados para avaliar e discutir o *design* gráfico, os controles disponibilizados e o funcionamento ideal. O objetivo foi construir painéis com boa aparência, mas atentando principalmente para a funcionalidade. Como os usuários finais deveriam ser engenheiros, a apresentação do maior número possível de informações foi considerada uma boa opção, porém evitando excesso de poluição visual.

4.3.1 Resultados

A segunda etapa teve duração total de nove semanas. Neste prazo, as seguintes atividades foram realizadas:

- **Tela Principal do Controlador**

  Esta tela (Figura 4.12) estava concluída e funcional, exceto por alguns botões ainda não habilitados, como o do Colimador e os de testes automáticos, que aguardavam a codificação das funções a que seriam destinados. Ela foi construída em tamanho pequeno, para permitir que fique aberta em um canto da tela, enquanto os instrumentos ou testes são executados. Foi planejada de maneira que permite a utilização de vários equipamentos ao mesmo tempo e verifica, ao iniciar um procedimento de teste, quais painéis precisam ser fechados, avisando o usuário e aguardando confirmação.

Figura 4.12: Tela principal do Controlador GSE

- **Acesso ao I/O Digital de Alta Velocidade**

Antes do início do desenvolvimento do gerador de formas de onda digitais e do analisador lógico, foi necessário testar as rotinas de acesso à placa de I/O digital de alta velocidade. Estas funções, disponibilizadas pelo *Labview*, são agrupadas em um conjunto identificado como HSDIO (*"High-Speed Digital Input/Output"* – Entrada/Saída Digital em Alta Velocidade).

As funções de acesso HSDIO são simples, porém os exemplos fornecidos não traziam todas as características necessárias ao MUX-GSE. Isto foi uma constante durante o projeto, pois a maioria dos exemplos da *National Instruments* é construída utilizando os assistentes disponibilizados pelo *Labview*. São práticos e fáceis de codificar, porém não permitem muitas opções, limitando a flexibilidade do produto. Com isso, na grande maioria dos casos, os exemplos não ajudaram muito. Os assistentes tiveram que ser descartados e o código construído utilizando as funções básicas disponíveis.

Assim, foi necessário criar aplicativos simples para testar as funções de entrada/saída de dados com as opções de *trigger*, *clock*, taxa de amostragem, etc. que seriam utilizadas nos instrumentos virtuais definitivos. Estes testes tomaram aproximadamente duas semanas de trabalho. Após sua conclusão, as rotinas foram descartadas e as idéias aplicadas em sua construção serviram para a codificação das funções de entrada/saída de dados digitais para o gerador de formas de onda e o analisador lógico.

- **Gerador de Formas de Onda Digitais**

Ao final da etapa, o painel de controle do gerador de formas de onda digitais (Figura 4.13) estava concluído e era totalmente funcional. O equipamento foi projetado com capacidade de gerar até oito ondas simultâneas, apesar da placa de I/O digital permitir até trinta e dois canais. Esta decisão foi baseada no acordo de produzir somente os comandos mais urgentes, com possibilidade de evolução em etapas posteriores. A equipe considerou que oito canais seriam suficientes e, se fossem necessários mais canais, seria possível modificar o *software* a qualquer momento.

O instrumento virtual construído permite a geração de um valor fixo (0 ou 1), valores alternados com freqüência igual a uma fração do *clock* básico, ou saída aleatória. É possível ainda personalizar a forma de onda, definindo o valor desejado para cada ponto gerado.

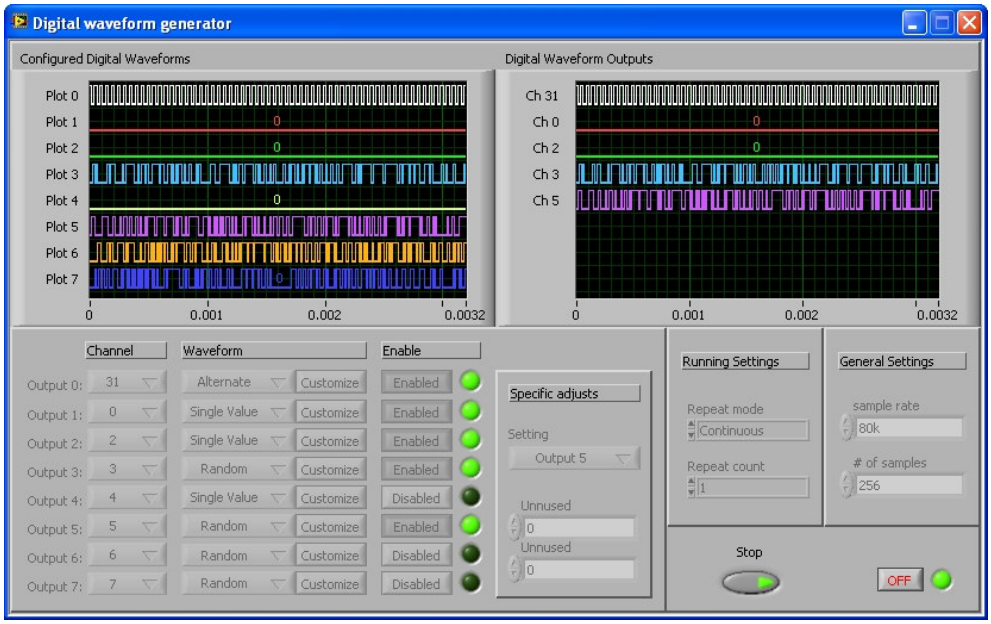


Figura 4.13: Painel de controle do Gerador de Fomas de Onda Digitais

- **Analisador Lógico**

O painel de controle do analisador lógico (Figura 4.14) também foi concluído nesta etapa. O instrumento virtual foi projetado com capacidade para até dezesseis entradas digitais, ao invés das trinta e duas permitidas pela placa de I/O. Apesar desta redução, considerou-se que provavelmente o equipamento seria capaz de suprir todas as necessidades do projeto.

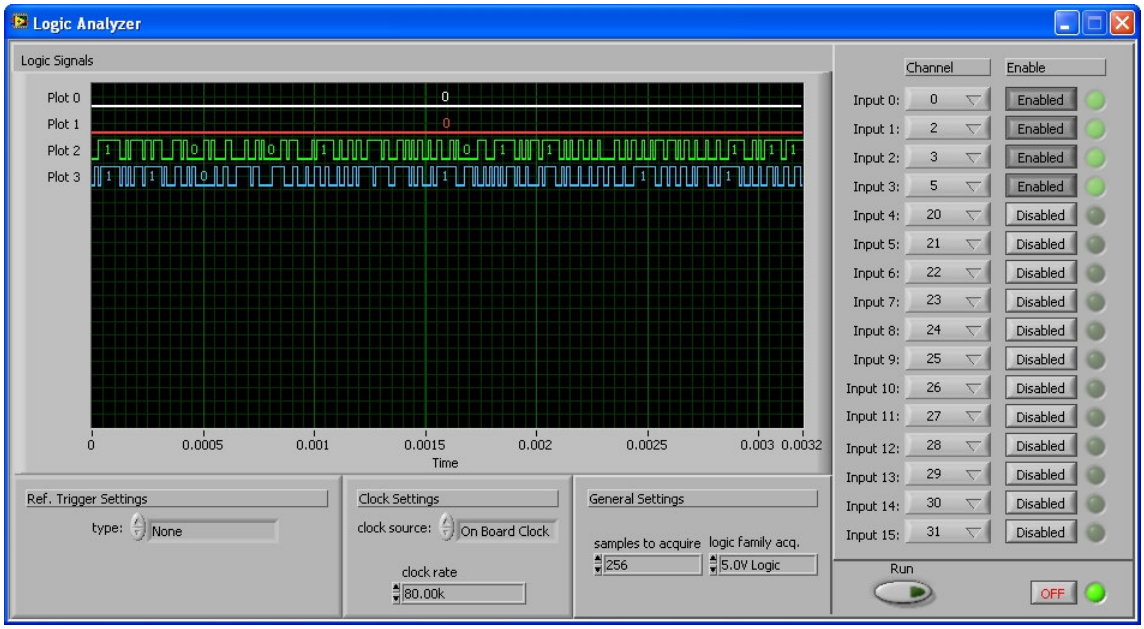


Figura 4.14: Painel de controle do Analisador Lógico

- **Acesso GPIB**

O MUX-GSE utiliza uma placa PCI com interface GPIB. Esta placa pode ser controlada através de um conjunto de funções básicas integradas ao *Labview*. A partir dele, nesta segunda etapa de desenvolvimento, foram criados VIs para facilitar a programação dos painéis de controle dos equipamentos de bancada controlados por esta interface.

As funções desenvolvidas permitem identificar o equipamento, enviar comandos, receber dados, avaliar códigos de erro, etc. Este conjunto não foi construído na forma de protótipo, porque não se considerou necessário. As funções disponíveis já eram simplificadas e então optou-se pela programação direta dos VIs definitivos. Nesta etapa, foram utilizadas na construção dos painéis de controle da fonte de alimentação e do freqüencímetro.

- **Fonte de Alimentação**

O painel de controle da fonte de alimentação (Figura 4.15) contém todos os comandos disponíveis no equipamento, permitindo funcionamento com tensão ou corrente controlada, acionamento das duas saídas disponíveis (5V ou 25V) e habilitação ou inibição da saída. Ele possui ainda um indicador imitando um display digital, com todas as informações provenientes da fonte. Todo seu funcionamento é baseado nos VIs de comandos GPIB. Este painel estava totalmente concluído ao final da segunda etapa do trabalho.

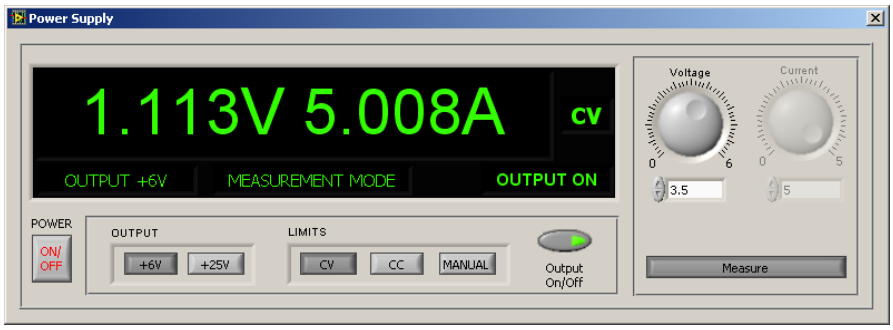


Figura 4.15: Painel de controle da Fonte de Alimentação

- **Freqüencímetro/Contador**

Assim como a fonte de alimentação, o freqüencímetro utilizado é um equipamento de bancada comandado via GPIB. Seu painel de controle (Figura 4.16)

foi desenvolvido utilizando os VIs de acesso a esta interface. Neste caso, não foram disponibilizadas todas as opções possíveis, mas somente as relevantes para o projeto, economizando tempo de trabalho que seria gasto na produção de rotinas que provavelmente nunca serão necessárias.

Figura 4.16: Painel de controle do Freqüencímetro

- **Interface TCP/IP**

Logo no início da etapa foi construído um protótipo para comunicação TCP/IP. Como a equipe não tinha nenhum especialista neste protocolo e as rotinas do *Labview* não são tão bem definidas quanto as de outras interfaces, a opção foi testar o funcionamento utilizando programas simplificados. Esta escolha foi a mais acertada, pois a construção de um servidor TCP/IP causou bem mais problemas que o esperado.

A execução do servidor para a rede do MUX-GSE em *Labview* em uma máquina conectada à rede interna da Opto Eletrônica, somada a uma configuração inapropriada do sistema operacional (Windows XP), provocou a geração de uma infinidade de mensagens de erro no servidor da empresa, paralisando totalmente a rede interna. Este fato prejudicou uma série de operações, como requisições de compra, controle de estoque, atendimento de pedidos de clientes, etc. A causa da falha só foi

detectada pelo departamento de Tecnologia da Informação (TI) depois de mais de quarenta e oito horas.

