Rafael Lopes de Matos

Especificação do Processo de Desenvolvimento de Software Embarcado Utilizando o BPMN

Monografia apresentada ao PECE - Programa de Educação Continuada em Engenharia da Escola Politécnica da Universidade de São Paulo como parte dos requisitos para conclusão do curso de MBA em Tecnologia de *Software*.

São Paulo

2013

Rafael Lopes de Matos

Especificação do Processo de Desenvolvimento de Software Embarcado Utilizando o BPMN

Monografia apresentada ao PECE - Programa de Educação Continuada em Engenharia da Escola Politécnica da Universidade de São Paulo como parte dos requisitos para conclusão do curso de MBA em Tecnologia de *Software*.

Área de Concentração: Tecnologia de Software

Orientador: Jorge Rady de Almeida Júnior

Doutor em Engenharia Elétrica - EPUSP

São Paulo

2013

FICHA CATALOGRÁFICA

Matos, Rafael Lopes de

Especificação do Processo de Desenvolvimento de Software Embarcado Utilizando o BPMN / R.L. de Matos. -- São Paulo, 2013.

91 p.

Monografia (MBA em Tecnologia de Software) -- Escola Politécnica da Universidade de São Paulo. Programa de Educação Continuada em Engenharia.

1. Desenvolvimento de software 2. Software embarcado 3. Notação BPMN I. Universidade de São Paulo. Escola Politécnica. Programa de Educação Continuada em Engenharia II. t.

DEDICATÓRIA

Aos meus pais, Claudete e Francisco, por todo amor e todas as grandes lições que pude receber e que ainda terei. Não há nada que possa expressar minha gratidão sem limites aos dois. Vocês são tudo para mim.

À minha irmã Francislene, Ricardo, Larissa, Lucas e Letícia, que são os melhores amigos que eu poderia ter.

Ao meu irmão Marcelo, por me lembrar a cada dia o valor da vida.

À Izabel, minha companheira nesta jornada e outras mais, por todo amor, compreensão e apoio.

AGRADECIMENTOS

A Deus, por sempre estar ao meu lado, por guiar-me nas escolhas e por proporcionar todas as oportunidades que tenho em minha vida.

A todos os parentes e amigos que deram apoio em mais este desafio.

Ao professor Jorge Rady de Almeida Júnior pela orientação, sem a qual este trabalho não se realizaria.

Aos professores do curso de Tecnologia de *Software*, pelos seus ensinamentos e aos funcionários do PECE e da Escola Politécnica, que durante esses dois anos, contribuíram de algum modo para meu enriquecimento pessoal e profissional.

A você, prezado(a) leitor(a), por prestigiar este trabalho. Espero que este o(a) ajude em seus objetivos.

"Escolhe um trabalho de que gostes; e não terás que trabalhar nem um dia na tua vida.".

Confúcio

RESUMO

Este trabalho especifica o processo de desenvolvimento de *software* embarcado utilizando-se a notação BPMN. Os objetivos deste são a melhora na visualização dos processos, rastreabilidade durante a execução do mesmo e melhor definição de tarefas e papéis em ambiente industrial, proporcionando aumento da eficiência no desenvolvimento de aplicações em componentes eletrônicos. Expõe-se, então, uma sucessão de evoluções para a criação de um modelo, passando pela definição de papéis, metodologias ágeis, levantamento de requisitos, monitoramento das tarefas, refatoração e tipos de documentação necessárias, resultando em uma proposta genérica para utilização na manufatura de sistemas embarcados, podendo ser utilizado em ambientes empresais ou similares, passível de monitoramento e de sugestões de melhorias futuras. Adicionalmente, ressaltam-se aspectos não incluídos no processo padrão, como o fator humano, o local de trabalho, linguagens de programação e ferramentas computacionais utilizadas, além do contato com outros elementos internos e externos à empresa. O modelo proposto cobre diversos contextos para a produção de *software* embarcado, tornando-se referência para adequação aos processos de empresas com diversas características.

Palavras-chaves: Desenvolvimento de *software*. *Software* embarcado. Notação *Business Process Modeling Notation* (BPMN).

ABSTRACT

This paper specifies the process of embedded software development using BPMN notation. The objectives are the improvement in process visualization, traceability during its execution and better definition of roles and tasks in an industrial environment, allowing efficiency improvement when developing electronic devices applications. It is exposed in a set of evolutions for the creation of a model, defining roles, agile methodologies, requirements, task monitoring, refactoring and documentation needed, resulting in a multipurpose model for embedded systems manufacturing, which could be used in empreprises or similar environments, with possibility of monitoring and improvements. Additionally, it focus on aspects out of the standard, such as human factor, workplace environment, programming languages, computer tools and elements for inside and outside the company. The proposed model covers diverse contexts for embedded software production, being a reference for different companies set of processes.