Após a solução dos problemas iniciais, o servidor produzido mostrou-se eficiente e seguro. Em conjunto com ele, foi criado um *software* cliente, com autenticação de usuário. Ambos, quatro semanas após o início do trabalho, foram testados e aprovados. Estes resultados não foram aproveitados imediatamente, mas depois foram incorporados aos três aplicativos produzidos: Controlador GSE, SEI e Cliente.

- **I/O Digital e Analógico**

Para comando das placas de I/O digital e analógico são usadas rotinas do *Labview* que formam um conjunto chamado DAQ (*"Data Acquisition"* – Aquisição de Dados). As rotinas desenvolvidas definem endereços de multiplexação de telecomandos e telemetrias, além de executar a leitura/gravação de dados e o processamento necessário para isso.

Nesta etapa as tarefas (*"tasks"*), que são estruturas contendo informações sobre a lista de pinos e sua configuração (entrada/saída digital/analógica, opções de *trigger*, etc.), foram definidas usando um *software* auxiliar chamado *"Measurement & Automation"*, que permite todos os ajustes e mantém os dados no sistema. Em etapas posteriores, esta estratégia foi revista, com a criação de um VI específico para a configuração inicial, o que automatiza o processo no caso de reinstalação dos *softwares*.

- **Transmissão de Telecomandos / Leitura de Telemetrias**

Foi desenvolvido um VI que permite o envio de qualquer telecomando, dado seu número, independentemente do tipo (a MUX obedece dois tipos de telecomandos: liga/desliga e carregamento de memória – *"memory load"*). Ele não foi construído como interface gráfica final, mas como subrotina para utilização em outras, de mais alto nível. Apesar disso, ele permite o envio de todos os telecomandos do sistema, para teste de funcionamento, pois todos os VIs do *Labview*, mesmo os subVIs, podem ser abertos e executados como se fossem o programa principal.

O mesmo foi realizado para as telemetrias. Foi construído um VI intermediário capaz de fazer a leitura de todas as telemetrias previstas (com três possíveis tipos: nível, analógica ou digital serial), dado seu número.

Estes Vis, construídos usando como base as rotinas de acesso ao I/O digital e analógico, foram testados logo após sua finalização, através da verificação, com osciloscópio, dos valores das saídas, da leitura de valores conhecidos de tensão nos pinos analógicos/digitais e da leitura de formas de onda construídas através do gerador do MUX-GSE. Seu teste definitivo, entretanto, só ocorreu meses depois, após a montagem completa e integração das placas eletrônicas.

- **Interface do Sistema de Exibição de Imagens**

Para a interface do SEI (Figura 4.17), optou-se por um modo de visualização instantâneo da imagem do CCD (que é formada por uma única linha em cada banda[24]), mostrando a linha inteira comprimida na tela e uma visão ampliada, permitindo a análise de cada ponto através da cor e do valor. Existe ainda um outro modo, em que

[24] A MUX é equipada com um CCD quadrilinear (uma linha para cada banda) e as imagens da terra são formadas por diversas leituras sincronizadas com o movimento do satélite.

as linhas capturadas vão subindo pela tela, formando uma imagem. Isto permite a visualização dos alvos projetados pelo colimador e pelo simulador de cena.

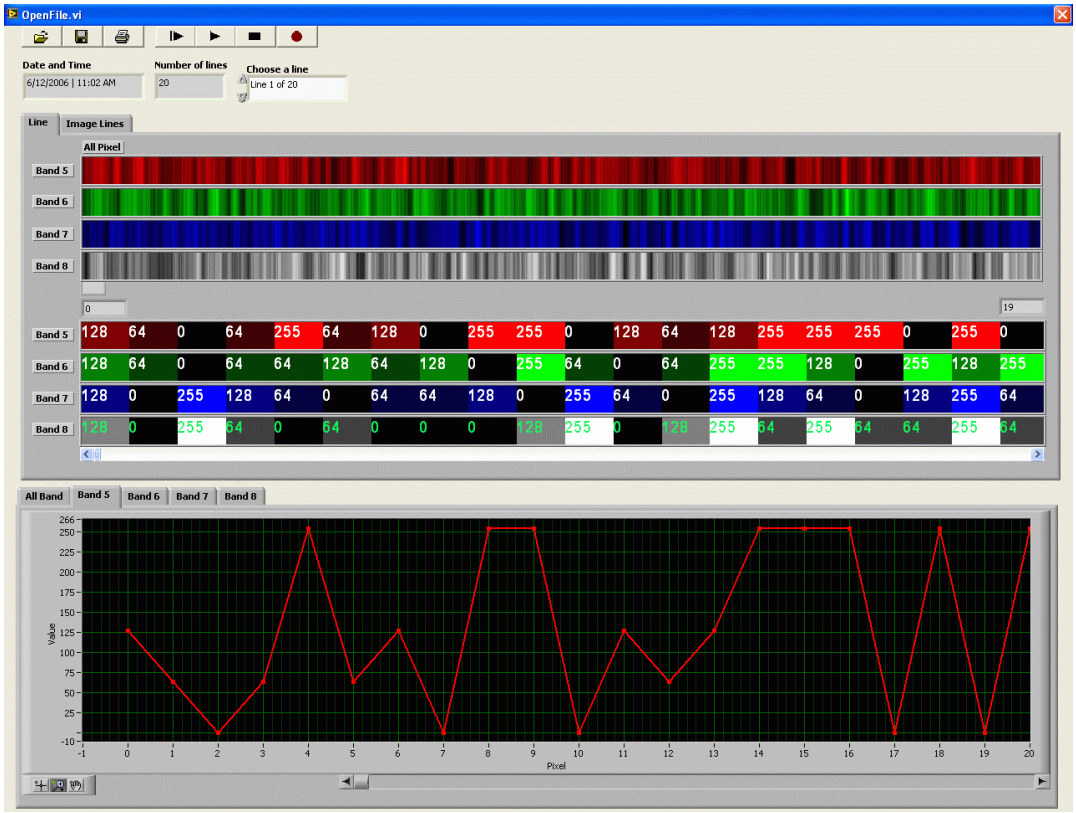


Figura 4.17: Tela do Sistema de Exibição de Imagens

A definição deste formato foi demorada. Contou com a opinião de diversas pessoas. A tela foi refeita diversas vezes até que se encontrasse uma configuração que conseguisse concentrar a grande maioria das opiniões. Ao final, o resultado foi aprovado por todo o time de desenvolvimento da MUX, além de satisfazer os pesquisadores do INPE que avaliaram o MUX-GSE.

- **Acesso ao I/O Digital de Alta Velocidade LVDS**

O acesso à placa de I/O LVDS é feito pelas mesmas funções HSDIO usadas para a placa de I/O digital do gerador de formas de onda digitais e do analisador

lógico. Assim, os resultados da prototipagem realizada para teste do HSDIO foram utilizados também para controle desta interface.

Com isso, foram produzidos VIs para captura dos dados vindos da câmera. Eles foram então utilizados para preencher a tela do SEI e também para gravação dos arquivos com imagens da câmera. Para que isso fosse possível, foram construídas também as rotinas de decodificação das informações. Assim, era possível simplesmente solicitar a captura em tempo real e receber os dados decodificados, prontos para uso em qualquer parte do programa.

Ao final desta etapa, todo este conjunto estava funcional. A velocidade de aquisição/exibição, entretanto, era muito baixa e não foi considerada satisfatória. Nas etapas seguintes foram realizados diversos testes com o intuito de acelerar esta atividade.

4.3.2 Conclusões

Ao final de apenas nove semanas, muitas rotinas haviam sido desenvolvidas, partindo praticamente do zero. Os painéis de todos os equipamentos estavam prontos e eram utilizados esporadicamente. As telas principais tanto do Controlador GSE quanto do SEI estavam funcionais, aguardando apenas a conclusão das funções para que todos os botões pudessem ser habilitados. O acesso a todas as placas PCI utilizadas já tinha sido realizado e rotinas intermediárias haviam sido desenvolvidas.

A etapa foi considerada um sucesso. Foi até mesmo cogitada a possibilidade de uma distribuição do *software*, porém até aquele momento, apenas um conjunto de *racks* do MUX-GSE estava montado e, portanto, com a equipe trabalhando utilizando-o constantemente, seu *software* não teria utilidade, pois a equipe da MUX não teria condições de aplicá-lo.

Após este período utilizando o *Labview*, todos já tinham adquirido prática razoável e detectado melhores soluções para muitas situações. Durante todo o desenvolvimento, diversas partes do código foram refeitas, refinando a técnica conforme a equipe melhorava sua habilidade. Mesmo assim, diversas rotinas, que não foram acessadas nas iterações finais da etapa, mereciam revisão. A equipe discutiu o caso e optou por deixar tudo como estava, já que o funcionamento era normal. Cada parte seria corrigida quando fosse acessada, aplicando o *design* contínuo e a refatoração no dia-a-dia, ao invés de buscar código mal construído em todos os diagramas.

4.4 Telas de Testes

O início da terceira etapa de desenvolvimento coincidiu com um grave problema do projeto: a empresa que fabricaria o colimador e os simuladores de cena apresentou um novo orçamento com valor muito acima das propostas anteriores e prazo de entrega totalmente inviável. Para não atrasar todo o projeto da MUX, a própria Opto Eletrônica se dispôs a desenvolver estes equipamentos no curto espaço de tempo disponível, tentando manter as datas de entrega.

Até por falta de alternativas, o INPE aceitou a proposta. Este fato não trouxe prejuízos aos testes da câmera, porque alguns componentes que seriam importados e fornecidos pelo INPE também não chegaram no prazo estipulado, atrasando a produção dos modelos da MUX que seriam validados pelo MUX-GSE. Assim, o projeto ganhou algum prazo extra além da data combinada.

Para o desenvolvimento de *software*, a conseqüência mais direta destas mudanças foi a ocupação de parte do tempo de trabalho dos engenheiros da equipe na seleção e negociação de equipamentos eletrônicos e mecatrônicos utilizados para controle do colimador e do

simulador de cena. Outro fator importante foi a impossibilidade de desenvolvimento das rotinas de testes ópticos.

Para aumentar ainda mais o trabalho, havia um marco do projeto se aproximando, exigindo a revisão de todos os documentos já produzidos e a apresentação de novas informações. Como era prevista a revisão do projeto dos equipamentos do banco óptico, ficou decidido que seriam apresentados todos os seus componentes, para aprovação imediata e liberação da compra o mais rapidamente possível. Com isso, dois programadores passaram a trabalhar na especificação e seleção destes itens. Os outros dois tiveram que dividir o tempo de trabalho com a organização dos relatórios dos circuitos elétricos do MUX-GSE (esquemas elétricos, diagramas de montagem, listas de componentes, etc.).

O outro objetivo imediato, este relacionado diretamente ao *software*, foi a produção de todas as interfaces de testes, ainda que de maneira provisória, para que fossem apresentadas nos documentos produzidos para este marco do projeto. Os dois programadores da equipe responsáveis pelas informações da eletrônica dedicaram-se em tempo parcial a esta tarefa. Sabia-se de antemão que muitas das telas seriam modificadas no futuro, quando fossem codificadas as rotinas de execução dos testes, porém se desejava ter uma idéia aproximada de como elas seriam e permitir uma avaliação oficial por parte do cliente.

Com isto, definiu-se pela execução de uma etapa curta, com duração de apenas três semanas e com a intenção única de produzir todas as telas preliminares de testes, com tempo ainda para incluí-las nos documentos de conclusão do marco do projeto. Esta tarefa foi conduzida por apenas dois membros da equipe que dividiam tempo com outras atividades.

Os demais se dedicaram à seleção dos componentes para construção do colimador e dos simuladores de cena e à revisão e redação dos documentos do projeto. Novamente, a estratégia adotada para documentação foi a produção apenas do que fosse estritamente necessário e a inclusão apenas das informações relevantes, evitando grande volume de dados

para revisão em etapas futuras. Mesmo assim, devido à necessidade de aprovação de todo o projeto por parte do INPE, foi necessário conteúdo razoavelmente extenso.

4.4.1 Resultados

Mesmo com o curto tempo disponível, foram realizadas algumas reuniões, com a presença da equipe completa, antes do início da criação das interfaces. Uma das primeiras decisões foi que todas as telas deveriam seguir o mesmo padrão, com diversas características comuns. Isto, além de melhorar a aparência geral do *software*, acelera o aprendizado de uso e, principalmente, facilita e acelera também a construção das telas e sua funcionalidade, pois é possível repetir partes da interface e do código de execução em muitas delas.

Os membros da equipe da MUX foram ouvidos antes do início dos trabalhos e após a criação das primeiras telas, para que suas opiniões pudessem ser incorporadas nas etapas iniciais, aumentando a probabilidade de satisfação do cliente e reduzindo a necessidade de modificações futuras.

Foi decidido que os testes seriam executados numa seqüência de passos, definida para cada um deles. Seria colocada uma barra de controles na parte superior das telas, com comandos básicos de abertura e gravação de arquivos, impressão de resultados, além de um conjunto de botões imitando um aparelho de DVD, com funções de realização do próximo passo, de todos os passos restantes, parada imediata, retorno à condição inicial e execução completa seguida de gravação dos resultados. Logo abaixo da barra de controles, foi posicionada uma caixa de instruções, responsável por exibir o próximo passo.

No total, foram produzidas trinta telas para testes. Algumas delas são mostradas na Figura 4.18 abaixo.

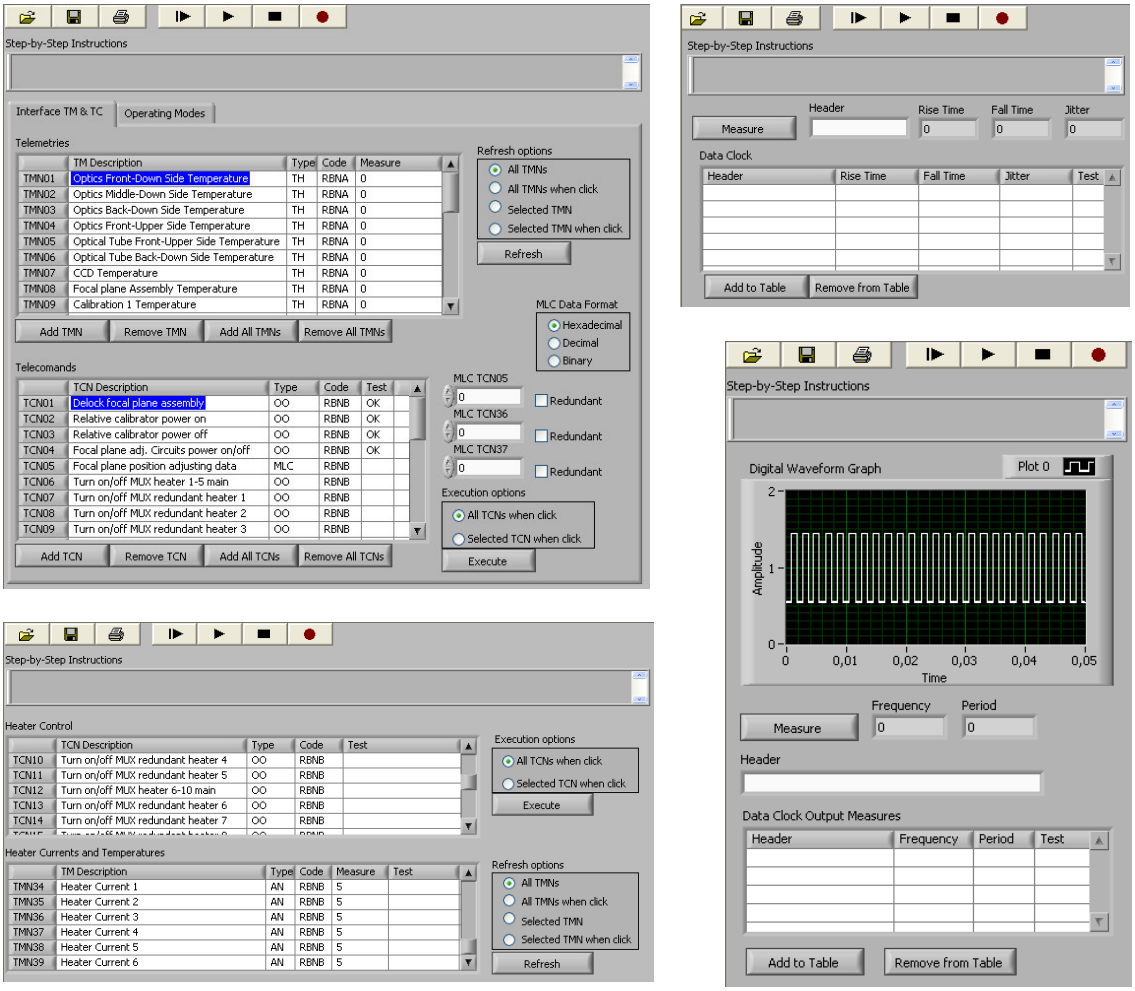


Figura 4.18: Telas de testes automáticos: interfaces TM&TC (sup. esq.), estabilidade do *clock* de dados (sup. dir.), controle de temperatura (inf. esq.) e taxa de saída de dados (inf. dir.)

4.4.2 Conclusões

Neste curtíssimo espaço de tempo, apenas três semanas, foram construídas trinta telas de testes, por apenas dois programadores trabalhando em regime parcial. Ao mesmo tempo foram redigidos oito relatórios de projeto e setenta e seis outros documentos relativos às placas eletrônicas.

O volume de documentos entregue foi elevado para um projeto ágil; entretanto, este problema era esperado, uma vez que a própria licitação da MUX já trazia esta exigência. A

equipe de desenvolvimento tentou minimizar as dificuldades, limitando os relatórios às informações mais relevantes, sem prejudicar a compreensão das idéias apresentadas.

Nesta etapa diversas situações inesperadas mudaram drasticamente os rumos do projeto. Isto comprova que a aplicação do método ágil foi a melhor escolha. Um planejamento rígido no início dos trabalhos certamente teria sido invalidado por estas mudanças, ou seja, teria havido desperdício na determinação de um cronograma completo e mais ainda na sua correção.

A documentação entregue ao INPE foi aprovada com a solicitação de poucas modificações que não impactaram no projeto e foram executadas rapidamente. As telas foram aceitas, porém seu desenvolvimento em tempo tão curto prejudicou o planejamento da execução dos testes. Muitas das telas tiveram que ser quase que totalmente refeitas no futuro, devido a características dos procedimentos de teste não levadas em conta nesta etapa. Isto mostra que a tentativa de gerar grande quantidade de código em curto espaço de tempo reduz a qualidade e acaba obrigando a revisão do produto.

4.5 Testes Eletrônicos e Mecânicos

Concluída a seleção de componentes eletrônicos e mecatrônicos para compor os equipamentos do banco óptico e aprovadas as telas de testes, restava agora iniciar uma fase de codificação das rotinas de testes automáticos. A equipe empenhou-se nesta tarefa e também na otimização da rotina de aquisição de imagens, já construída, mas que ainda não tinha alcançado performance aceitável.

Foram eleitos para esta etapa apenas os testes eletrônicos e mecânicos. Os equipamentos ópticos não estariam disponíveis tão cedo, portanto os procedimentos dependentes deles foram ignorados temporariamente. Mesmo as rotinas construídas não

poderiam ser testadas imediatamente, porém aguardava-se pelo término da montagem de um conjunto de placas eletrônicas em breve, o que permitiria a execução do sistema *hardware/software* completo pela primeira vez.

Quando a montagem foi concluída, as placas, em total de treze, foram testadas uma a uma, implicando em tarefa longa que exigiu o trabalho de um membro da equipe em tempo integral e a ajuda de outro em regime parcial por aproximadamente seis semanas, considerando teste, identificação e correção de falhas. A quantidade de defeitos foi grande em relação a projetos anteriores, decorrendo da pouca experiência dos projetistas das placas, do volume de trabalho e do curto tempo disponível. Durante todo este tempo não foi possível aferir o funcionamento das rotinas construídas, trabalho que ficou para a etapa seguinte.

Com tudo isto, o desenvolvimento das rotinas de teste foi conduzido por dois engenheiros em tempo parcial. Um deles participou da depuração das placas eletrônicas enquanto o outro ajudou na evolução da aquisição de imagens. Isto alongou um pouco a etapa, que teve duração de seis semanas. Isto não foi considerado um problema, pois era fundamental ter um conjunto eletrônico à disposição para execução de testes em condições reais. A otimização da aquisição de imagens, também importantíssima, tomou muito tempo, não se restringindo a esta etapa e, por isso, será apresentada à parte na Seção 4.6.

Além das rotinas de teste automáticos, foram construídos VIs básicos para gravação, exibição e gerenciamento de arquivos de *log*, que já foram inseridos em alguns procedimentos de testes e na tela principal do Controlador GSE, para que pudessem ser testados. Depois disso, eles poderiam ser utilizados em qualquer parte do programa, facilitando a manutenção das informações de *log*, previstas nos requisitos do sistema.

Também foram produzidos VIs para abertura e gravação de resultados dos testes. Eles foram criados de maneira genérica, para que pudessem ser aplicados em todos os

procedimentos que fossem implementados. Isto exigiu a definição de um formato de arquivo e a seleção das informações relevantes, que precisariam ser armazenadas.

4.5.1 Resultados

Ao término de seis semanas, estavam concluídas as rotinas de execução de doze testes (Tabela 4.3) de parâmetros elétricos e mecânicos. A maioria era bastante simples, envolvendo somente a medida de parâmetros utilizando os instrumentos disponíveis, por isso o desenvolvimento ocorreu sem problemas. O teste das interfaces TM&TC e modos de operação, porém, é muito complexo e a equipe não considerou satisfatória a solução executada. Ela serviria temporariamente, para realizar o primeiro teste de integração *hardware/software*, porém depois precisaria de uma criteriosa revisão.

Tabela 4.3 – Rotinas de testes desenvolvidas na quarta etapa

Rotinas de testes codificadas na quarta etapa
Clock de dados / *jitter*
Formato de dados
Dimensões
Aterramento
Estado elétrico inicial
Tempo de integração
Massa
Taxa de saída de dados
Potência consumida
Tempo de amostragem
Pontos de teste
Interface TM&TC/Modos de operação

Os demais testes foram aprovados, ao menos inicialmente. Eles teriam que ser executados em conjunto com o hardware real, pois a verificação simulada sempre pode deixar passar alguma inconsistência, o que efetivamente ocorreu em alguns casos. Muitos deles passariam ainda por revisões, principalmente relacionadas à interação com o usuário, pois algumas falhas conceituais, envolvendo também a tela de interface, foram detectadas somente

depois e todas as rotinas haviam sido construídas seguindo o mesmo padrão. Esta correção será apresentada na Seção 4.10.

As rotinas de *log* desenvolvidas foram consideradas suficientes para as necessidades do projeto. Bastava agora aplicá-las em todos os VIs responsáveis por gerenciamento de erros, execução de testes e todos os demais eventos a serem registrados.

A abertura e a gravação de resultados de exames funcionaram perfeitamente. Os VIs criados eram capazes de satisfazer as necessidades de todas as rotinas de testes criadas até então e, possivelmente, também as demais. Somente para a gravação e abertura de imagens da MUX pelo SEI é que foi necessário desenvolver funções diferenciadas. Os testes já concluídos tiveram ativadas as funções de gravação e abertura de arquivos.

4.5.2 Conclusões

Esta etapa do projeto foi produtiva em termos de software, mas, principalmente, foi fundamental para a continuidade dos trabalhos. A conclusão de um conjunto eletrônico do MUX-GSE era aguardada ansiosamente para possibilitar a verificação de partes já concluídas.