Keywords: Software development. Embedded software. BPMN.

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS E SIGLAS

AADL *Architecture Analysis & Design Language*

AMD *Advanced Micro Devices*

API *Application Programming Interface*

ARM *Advanced Reduced Instruction Set Computer (RISC) Machine*

ASIC *Application Specific Integrated Circuit*

BCA *Bus Cycle Accurate*

BPD *Business Process Diagram*

BPM *Business Process Management*

BPMI *Business Process Management Initiative*

BPMN *Business Process Modeling Notation*

CAD *Computer-Aided Design*

CMM *Capability Maturity Model*

CMMI *Capability Maturity Model Integration*

CVS *Concurrent Version System*

DSP *Digital Signal Processor*

ENIAC *Electronic Numerical Integrator And Computer*

FSM *Finite-State Machine*

FPGA *Field-Programmable Gate Array*

FW *Firmware*

GIT *Global Information Tracker*

GNU *GNU is Not Unix*

HW *Hardware*

IBM *International Business Machines*

IDE *Integrated Development Environment*

IEC *International Electrotechnical Commission*

IEEE *Institute of Electrical and Electronics Engineers*

IIC *Inter-Integrated Circuit*

ISO *International Organization for Standardization*

JTAG *Joint Test Action Group*

LSI *Logarithmic Sobolev Inequality*

MDD *Model-Driven Development*

MIPS *Microprocessor without Interlocked Pipeline Stages*

NEC *Nippon Electric Company*

OMG *Object Management Group*

OTG *On-The-Go*

PMBOK *Project Management Body of Knowledge*

POSIX *Portable Operating System Interface*

RAM *Random Access Memory*

RISC *Reduced Instruction Set Computer*

ROM *Read Only Memory*

RTL *Register Transfer Level*

RTOS *Real-Time Operating System*

SE *Software* Embarcado

SHARC *Super Harvard Architecture single-chip Computer*

SOA *Service-Oriented Architecture*

SoC *System on a Chip*

SPARC *Scalable Processor Architecture*

SPIES *Software Process Improvement for Embedded Systems*

SVN *Apache Subversion*

SW *Software*

TDD *Test Driven Development*

TF *Timed Function*

UML *Unified Modeling Language*

USB *Universal Serial Bus*

UTF *UnTimed Function*

Wi-Fi *Wireless Fidelity*

XP *Extreme Programming*

SUMÁRIO

1 INTRODUÇÃO

1.1 MOTIVAÇÕES

A preocupação com a melhoria dos processos produtivos em qualquer área tem aumentado sensivelmente nas últimas décadas. Para facilitar a visualização destes processos de maneira analítica e gráfica foi criado, entre algumas opções existentes, o *Business Process Modeling Notation* (BPMN).

A área de *Software* Embarcado, uma evolução do simples e rudimentar *hardware* da década de 1940 (MORIMOTO, 2007) conhecido como *Electronic Numerical Integrator And Computer* (ENIAC), que passou por evoluções de tamanho, desempenho e capacidade de armazenamento de dados (possibilidade de embarcar código), tem crescido em uso e importância na sociedade atual, que é cada vez mais conectada. Tal cenário implica na necessidade de melhorar seus projetos e sua documentação de modo a facilitar seu processo de criação e, por consequência, os seus processos de concepção.

Nos últimos anos o impacto do *software* nas funcionalidades e inovação de sistemas embarcados aumentou com rapidez. Navegadores, editores de fotografia, comunicadores instantâneos e gerenciadores de equipamentos remotos estão entre os programas que tiveram notável evolução nos últimos tempos conforme as capacidades do *hardware* assim o permitiram. A necessidade de rápidas melhorias das aplicações para tornar-se a referência no competitivo mercado de soluções comerciais e domésticas, o aumento de sua complexidade e a necessidade de confiabilidade destes tornou viável sua produção inclusive economicamente (LIGGESMEYER; TRAPP, 2009), pela escala tomada. O desenvolvimento de sistemas autônomos de pequeno porte consistia apenas em tarefas simples, mas trabalhosas, como controle de sistemas por meio de algoritmos pequenos, por exemplo. O referido aumento de complexidade e demanda por várias tarefas tornou imprescindível o papel da engenharia de *software* na área.

A criação de um processo para a concepção de equipamentos eletrônicos com *software* embarcado em seus microcontroladores deve avaliar os elementos presentes no ambiente de sua criação, que por vezes são negligenciados e acabam por influir no resultado

final do projeto, por meio da qualidade do produto final e da data de sua entrega. Torna-se então desejável o controle da produção de eletrônicos para suprir a alta demanda por estes equipamentos, em tempo hábil e com alta qualidade para o consumidor também cada vez mais exigente. Esta preocupação é proveniente das experiências de trabalho do autor deste trabalho.