As rotinas construídas sofreram diversas modificações nas etapas futuras, em especial a de testes da interface TM&TC e modos de operação, que não havia agradado aos membros da equipe. Estas mudanças foram frutos de problemas que ocorreram principalmente por falha de comunicação. Se os testes tivessem sido discutidos antes de sua codificação, certamente melhores soluções teriam surgido.

Entretanto, como a equipe estava muito dividida, com todos os engenheiros atarefados e preocupados com tarefas muito distintas, o nível de interação havia diminuído bastante naqueles dias. Os trabalhos teriam sido mais proveitosos se tivessem ocorrido reuniões para

discussão do andamento das atividades, pois assim provavelmente estas falhas teriam sido detectadas antes.

4.6 Otimização da Aquisição de Imagens

A aquisição de imagens foi a função do MUX-GSE que mais proporcionou desafios à equipe de desenvolvimento. Os dados fornecidos pela câmera em dois canais seriais padrão LVDS, são transmitidos a uma taxa de 65,14 Mbit/s. A placa selecionada para recepção destas informações tem capacidade para ler e armazenar dados em até 100 Mbit/s. Sua capacidade de memória é de 2 Mbit por canal.

As rotinas desenvolvidas, em um primeiro momento, foram capazes de armazenar até 1 s de dados, quando a memória da placa de aquisição ficava totalmente preenchida e era necessário parar a operação. A exibição na tela chegava a 3 fps, ignorando os quadros intermediários, ou seja, entre duas imagens exibidas, 111 eram ignoradas.

Outro problema grave era o tempo gasto para detecção do sinal de sincronismo, que é um conjunto de dados que informa a posição de início da linha. A primeira rotina construída demorava em torno de 273 ms, e somente após a aquisição de 92 quadros, em média, conseguia identificar o sincronismo.

Para tentar melhorar esta condição, o que era imprescindível, um engenheiro foi selecionado para trabalhar exclusivamente nesta rotina, contando com outro, que em parte do tempo o ajudava programando em par ou discutindo problemas e soluções. Foram testadas idéias variadas através de prototipagem. Todos os VIs envolvidos no processo foram avaliados quanto ao tempo de execução, para determinar as tarefas mais críticas, que mereciam maior atenção.

Por fim, foi cogitada a possibilidade de desenvolvimento de uma placa eletrônica contendo um CPLD[25] para encadeamento dos dados, transformando-os em um conjunto paralelo com tamanho de 1 byte. Nesta condição, a placa de aquisição utilizada consegue trabalhar melhor, permitindo aquisição de bytes na mesma taxa que recebe bits.

4.6.1 Resultados

Após a construção e testes de diversos protótipos, em um trabalho total de mais de três meses, a única opção satisfatória que não dependia de modificações no *hardware* foi a criação de uma Biblioteca de Vínculo Dinâmico (*"Dinamyc Link Library"* – DLL) em C++ e seu acesso através do *Labview*. A evolução na performance foi significativa. A Tabela 4.4 mostra alguns dados comprovando este feito.

Tabela 4.4 – Evolução na aquisição de imagens da MUX

Atividade	Antes	Depois
Identificação do sinal de sincronismo (média)	273s 92 quadros	40ms 13,4 quadros
Armazenamento máximo (sem perda de informações)	1 s 336 quadros	Ilimitado
Taxa de exibição	3 fps	10 fps
Quadros ignorados na exibição	111 quadros	32,6 quadros

Com esta nova condição, a utilização de uma placa para organização dos bits em dados paralelos foi descartada momentaneamente. Existe a possibilidade de implementar esta solução no futuro, para aumentar ainda mais a capacidade do sistema, porém, como já é possível o armazenamento contínuo dos dados, sem perda de informações, talvez esta idéia seja abandonada definitivamente.

[25] CPLD (*"Complex Programable Logic Device"* – Dispositivo Lógico Programável Complexo) é um dispositivo semicondutor que permite aos usuários a definição de suas funcionalidades, sendo muito utilizado para o processamento de informações digitais.

A única limitação atual, que não chega a ser um grande inconveniente, é que o SEI pára a exibição de imagens durante a transmissão de informações via LAN. Como a quantidade de dados solicitada durante a execução normal dos testes nunca é muito grande, a parada máxima é menor que 1 s, o que não prejudica em nada a utilização do MUX-GSE.

4.6.2 Conclusões

Após a otimização, que foi uma tarefa árdua, que exigiu muitos testes e estudo aprofundado sobre o funcionamento do *Labview*, a execução das rotinas de aquisição e exibição de imagens passou a uma condição muito boa. A apresentação de dados na tela é limitada somente pela capacidade de atualização do monitor e da placa de vídeo. Com a capacidade de armazenamento ininterrupta, até o preenchimento de todo o disco rígido, o desempenho previsto foi atingido e as necessidades para realização dos testes foram superadas.

4.7 Testes Integrados Hardware/Software

Após a eliminação de falhas das placas eletrônicas, iniciou-se a revisão em seu projeto e o desenvolvimento de um novo *layout*. O conjunto já montado, entretanto, foi mantido temporariamente para permitir a avaliação do funcionamento das rotinas de teste eletrônicos do MUX-GSE, em especial a de interfaces TM&TC. Esta rotina é necessária em praticamente todos os demais testes, pois os telecomandos são utilizados na configuração dos modos de funcionamento da câmera, sendo imprescindíveis para o início da aquisição de imagens. As telemetrias são utilizadas constantemente para monitoramento da MUX, assegurando seu perfeito funcionamento e detectando situações de emergência.

Em primeiro lugar, foi testado cada um dos telecomandos, certificando-se da correta resposta das interfaces. Os pinos correspondentes foram avaliados através de osciloscópio. Em seguida, foram testadas as telemetrias. Os sinais foram injetados nas entradas das interfaces através de fontes de alimentação e o valor da telemetria observado na tela do software.

Após esta avaliação, passou-se à utilização da tela de testes de TM&TC, realizando rotinas automáticas completas. Sabia-se de antemão que haveria problemas na execução (Seção 4.5). Eles implicaram na modificação do modo de interação do usuário com o *software*. Estas mudanças foram relativamente sérias, pois exigiram alterações no funcionamento de todas as telas de testes já construídas, que foram realizadas posteriormente, na etapa apresentada na Seção 4.10.

Apenas as alterações mais urgentes foram implementadas de imediato para permitir a execução dos testes de sistema, deixando um planejamento mais aprofundado do *software* para depois da aprovação de todas as placas, de modo que a nova versão da eletrônica pudesse ser encaminhada para produção sem perda de tempo.

4.7.1 Resultados

Todas as placas foram aprovadas após algumas modificações simples no software: erros de endereçamento, modificações nos valores de constantes de calibração, etc. O grande problema encontrado foi a falha no funcionamento das interfaces, exigindo uma nova etapa para modificação das demais rotinas de testes já construídas. Este problema foi contornado de maneira temporária para não atrasar a fabricação do novo lote de placas.

4.7.2 Conclusões

Esta foi uma etapa um tanto quanto simples, já que o *hardware* já havia sido previamente testado. Sua condução foi facilitada também pelo fato dos programadores serem engenheiros eletricistas, com visão global do sistema, portanto mais capacitados para compreenderem e detectarem causas de erros.

Esta etapa teve importância por garantir o funcionamento dos circuitos eletrônicos propostos e por permitir observar melhor o funcionamento da interface homem-máquina, facilitando a determinação das alterações que teriam que ser realizadas nas telas dos testes.

4.8 Rotinas Não Previstas de Suporte à MUX

Algumas rotinas não originalmente previstas foram solicitadas pelos engenheiros da MUX para permitir a execução de testes de algumas placas de maneira isolada, facilitando a identificação de problemas. Basicamente, foi solicitada a geração de todos os sinais necessários para funcionamento CCD. Para esta tarefa não interferir nos testes da eletrônica, apenas um desenvolvedor foi designado.

Pela complexidade dos dados, ao invés de utilizar diretamente o painel do gerador virtual de formas de onda digitais, optou-se por construir um VI específico para este teste, com a definição dos sinais necessários através de um algoritmo implementado no próprio diagrama do *software*.

O responsável pelo desenvolvimento ouviu constantemente os engenheiros da MUX, tanto para compreender adequadamente os algoritmos necessários, tanto para definir a seqüência mais útil de produção.

Como a necessidade era urgente, não foi dada atenção à interface gráfica, que foi construída de maneira simples, com todos os controles à mostra, mesmo os indicadores de erro e o mostrador da tabela de dados gerada. Estas informações geralmente não são apresentadas ao usuário, pois não têm utilidade em situação normal. Neste caso, entretanto, toda a importância foi dada ao funcionamento do sistema em curto espaço de tempo.

4.8.1 Resultados

As rotinas necessárias foram construídas e testadas em menos de duas semanas, mesmo com a divisão do tempo de trabalho do programador com outras atividades. Apesar do algoritmo não ser muito simples, a eliminação do tempo consumido na construção de interfaces amigáveis encurtou bastante o trabalho de desenvolvimento.

As formas de onda produzidas foram avaliadas através de um analisador lógico e, após algumas correções, foram aprovadas. Os sinais do MUX-GSE foram aplicados à placa responsável por comandar o CCD e todos os pinos relevantes foram avaliados através de osciloscópio e analisador lógico.

Posteriormente, estas rotinas foram aplicadas no teste do primeiro modelo do CCD da MUX fornecido (Figura 4.19). Este componente, destinado a testes mecânicos e eletrônicos, é idêntico ao que equipará o modelo de vôo da câmera, porém sem o mesmo nível de qualidade. Este teste foi acompanhado pelos especialistas do INPE e foi a primeira vez que um CCD deste tipo foi utilizado no Brasil. Algumas falhas foram detectadas na eletrônica da MUX, porém foram rapidamente corrigidas e a imagem do CCD pôde ser observada através da tela do osciloscópio.

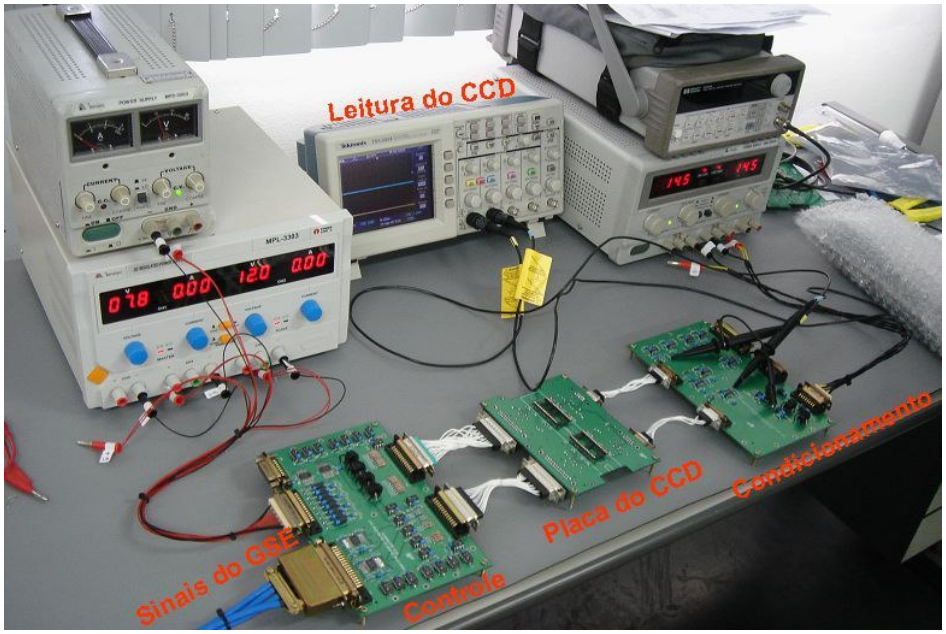


Figura 4.19: Montagem para teste do CCD utilizando sinais do MUX-GSE

4.8.2 Conclusões

Apesar do pouco tempo disponível, as rotinas construídas permitiram a execução dos testes com a eletrônica da MUX e seu CCD. Os resultados agradaram à equipe de desenvolvimento da câmera e aos especialistas do INPE que acompanharam os procedimentos.

A interface gráfica pobre não foi problema, pois apesar de pouco apresentável, é muito prática e permite a seleção dos parâmetros relevantes, como a porta utilizada para saída de cada um dos sinais.

A flexibilidade do cronograma permitiu a construção destas rotinas sem prejudicar o ritmo de trabalho da equipe, pois o restante do trabalho foi acomodado para aceitar esta mudança inesperada.

4.9 Distribuição da Primeira Versão

Com a finalização das rotinas de testes elétricos e mecânicos, optou-se pelo congelamento de uma versão inicial, liberando-a para uso enquanto continuava o desenvolvimento normal das novas funções e o refinamento das já construídas. A distribuição seguiu os procedimentos do sistema de gestão de qualidade adotada pela Opto Eletrônica, que já haviam sido aprovados pelo INPE. Foram realizados teste de unidade e de integração e a validação dos requisitos referentes às partes funcionais do software, discriminadas na Tabela 4.5.

Tabela 4.5 – Funções validadas na primeira versão distribuída

Rotinas validadas na versão 1.0	
Telas principais	Controlador GSE SEI
Telas de equipamentos	Osciloscópio (fornecida pelo fabricante) Multímetro (fornecida pelo fabricante) Gerador de formas de onda digitais Analisador lógico Fonte de alimentação Freqüencímetro
Rotinas de teste	Estabilidade do *clock* de dados Codificação dos dados e formato de transmissão Dimensões Contato e aterramento Estado elétrico inicial Tempo de integração Massa Taxa de saída de dados Consumo de potência Período de amostragem Interfaces TM&TC/Modos de operação
Funções adicionais	Aquisição de imagens Comunicação via TCP/IP Gravação e abertura de resultados de teste Gravação, exibição e gerenciamento de *log*

Ressalta-se que esta foi uma validação parcial, pois a completa só pode ser realizada após o cumprimento de todos os requisitos do sistema. Nesta etapa, nenhum dos testes ópticos havia sido produzido, portanto estes requisitos não estavam satisfeitos. Sabia-se também que muitas partes do software seriam modificadas no futuro, mas isso não representava problema, pois nas próximas distribuições toda a validação seria refeita.

4.10 Depuração e Otimização das Rotinas de Teste

Esta foi uma etapa longa, cujos objetivos eram tornar as rotinas de teste já desenvolvidas as mais automatizadas possíveis, evitando intervenções não necessárias do usuário. Ele poderia observar a execução passo a passo, mas pressionando o botão *"play"* todo o procedimento seria executado e somente as entradas necessárias seriam solicitadas.

O problema é que as funções criadas inicialmente não levaram em conta todas as informações relacionadas a alguns testes e o diagrama de execução continha falhas que podiam acontecer dependendo das atitudes tomadas pelo usuário. Para correção do problema, buscou-se a definição de um método padronizado de funcionamento dos VIs que pudesse ser aplicado em todos os testes, do mais simples ao mais complexo.

Partindo-se da idéia que algo capaz de solucionar o problema mais difícil é suficiente para resolver todos os casos mais fáceis, a equipe concentrou esforços no funcionamento do teste da interface TM&TC e modos de operação. Esta tarefa foi difícil, pois a configuração da interface no *Labview*, levando em conta todas as possíveis condições e intervenções do usuário, é lenta. A configuração utilizando nós de propriedade (*"property nodes"*) exigiu muitas mudanças nos diagramas. Isso leva tempo, pois é necessário ficar arrastando os itens e reorganizando-os.

Esta atividade foi ainda prejudicada por algumas situações inesperadas. O desligamento da empresa de um dos engenheiros da MUX obrigou o deslocamento de um dos membros da equipe do MUX-GSE para o desenvolvimento das rotinas de processamento de sinais da câmera. Assim, durante oito semanas, apenas três pessoas tiveram que prosseguir com os trabalhos, até que um novo engenheiro foi contratado. O novo membro teve uma fase de integração, que será mostrada na Seção 4.12.

Além disso, nesta etapa de otimização, quase toda a programação era feita utilizando o *hardware* do MUX-GSE para avaliação imediata do funcionamento. Isso provocava interrupções freqüentes no trabalho, pois os circuitos eletrônicos da MUX estavam passando por uma fase de testes intensos, para solução de falhas no processamento do sinal de vídeo. Estes testes dependiam do MUX-GSE e, assim, as duas equipes tinham que dividir a utilização do sistema.

Os engenheiros do MUX-GSE também tiveram outras atividades, como a otimização das rotinas de aquisição de imagens, que ainda estava em curso, a montagem e testes da segunda versão das placas eletrônicas e a redação das especificações e procedimentos de todos os testes previstos.

Tantos inconvenientes fizeram com que esta etapa se estendesse por quase vinte semanas. Isso não foi um grande problema porque os equipamentos comprados para produção do colimador e dos simuladores de cena atrasaram mais de três meses, devido a diversos problemas externos, não relacionados à atuação da equipe. Ou seja, a correção das rotinas de teste, apesar de demorada, ocorreu em prazo adequado ao projeto e não prejudicou o desenvolvimento da MUX.

4.10.1 Resultados

Após muita discussão e a tentativa de várias soluções distintas, optou-se por um sistema em que o diagrama das rotinas de teste possui dois *loops* executados simultaneamente (Figura 4.20). O de baixo faz a verificação dos comandos do usuário e encadeia as solicitações realizadas em uma fila, enquanto que o superior faz a leitura da fila e executa as ações solicitadas. Nenhuma intervenção do usuário é aceita diretamente neste *loop*, à exceção do cancelamento das operações.

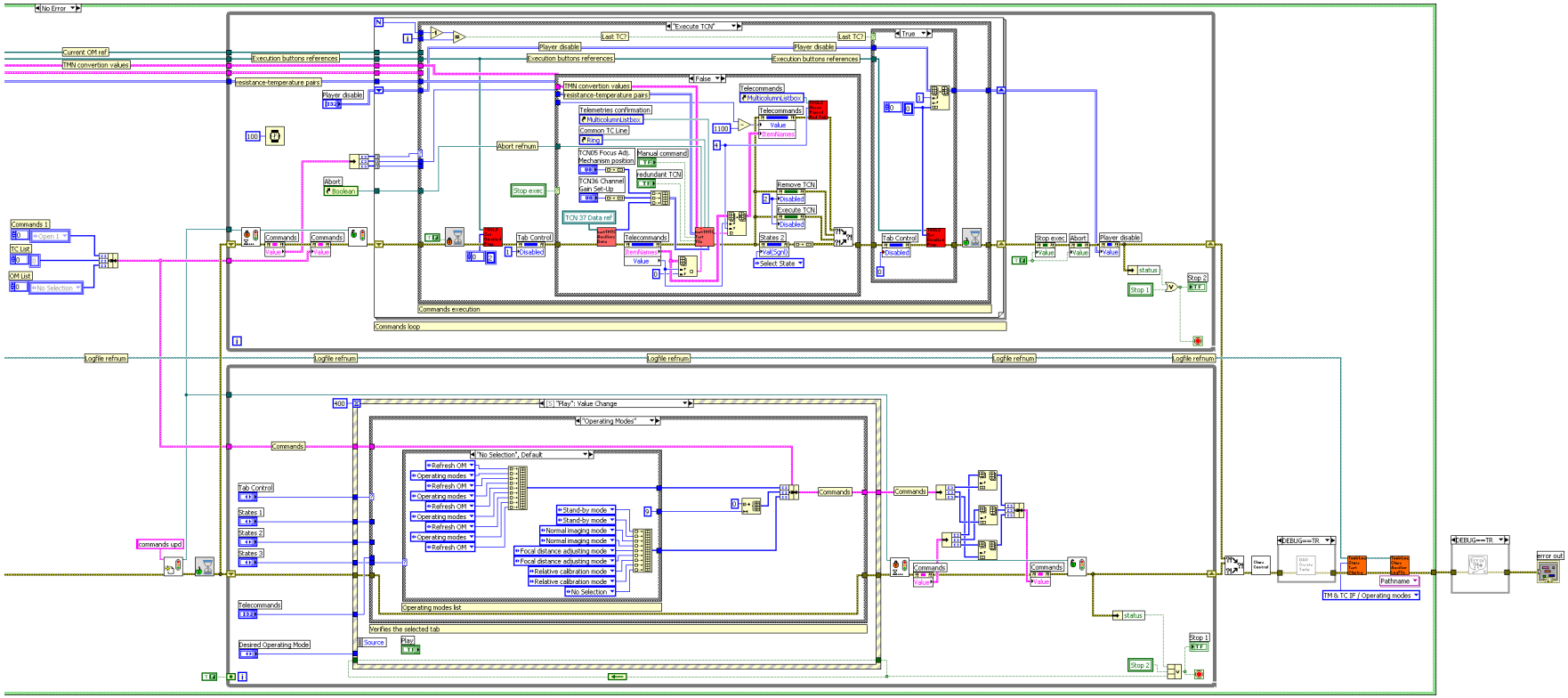


Figura 4.20: Parte do diagrama de teste da interface TM&TC e modos de operação

Para garantir a integridade dos dados na fila, foi utilizado um semáforo (*"semaphore"*), componente do *Labview* que permite bloquear o acesso a um determinado dado enquanto se realiza a gravação de informações. Sem isso, seria possível que um dado fosse inserido na fila pelo *loop* inferior ao mesmo tempo em que uma leitura das informações estivesse ocorrendo no superior. Esta condição teria resultados imprevisíveis.

A interface foi planejada de modo que os testes podem ser executados na ordem preferida, ou seja, alguns podem ser ignorados se o usuário o desejar. Para isso, a cada passo, as condições iniciais necessárias são verificadas e ajustadas, sendo as condições finais avaliadas para determinar se o resultado foi de falha ou sucesso. Para melhorar a visualização das informações foi acrescentada uma aba às duas já existentes, separando o teste de telemetrias do de telecomandos.

Após a conclusão da rotina de testes da interface de TM&TC e modos de operação, o mesmo padrão de diagrama foi aplicado a todas as demais já construídas, que eram mais simples e puderam seguir o mesmo esquema de funcionamento.

4.10.2 Conclusões

Foram mais de quatro meses até conseguir definir um padrão de funcionamento para as rotinas de teste e deixar todas as já existentes com o mesmo estilo. Ao final da etapa, entretanto, todos funcionavam perfeitamente. O longo tempo gasto nesta operação pode ser explicado pelos diversos imprevistos ocorridos. A utilização de um método ágil ajudou a equipe a absorver todos estes problemas sem permitir que o *software* se tornasse fator crítico no andamento do projeto.

Esta flexibilidade é importante até mesmo para quando o imprevisto é devido a uma falha na implementação da própria metodologia, que foi o que ocorreu na construção da primeira versão das rotinas de teste, como mostrado na Seção 4.5.2.

4.11 Teste de Conceito da Medida de MTF

O teste da MTF[26] é uma das funções mais importantes para determinação da qualidade do sistema óptico produzido. Além de componentes ópticos de qualidade, exige rotinas de processamento no domínio da freqüência. Existem vários métodos de medida, com vantagens e desvantagens. No MUX-GSE optou-se pela execução de uma varredura do CCD com uma fenda luminosa e o processamento matemático dos valores obtidos no ponto de medida.

O método selecionado é completo, permitindo a obtenção de toda a curva da MTF. É descrito em vários trabalhos científicos, porém pode apresentar diversas dificuldades técnicas. Para evitar problemas, mesmo antes da chegada dos componentes ópticos e mecatrônicos para construção do colimador, optou-se pela realização de testes de processamento, para verificar o funcionamento do algoritmo proposto.