1.2 OBJETIVO

O objetivo deste trabalho é especificar um processo de desenvolvimento de *software* embarcado em notação de fácil compreensão, no caso o BPMN, por seu uso em considerável parte do mercado, enfatizando adicionalmente elementos que não podem ser acrescidos ao processo em si, mas que alteram o resultado final por estar relacionado ao desenvolvimento. Almeja-se que o processo proposto seja uma referência para qualquer entidade que deseja criar um produto eletrônico com aceitáveis níveis de controle sobre as atividades, durante sua produção, no âmbito de desenvolvimento de *hardware* e *software*, sem detalhar ou alterar os processos de setores próximos aos apresentados, como importações e produção.

O modelo proposto deve ser adaptado de acordo com a cultura do local de trabalho, a sua estrutura organizacional (colaboradores disponíveis) e dos prazos estipulados pela gerência. O detalhamento da proposta é dividido em etapas: a partir de um modelo de desenvolvimento simples encontrado na literatura, adicionado de elementos descritos a cada passo, vindos a partir da literatura técnica e decisões e sugestões advindas do autor, cujo convívio com a área técnica permite essa colaboração.

1.3 JUSTIFICATIVAS

A melhora nos processos de sistemas embarcados mostra significante redução no tempo dos ciclos de seu desenvolvimento, agregando valor para todas as atividades a eles relacionadas (GARCIA; ANDREA, 2010), apesar de sua prática não ser completamente adotada nas empresas da área. Ainda que a especificação adequada exista, por muitas vezes não segue uma notação de fácil visualização e edição, como apresentadas nos modelos de Gupta (2001), Chong e Xingchuan (2009) e Mitsui; Kambe e Koizumi (2009). Em outras

circunstâncias há também a modelagem realizada em linguagens inadequadas para tal, como citado por Antonio; Ferrari e Fabbri (2012) em seu estudo.

Com o intuito de melhorar a apresentação dos diagramas de processos a todos os envolvidos no desenvolvimento de sistemas embarcados com uma notação adequada para tal, foi escolhido o BPMN por sua robusta especificação e facilidade de uso de suas ferramentas. Ao final de sua implementação o valor agregado deve compensar o esforço gasto na migração, tendo retorno em curto prazo a médio prazo, de modo rastreável.

1.4 ESTRUTURA DO TRABALHO

Este trabalho aborda, no capítulo 2, a notação de processos conhecida por BPMN, abordando seu histórico, ferramentas, elementos que a compõem e exemplos de uso. Em seguida, o capítulo 3 decreve brevemente os principais aspectos referentes ao *software* embarcado, enquanto no capítulo 4, utilizando a notação BPMN, é apresentada a proposta central desta monografia: um processo para o desenvolvimento de *software* embarcado, embasado em tópicos na sua criação, como os papéis das pessoas, levantamento de requisitos. gerenciamento e documentação. O capítulo 5 apresenta considerações adicionais do modelo proposto, as quais devem ser analisadas durante a aplicação da especificação no ambiente de trabalho, entre elas o ambiente de trabalho, fábrica de *software* e interação com clientes e fornecedores, enquanto que o capítulo 6 contém as considerações finais deste trabalho.

2 *Business Process Modeling Notation* (BPMN)

Com a alta demanda e rapidez exigida em qualquer produção existente nos dias atuais, surge a necessidade de poder planejar, organizar e executar processos de negócio na cadeia de desenvolvimento e produção de todo e qualquer produto ou serviço entregue. Das variáveis críticas para cumprir prazos e manter a qualidade no produto final, deve-se considerar essencialmente o processo, que deve ser incorporado principalmente nas fases iniciais de desenvolvimento (KOUS, 2010) de forma a possibilitar a automação deste, seja total ou parcial, por meio da relação entre homens e máquinas seguindo regras estritamente delimitadas (RIESCO et al., 2009).

Existem indicativos da relação entre gerenciamento de processos e sucesso dos negócios. Se a organização puder responder prontamente a uma demanda maior que a costumeira por meio de readequação do processo de negócio, a empresa estará no caminho certo para o sucesso (WONGWATKIT, 2012).

Um processo bem definido pode não somente ajudar a própria empresa, mas também introduzir avanços em toda a cadeia produtiva, por exemplo podendo informar clientes sobre imprevistos ou previsões para melhor planejamento de ambos (WONGWATKIT, 2012).