4.11.1 Resultados

Foi construído um VI simplificado (Figura 4.21) com as operações matemáticas que realizam o cálculo do MTF a partir de uma linha de imagem. A eficácia do cálculo foi comprovada através da entrada de valores conhecidos e a comparação da saída obtida com a teórica. Um dos testes foi a aplicação de uma linha com contraste perfeito, ou seja, variação

[26] MTF (*"Modulation Transfer Function"* – Função de Transferência de Modulação) é a curva que descreve a resposta espectral de um sistema óptico a uma imagem de entrada. É uma análise no domínio da freqüência que permite determinar o nível de contraste que o sistema óptico é capaz de reconhecer.

do máximo ao mínimo valor de um ponto ao seguinte. Como esperado, a curva obtida foi o máximo MTF permitido pelo CCD, determinado pelo tamanho dos pontos.

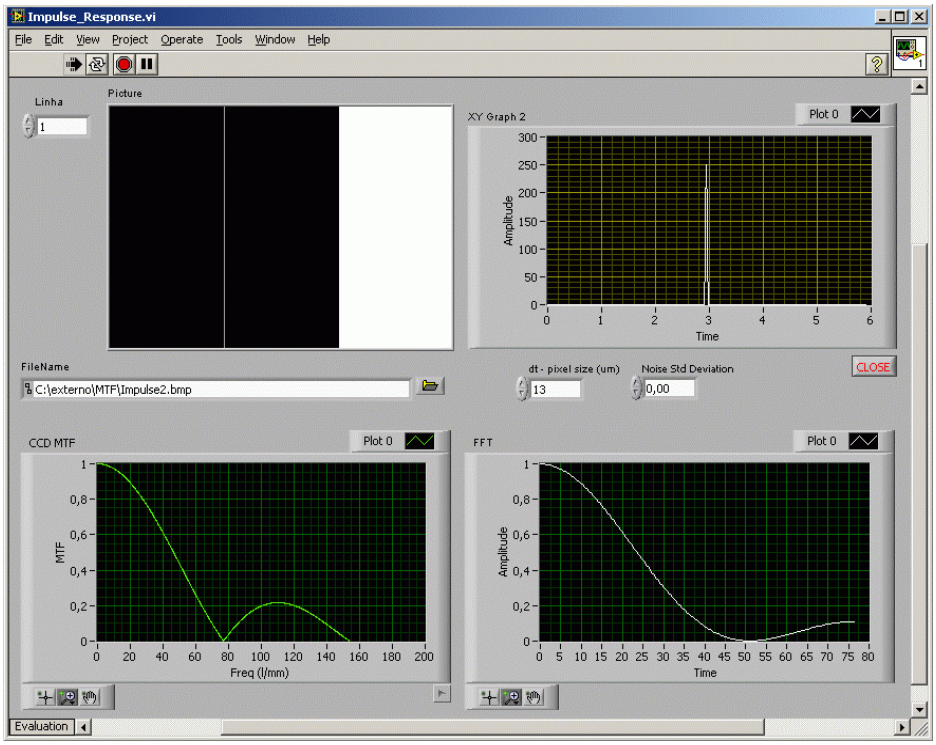


Figura 4.21: Interface do protótipo de cálculo de MTF

Por se tratar apenas de um protótipo, não foi gasto muito tempo na confecção da interface gráfica. Foram utilizados componentes padrão para exibição das informações de modo que se pudesse avaliar qualitativamente os resultados. A avaliação quantitativa foi feita através da observação dos valores nas variáveis internas, usando funções de *"debug"*.

4.11.2 Conclusões

Os resultados obtidos foram condizentes com a teoria, portanto o algoritmo foi considerado aprovado. Restava agora a chegada dos equipamentos ópticos e mecatrônicos para validar o colimador e o sistema completo de medição.

4.12 Auto-teste

Durante o trabalho, por volta da septuagésima quinta semana, um novo engenheiro foi integrado à equipe de desenvolvimento do MUX-GSE para substituir o que havia sido deslocado para a equipe da MUX. Para colocá-lo rapidamente a par do projeto, após a leitura dos principais documentos já produzidos, ele foi escalado para testes de *hardware* e implementação de *software* da rotina completa de auto-teste, que foi solicitada e definida após a revisão do PDR.

As placas eletrônicas já haviam sido projetadas e estavam sendo montadas. Restava, portanto, integrá-las ao sistema, testar seu funcionamento e criar as rotinas de *software* capazes de realizar todas as operações planejadas.

Para facilitar o entrosamento, o novo funcionário foi acompanhado durante os testes das placas, para que pudesse tirar dúvidas e conhecer o sistema mais rapidamente. Na área de *software* houve supervisão no início dos trabalhos, mas não contínua. As rotinas a serem construídas eram discutidas em grupo durante o planejamento e o código produzido era revisado. Todos ajudavam o tempo todo com as respostas requeridas.

Rapidamente, o engenheiro se adaptou e passou a trabalhar em bom ritmo. O desenvolvimento do auto-teste foi um pouco prejudicado pela necessidade dos *racks* do MUX-GSE, já utilizado também para avaliação das modificações das rotinas de teste e também para depuração dos circuitos eletrônicos da MUX. Todos tiveram que se esforçar para deixar os equipamentos livres para os demais, fazendo modificações no *software* utilizando seus próprios micros e testando somente quando fosse absolutamente necessário.

O auto-teste é realmente extenso, pois exige a medição de todos os cabos de conexão do MUX-GSE, ou seja, é uma rotina que se repete várias vezes com configuração diferente de

portas para endereçar todos os pinos de ligação. Além disso, é normal que o trabalho seja mais lento durante os primeiros meses em um novo emprego, até que haja completa integração com os colegas, o ambiente e métodos de trabalho e o projeto como um todo.

Além disso, houve algum atraso extra devido à necessidade de mudança do departamento de P&D para suas novas instalações, com todas as características necessárias para produção dos modelos de vôo da MUX. Apesar das melhores condições oferecidas, a mudança prejudicou o trabalho por aproximadamente duas semanas, enquanto todos os laboratórios eram reorganizados.

4.12.1 Resultados

Tanto a eletrônica quanto as rotinas de auto-teste (Figura 4.22) foram concluídas e passaram por uma bateria de testes, com a utilização de fonte controlada, multímetro e osciloscópio, para assegurar a detecção de situações de perigo e a aprovação do sistema quando este apresentar boas condições de uso.

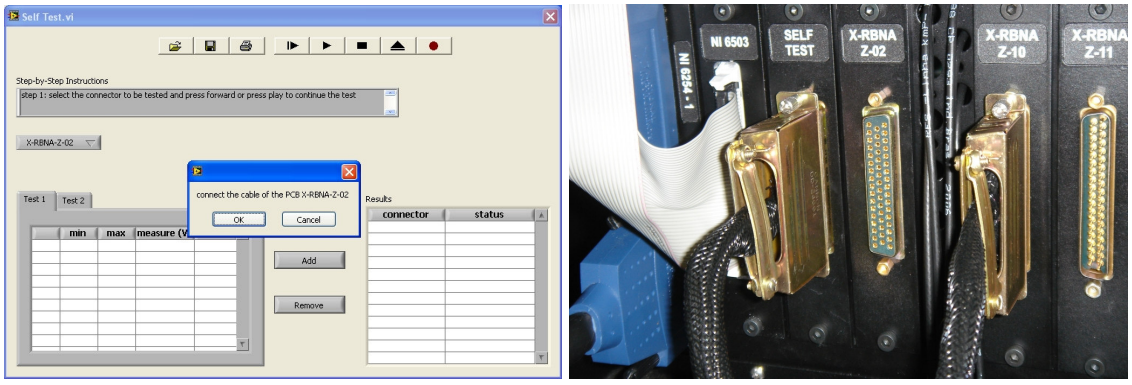


Figura 4.22: Tela (esq.) e cabo (dir.) de auto-teste

4.12.2 Conclusões

As rotinas de auto-teste são a comprovação da integração do novo membro. Isto mostra que a documentação do projeto, mesmo sendo econômica, é suficiente para transmitir o conhecimento necessário a quem se junta à equipe. É claro que a comunicação direta também tem papel importante. É uma ferramenta que acelera bastante o processo e não se pode negar que parte do conhecimento é mantida nas mentes dos engenheiros. Apesar disto ser um problema para muitos gerentes, está perfeitamente de acordo com a idéia de valorização dos indivíduos nos métodos ágeis.

4.13 Controle do Colimador e do Simulador de Cena

Quando a primeira parte dos componentes importados do colimador e dos simuladores de cena foi entregue, rapidamente foi executada uma bateria de testes e teve início a programação de rotinas para seu controle via *software*. As telas foram planejadas e discutidas, mesmo não sendo ainda a versão definitiva, já que faltavam as fontes de iluminação, que também comporiam estes instrumentos.

Um dos engenheiros cuidou da tela do colimador principal (Figura 4.23) e as rotinas de acesso ao controlador das quatro plataformas de precisão automáticas que compõem o equipamento, enquanto outro desenvolveu a tela do simulador de cena (Figura 4.24), com as rotinas de comando das duas plataformas deste instrumento. Ele também criou os VIs para aquisição da temperatura (usados no colimador) e da pressão (usados em ambos). O desenvolvimento foi rápido e em apenas duas semanas as interfaces estavam funcionais,

aguardando a chegada da segunda remessa de componentes para que pudessem ser completadas.

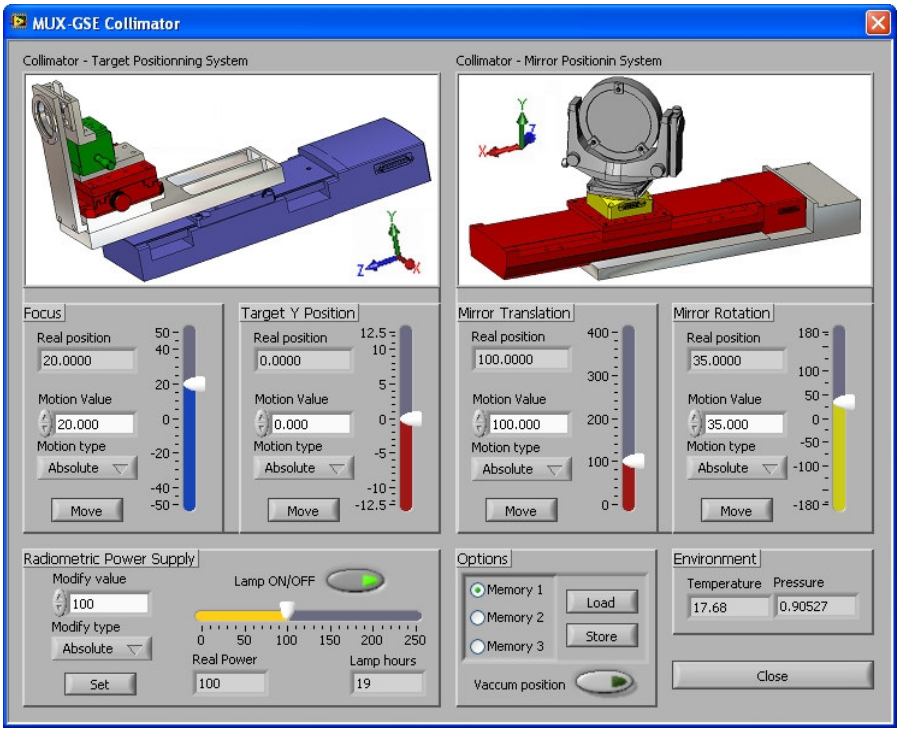


Figura 4.23: Tela de controle do colimador principal

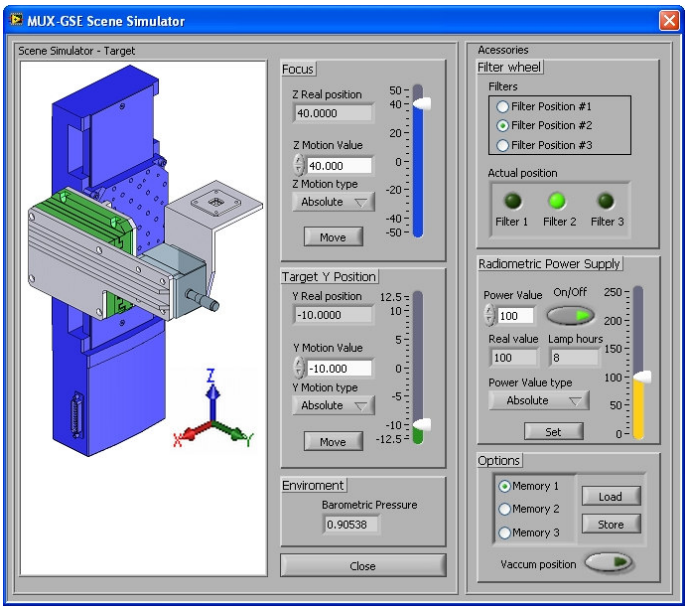


Figura 4.24: Tela de controle do simulador de cena

Com a chegada dos componentes restantes, as telas foram completadas em apenas mais uma semana de trabalho, que contou com a participação de todos em tempo parcial, devido a problemas não ligados diretamente ao projeto. As telas prontas foram utilizadas para os primeiros testes do colimador e dos simuladores de cena, já com todos os mecanismos integrados. Através delas foi possível detectar algumas falhas que foram corrigidas e verificar o desempenho dos dispositivos.

4.13.1 Resultados

As interfaces construídas foram completamente funcionais e permitiram a execução de todas as ações possíveis para o colimador e o simulador de cena. Isto foi feito ainda antes que todos os mecanismos estivessem completamente montados. Novamente, apesar de todos os contratempos, a equipe de *software* conseguiu mantê-lo longe do caminho crítico do projeto.

4.13.2 Conclusões

O desenvolvimento destas telas foi rápido, pois devido à sua importância, na primeira etapa dois engenheiros puderam se dedicar integralmente ao trabalho e na segunda, apesar da impossibilidade de trabalho em tempo integral, todos os membros da equipe participaram ativamente. A possibilidade de direcionar os trabalhos de acordo com a relevância das funções permitiu alcançar estes resultados.

Com a conclusão destas telas e a possibilidade de controle manual da esfera integradora, enquanto sua tela de controle não é construída, é possível realizar qualquer um dos testes previstos e que ainda não foram automatizados. Assim, nenhum procedimento de teste da MUX será inviabilizado por falta do MUX-GSE.

4.14 Aspectos de Manutenção do Software

Neste ponto, os *softwares* do MUX-GSE já permitiam a realização manual de todos os testes previstos. Restava agora a automatização dos testes ópticos, a tela da esfera integradora e outros itens de menor relevância, como mostra a Tabela 4.6. Alguns dos testes são complexos, como a MTF e o campo angular de visão, porém os algoritmos necessários já eram conhecidos e quando os três conjuntos de *racks* do MUX-GSE estivessem disponíveis a depuração das funções seria mais rápida.

Tabela 4.6 – Próximas funções a serem desenvolvidas para o MUX-GSE

Próximas etapas de desenvolvimento	
Telas principais	Esfera integradora
Rotinas de teste	Alinhamento
	Precisão do registro banda a banda
	Campo angular de visão
	Controle de ganho
	Resolução geométrica
	Distorção da imagem
	MTF
	Temperatura de operação
	Sensibilidade à polarização
	Luz espalhada
	Calibração radiométrica
	Faixa radiométrica
	Resolução radiométrica
	Resposta espectral
	Controle de temperatura
Funções adicionais	Impressão de resultados
Telas de equipamentos	Aplicativo Cliente

Estimava-se que a construção de todas as rotinas restantes tomasse ainda pelo menos três meses. Sabia-se, entretanto, que o projeto não terminaria neste ponto. Durante todo o desenvolvimento das câmeras MUX nos anos seguintes, poderia haver necessidade de modificações ou de correção de erros detectados durante a utilização.

Esta etapa, correspondente à manutenção, cabe aos membros da equipe original. Mesmo que haja algum imprevisto e outras pessoas tenham que executá-la, a documentação

produzida é suficiente para que alguém capacitado compreenda o funcionamento do *software* em pouco tempo e possa dar continuidade aos trabalhos. Para isso, pode-se contar com o código-fonte claro e bem documentado, o conjunto de relatórios entregue ao INPE e o pacote de manuais, ajuda, etc., que deve ser produzido após a conclusão do *software*, para validação da versão completa.

5 Conclusões Gerais

O desenvolvimento de *softwares* do MUX-GSE foi conduzido com sucesso, a despeito de alguns contratempos, conforme mencionado no decorrer do texto. Apesar de ser impossível prever estas situações, em projetos envolvendo tecnologia de ponta, elas são sempre esperadas. Tais dificuldades foram contornadas pela modificação na seqüência das atividades de desenvolvimento do *software*, que não trouxe maiores prejuízos graças à flexibilidade no planejamento. Com um método tradicional, mudanças no cronograma seriam muito mais difíceis.

Todas estas alterações mostram que a opção por um método flexível foi a mais acertada. As funções foram produzidas por ordem de importância, dentro das possibilidades momentâneas, exceto na primeira parte do projeto, quando o objetivo era aprender e ganhar segurança na utilização do *Labview*. A utilização desta estratégia fez com que não houvesse nenhum momento em que os engenheiros da MUX necessitassem executar alguma operação cuja função do *software* não estivesse disponível.

Observaram-se, por outro lado, algumas falhas graves que devem ser revistas em projetos futuros. A maior delas foi a execução de planejamento e apresentação de cronograma sem a participação de membros da equipe de desenvolvimento de *software*. Praticamente todos os autores da área de engenharia de *software* consultados afirmam que somente o pessoal técnico tem condições de estipular prazos de entrega. Em um projeto ágil, gerentes, diretores e clientes têm seu papel, porém devem deixar que os desenvolvedores decidam sozinhos o tempo necessário para a construção das funções solicitadas. Quando isto não ocorre, a qualidade do produto é colocada em risco.

Além deste problema, nota-se que existe ainda grande dificuldade em estimar prazos, principalmente porque os engenheiros não dedicam todo o seu tempo de trabalho ao desenvolvimento de *software* e muitas vezes atuam simultaneamente em mais de um projeto. Para melhorar este quadro, talvez seja necessário adotar uma estratégia parecida com a proposta na XP. Nela, o dia de trabalho no projeto é diferente do dia útil e a estimativa é revista freqüentemente. Assim, é possível que um dia de trabalho represente três ou quatro dias úteis, dependendo da dedicação do funcionário às atividades, e este valor é corrigido de tempos em tempos, utilizando o histórico do projeto. Este é um ponto que merece maior discussão, para que atrasos excessivos nunca arrisquem a viabilidade de um produto.

Outra falha constatada foi a falta de investimento na preparação dos membros da equipe antes do início do projeto. O orçamento dos cursos de *Labview* apresentados foi considerado alto e os engenheiros tiveram que desenvolver técnicas por conta própria. Em um projeto com custo tão elevado, prazo apertado e alto risco envolvido, esta economia deveria ter sido melhor avaliada.

Em virtude da capacidade dos funcionários, isto não se tornou um problema sério, mas um tempo precioso do projeto foi gasto até que todos estivessem razoavelmente seguros da utilização da plataforma. Além disso, algumas soluções criadas certamente teriam sido construídas de maneira diferente se houvesse maior conhecimento prévio sobre as capacidades do *Labview*. Entretanto, o resultado final foi bem satisfatório. O código coletivo e a refatoração reduzem os efeitos nocivos, pois o código é sempre revisto e melhorado.

Outra importante qualidade do método utilizado é a possibilidade de se iniciar uma etapa sem encerrar a anterior, e até sobrepor várias atividades independentes. Isto pode ser evidenciado na Seção 4, que mostra que várias ações foram executadas simultaneamente. Apesar de somente um engenheiro ser responsável por cada uma delas, a interação fez com

que todos estivessem mais ou menos informados sobre todas as partes do *software* e pudessem trocar informações mantendo alto nível de qualidade.

A forte dependência das pessoas envolvidas no projeto não se mostrou um problema. Pelo contrário: a responsabilidade delegada aos desenvolvedores promoveu a união da equipe em busca das metas estabelecidas. Com isso, nos momentos em que houve necessidade, o esforço concentrado do grupo evitou atrasos e prejuízos ao projeto.

As características ágeis permitiram que os desenvolvedores trocassem experiências e buscassem novos conhecimentos ao longo do trabalho, tornando a equipe cada vez melhor preparada para desempenhar seu trabalho. Mesmo quando um novo membro foi integrado ao grupo, tanto a documentação, quanto a interação foram fundamentais e permitiram que esta mudança ocorresse sem grande impacto. Observou-se sim algum prejuízo, mas as práticas utilizadas amenizaram as perdas naturais nesta situação.

Os resultados obtidos corroboram a tese que o desenvolvimento ágil permite o mesmo nível de qualidade alcançado por métodos tradicionais. Apesar do reduzido volume de documentos produzidos e das técnicas relativamente simples, o sistema desenvolvido apresentou poucas falhas e as que surgiram foram detectadas e corrigidas rapidamente, já que o *software* era testado imediatamente em condições reais.

Por fim, é importante ressaltar que o método utilizado permitiu que *softwares* muito complexos fossem desenvolvidos em curto espaço de tempo, por uma equipe composta de apenas quatro pessoas, responsáveis ainda por outras atividades que tomavam muito de seu tempo. O produto obtido, além de não prejudicar o andamento das atividades de desenvolvimento da MUX, possui a qualidade necessária para transmitir confiança aos usuários do MUX-GSE.

Com isso, pode-se considerar que o objetivo principal do trabalho foi alcançado, pois o processo de *software* implementado foi efetivo nas situações corriqueiras e, mais

importante, permitiu a reação em condições adversas, mostrando-se mais capacitado para obtenção de sucesso neste projeto que métodos mais conhecidos. Isto não significa que uma abordagem ágil seja a melhor em qualquer situação, porém comprova que ela adapta-se perfeitamente às condições encontradas na Opto Eletrônica e, portanto, deve continuar sendo aplicada e aperfeiçoada nos próximos projetos.

Outras empresas de *software*, mesmo atuando em áreas diferentes da aeroespacial, podem aproveitar-se das práticas aqui apresentadas, desde que sejam consideradas suas características particulares e promovidas as devidas modificações no método de desenvolvimento. O processo de *software* apresentado neste trabalho, de maneira geral, pode ser adaptado e aplicado sempre que um método ágil for adequado, entretanto, quando o sistema apresentar requisitos rígidos, equipe muito grande ou entraves graves à comunicação, entre outras dificuldades, é provável que métodos baseados no modelo clássico sejam mais apropriados.

Referências*

ABRAHAMSSON, P. et al. **Agile Software Development Methods**: Review and Analysis. Espoo, Finlândia: VTT, 2002. 107 p.

AGÊNCIA ESPACIAL BRASILEIRA.**Programa Nacional de Atividades Espaciais**: *PNAE*. Brasília: Ministério da Ciência e Tecnologia, Agência Espacial Brasileira, 2005. 114 p.

BECK, K. **Extreme Programming explained**: embrace change. Reading, MA: Addison-Wesley, 1999a. 190 p.

_____. Embracing Change with Extreme Programming. **Computer**, NewYork, v. 32, n. 10, p. 70-77, out. 1999b.

BECK, K.; BOEHM, B. Agility through Discipline: A Debate. **Computer**, New York, v. 36, n. 6, p. 44-46, jun. 2003

BECK, K.; CLEAL, D. **Optional Scope Contracts**, 1999. Disponível em: <http://www.xprogramming.com/ftp/Optional+scope+contracts.pdf>, Acesso em: 14 jan. 2007.

BECK, K. et al. **Manifesto for Agile Software Development**, 2001. Disponível em: <http://www.agilemanifesto.org/>, Acesso em: 19 out. 2006.

BLOTNER, J. A. It's More than Just Toys and Food: Leading Agile Development in an Enterprise-Class Start-Up. In: AGILE DEVELOPMENT CONFERENCE, 2003, Salt Lake City. **Proceedings…**, New York: IEEE Computer Society, 2003, p. 81-91.

BRASIL. Lei n. 8.854, de 10 de fevereiro de 1994. Cria a Agência Espacial Brasileira (AEB). **Diário Oficial da União**, Brasília, 11 fev. 1994a. Seção 1, p. 2089.

* De acordo com:
ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS. **NBR 6023**: informação e documentação: referências: elaboração. Rio de Janeiro, 2002.

_____. Decreto n° 1.332, de 8 de dezembro de 1994. Aprova a atualização da Política de Desenvolvimento das Atividades Espaciais (PNDAE). **Diário Oficial da União**, Brasília, 09 dez. 1994b. Seção 1, p. 18887.

_____. Presidência da República. Agência Espacial Brasileira. **Programa Nacional de Atividades Espaciais**: 1998 – 2007. 2. ed. Brasília: AEB, 1998. 76 p.

_____. Protocolo Complementar Brasil X China, de 12 de novembro de 2004. Protocolo complementar ao acordo quadro entre o Governo da República Federativa do Brasil e a República Popular da China sobre cooperação em aplicações pacíficas de ciência e tecnologia do espaço exterior para desenvolvimento conjunto do satélite CBERS-2B. **Diário Oficial da União**, Brasília, 29 de nov. 2004. Seção 1, p. 49.

BROOKS JÚNIOR, F. P. No silver bullet: Essence and accidents of Software Engineering. **Computer**, New York, v. 20, n. 4, p. 10-19, abr. 1987.

COCKBURN, A. **Characterizing people as non-linear, first-order compoenents in software development**, 1999. Disponível em: <http://alistair.cockburn.us/index.php/Characterizing_people_as_non-linear,_first-order_components_in_software_development>, Acesso em: 20 out. 2006.

_____. **Agile Software Development**. Boston: Addison-Wesley, 2002. 278p.

_____. **People and methodologies in software development**. 2003. 87p. Tese (Doutorado) – Faculdade de Matemática e Ciências Naturais, Universidade de Oslo, Oslo, Noruega, 2003.

COCKBURN, A.; HIGHSMITH, J., Agile Software Development: The People Factor. **Computer**, New York, v. 34, n. 11, p. 131-133, nov. 2001.

COCKBURN, A.; WILLIAMS, L.The Costs and Benefits of Pair Programming. In: SUCCI, G.; MARCHESI, M. **Extreme Programming Examined**, Boston, MA: Addison-Wesley, 2001, p. 223-248.

DEMARCO, T.; BOEHM, B. The Agile Methods Fray. **Computer**, New York, v. 35, n. 6, p. 90-92, jun. 2002.

DEMARCO, T.; LISTER, T., **Peopleware**. Como Gerenciar Equipes e Projetos Tornando-os mais Produtivos. São Paulo: McGraw-Hill, 1990. 223p.

EPIPHANIO, J. C. N. CBERS – Satélite Sino-Brasileiro de Recursos Terrestres. In: SIMPÓSIO BRASILEIRO DE SENSORIAMENTO REMOTO, 12., 2005, Goiânia. **Anais...** São José dos Campos: INPE, 2005. p. 915-922. CD-ROM.

FOWLER, M., **The new methodology**, 2000. Disponível em: <http://www.martinfowler.com/articles/newMethodology.html>, Acesso em: 19 out. 2006.

GERAS, A.; SMITH, M.; MILLER, J. A Prototype Empirical Evaluation of Test Driven Development. In: INTERNATIONAL SYMPOSIUM ON SOFTWARE METRICS (METRICS'04), 10, 2004, Chicago. **Proceedings…** New York: IEEE Computer Society, 2004. p. 405-416.

HIGHSMITH, J. **History**: The Agile Manifesto, 2001. Disponível em: <http://www.agilemanifesto.org/history.html>, Acesso em: 19 out. 2006.

HIGHSMITH, J.; COCKBURN, A., Agile Software Development: The Business of Innovation. **Computer**, New York, v. 34, n. 9, p. 120-127, set. 2001.

HUO, M. et al. How does agility ensure quality? In: WORKSHOP ON SOFTWARE QUALITY (ICSE 2004), 2, 2004, Edinburg, Scotland. **Proceedings…**, IET Digital Library, 2004. p. 36-40.

INSTITUTO NACIONAL DE PESQUISAS ESPACIAIS. **CBERS**. Disponível em: <http://www.cbers.inpe.br/pt/ index_pt.htm>, Acesso em: 15 ago. 2006a.

_____. **CBERS** – Catálogo CBERS disponibiliza imagens da América do Sul. Disponível em: <http://www.cbers.inpe.br/pt/imprensa/not84.htm>, Acesso em: 15 ago. 2006b.

_____. **CBERS** – Estados Unidos Recebem com Sucesso Imagens do Satélite Sino-Brasileiro. Disponível em: <http://www.cbers.inpe.br/pt/imprensa/not79.htm>, Acesso em: 15 ago. 2006c.

_____. **CBERS** – Galeria de Fotos (CBERS-2). Disponível em: <http://www.cbers.inpe.br/ pt/imprensa/fotografia2.htm>, Acesso em: 22 ago. 2006d.

_____. **CBERS** – Satélites – As Câmeras dos Satélites CBERS-1 e 2. Disponível em: <http://www.cbers.inpe.br/pt/programas/cbers1-2_cameras.htm>, Acesso em: 15 ago. 2006e.

_____. **CBERS** – Satélites – Características Preliminares das Câmeras dos Satélites CBERS-3 e 4. Disponível em <http://www.cbers.inpe.br/pt/programas/cbers3-4_cameras.htm>, Acesso em: 20 ago. 2006f.

_____. **CBERS** – Satélites – Participação da Indústria Nacional na Construção dos Satélites CBERS-1 e 2. Disponível em <http://www.cbers.inpe.br/pt/programas/part_industrial.htm>, Acesso em: 20 ago. 2006g.

MILESKI, A. M. **Programa Espacial China-Brazil – Satélite Sino-Brasileiro de Recursos Terrestres (CBERS)**. Universidade Federal de Juiz de Fora – História militar, defesa, estratégia e tecnologia, Juiz de Fora: 2003. Disponível em: <http://www.defesa.ufjf.br/arq/art4.htm>. Acesso em: 21 ago. 2006.

MINISTÉRIO DA CIÊNCIA E TECNOLOGIA. **Qualidade e Produtividade no Setor de Software Brasileiro:** Pesquisa 1999. Disponível em: <http://www.mct.gov.br/index.php/content/view/3255.html>. Acesso em: 01 nov. 2006.

_____. **Qualidade e Produtividade no Setor de Software Brasileiro:** Pesquisa 2001. Disponível em: <http://www.mct.gov.br/index.php/content/view/3254.html>. Acesso em: 01 nov. 2006.

NATIONAL INSTRUMENTS. **Labview**: Development Guidelines. Austin, TX: National Instruments, 2003. 97 p.

NONEMACHER, M. L. **Comparação e avaliação entre o processo RUP de desenvolvimento de software e a metodologia Extreme Programming**. 2003. 164 p. Dissertação (Mestrado em Ciência da Computação) – Universidade Federal de Santa Catarina, Florianópolis, 2003.

POPPENDIECK, M. Lean Programming: Part 1 of 2: Assembly-line production techniques apply to software, too. **Software Development Magazine**, mai. 2001a. Disponível em: <http://www.ddj.com/dept/architect/184414734>. Acesso em: 04 nov. 2006.

_____. Lean Programming: W. Edwards Deming's Total Quality Management still rings true for software: Part 2 of 2. **Software Development Magazine**, jun. 2001b. Disponível em: <http://www.ddj.com/dept/architect/184414744>. Acesso em: 04 nov. 2006

_____. **Righteous Contracts**, 2002. Disponível em: <http://www.poppendieck.com/righteous.htm>. Acesso em: 14 jan. 2007.

PRESSMAN, R. S. **Software engineering:** A practitioner's approach. 6. ed. New York: McGraw-Hill, 2005. 880 p.

PROGRAMA Educa SeRe – Elaboração de Material Didático para o Ensino de Sensoriamento Remoto Utilizando Imagens CBERS. São José dos Campos: INPE. Disponível em: <http://www.inpe.br/unidades/cep/atividadescep/educasere/index.htm>. Acesso em: 27 ago. 2006.

RISING, L; JANOFF, N. S. The Scrum software development process for small teams. **IEEE Software**, New York, v. 17, n. 4, p. 26-32, jul./ago. 2000.

SAUSEN, T. M. **Sensoriamento remoto e suas aplicações para recursos naturais**. INPE. Programa Educa SeRe – Apostila, São José dos Campos, 2006. Disponível em: <http:// www.inpe.br/unidades/cep/atividadescep/educasere/>. Acesso em: 25 ago. 2006.

SILVEIRA, V. **Seis empresas brasileiras vão construir os satélites CBERS**. Gazeta Mercantil. Brasília: Agência CT. Ministério da Ciência e Tecnologia. Disponível em <http:// agenciact.mct.gov.br/index.php?action=/content/view&cod_objeto=23493>, Acesso em: 21 ago. 2006.

SHORE, J. Continuous Design. **IEEE Software**, New York, v. 21, n. 1, p. 20-22, jan./fev. 2004.

SIMPÓSIO BRASILEIRO DE SENSORIAMENTO REMOTO, 12, 2005, Goiânia. **Anais...** São José dos Campos: INPE, 2005. 1 CD-ROM.

SOCIEDADE BRASILEIRA PARA O PROGRESSO DA CIÊNCIA. GT sobre Política Espacial. **Uma avaliação do Programa Espacial Brasileiro**. Seminário sobre o Programa Espacial, Agência Espacial Brasileira, Brasília, 2001. Adobe Reader 6.0. Disponível em: <http://www.dpi.inpe.br/gilberto/present/avaliacao_pnae.pdf>. Acesso em: 10 ago. 2006.

SOFTWARE ENGINEERING, 1968, Garmisch, Alemanha. **Reports...** Brussels, Bélgica: Scientific Affairs Division, OTAN.

RACOON, L. B. S. Toward a tradition of software engineering. **ACM SIGSOFT Software Engineering Notes**, New York, NY, v. 23, n. 3, p. 105-110, mai. 1998.

TELES, V. M. **Extreme Programming**. Aprenda como encantar seus usuários desenvolvendo software com agilidade e alta qualidade. São Paulo: Novatec, 2004. 316 p.

_____. **Um estudo de caso da adoção das práticas e valores do extreme programming**. 2005. 180 p. Dissertação (Mestrado em Informática) – Instituto de Matemática e Núcleo de Computação Eletrônica, Universidade Federal do Rio de Janeiro, Rio de Janeiro, 2005.

UNIVERSIDADE DE SÃO PAULO. Sistema Integrado de Bibliotecas. Grupo DiTeses. **Diretrizes para apresentação de dissertações e teses da USP**: documento eletrônico e impresso. São Paulo: SIBi-USP, 2004. 110 p.

VASCONCELOS, C. R. **XPU – Um modelo para o desenvolvimento de sistemas centrado no usuário**. 2004. 185 p. Dissertação (Mestrado em Informática) – Centro de Ciência e Tecnologia, Universidade Federal de Campina Grande, Campina Grande, PB, 2004.

Livros Grátis

(http://www.livrosgratis.com.br)

Milhares de Livros para Download:

Baixar livros de Literatura
Baixar livros de Literatura de Cordel
Baixar livros de Literatura Infantil
Baixar livros de Matemática
Baixar livros de Medicina
Baixar livros de Medicina Veterinária
Baixar livros de Meio Ambiente
Baixar livros de Meteorologia
Baixar Monografias e TCC
Baixar livros Multidisciplinar
Baixar livros de Música
Baixar livros de Psicologia
Baixar livros de Química
Baixar livros de Saúde Coletiva
Baixar livros de Serviço Social
Baixar livros de Sociologia
Baixar livros de Teologia
Baixar livros de Trabalho
Baixar livros de Turismo