2.1 CONSIDERAÇÕES INICIAIS

O BPMN, uma Notação de Modelagem de Processos de Negócio (em tradução livre), está atualmente na especificação de versão 2.0 (OBJECT MANAGEMENT GROUP, 2012) e é utilizado em vários projetos com eficiência, eficácia e agilidade (WONGWATKIT, 2012) os quais demandam melhor controle e visualização nos processos. Trata-se de um padrão gráfico muito utilizado em análises de domínio e visualização de alto nível para sistemas (KOUS, 2010), pois possibilita melhor organizar as ideias através da visão macro do que pode ocorrer durante a execução das tarefas.

Entre as vantagens desta notação está a sua capacidade de abstração e adaptação aos inúmeros fluxos de atividades que podem exisitir em um negócio, todos diferentes entre si como, por exemplo, negócios da área da saúde, de ciências humanas, processos industriais, de manufatura e outros além do desenvolvimento de *software* embarcado, objetivo principal deste trabalho.

Pode-se citar algumas das aplicações do BPMN (KOUS, 2010):

- Por analistas de negócio nos esboços iniciais de processos;
- Por desenvolvedores técnicos responsáveis pela implementação desses processos;
- Por gerentes que precisam sempre monitorar as tarefas acima descritas.

Por ser amplamente utilizada para modelar processos de negócio, várias ferramentas e técnicas facilitam o gerenciamento, execução e monitoramento dos processos referidos (PILLAT; OLIVEIRA; FONSECA, 2012). Seu modelamento adequado pode tornar mais rápido e melhorar a qualidade do desenvolvimento de *software* através da evolução da infra-estrutura da empresa. Importante observar, todavia, que é necessário realizar a instanciação, de acordo com o ambiente atual, podendo haver resultados negativos, em caso contrário.

Baseando-se na premissa que o *Business Process Management* (BPM) é a observação de um conjunto de tarefas que ao final entregam um produto (seja ele palpável ou não), sua representação fornece meios para monitoramento e inclusive melhora do andamento das tarefas, podendo sempre ser alterado para melhor (PILLAT; OLIVEIRA; FONSECA, 2012).

Na maioria dos diagramas, o nível mais alto consiste no conjunto de processos de gerenciamento, a camada intermediária envolve os processos de desenvolvimento de projeto e na camada mais baixa encontram-se os processos para poder suprir as demandas do projeto (JUN; RUI; YI-MIN, 2007).

2.2 HISTÓRICO

A notação BPMN foi criada pela *Business Process Management Initiative* (BPMI) em 2004 e é mantida desde o ano seguinte pelo *Object Management Group* (OMG), após

fusão de ambas. Teve ainda as versões 1.1 em 2008 e 1.2 no ano de 2009. Sua atual versão é a 2.0, lançada desde janeiro de 2011.

A notação, como um todo, é composta por diagramas chamados *Business Process Diagram* (BPD) que facilitam o entendimento do processo em análise e são de fácil criação, ainda assim possibilitando a modelagem de situações complexas.

2.3 FERRAMENTAS

Existem, no mercado, ferramentas computacionais de código livre e as que necessitam de licenças pagas para sua utilização. Por muitas vezes a escolha da melhor ferramenta depende do uso que se fará dela. Por serem didáticos, os diagramas feitos neste trabalho foram feitos utilizando a ferramenta gratuita *Bizagi Process Modeler*, obtida através de Bizagi (2012).

Algumas das ferramentas utilizadas para a criação dos diagramas de processo de negócio podem ser citadas (LIU et al., 2011):

- BizAgi Xpress;
- jBPM (em Java);
- Bonita;
- Oracle BPM;
- ADONIS;
- MEGA.

Na prática, o uso da notação pode ter os seguintes resultados positivos (MUEHLEN; HO, 2008):

- Documentar o processo corrente (caso ainda não o seja);
- Detectar e corrigir pontos fracos dos atuais processos;
- Introduzir aos participantes toda a estrutura processual através de fluxogramas simples;

- Definir melhor o papel e tarefas de cada pessoa nos procedimentos;
- Simular os impactos financeiros e operacionais das mudanças;
- Quantificar efeitos de melhorias realizadas;
- Remover passos desnecessários.

Infelizmente, alguns argumentos contrários podem ser observados durante sua utilização (MUEHLEN; HO, 2008; CHINOSI; TROMBETTA, 2009):

- Pode requerer algumas iterações até determinar o modelo final (inerente ao processo de revisão das características);
- Falta de distinção entre elementos e definições (melhorado na versão 2.0, mas ainda com pequenas falhas);
- Aumento de complexidade na última versão, devido ao grande número de opções agregadas na especificação.

2.4 ELEMENTOS DA NOTAÇÃO

Para a criação dos diagramas de processos de negócio, foram criados diversos elementos, separados em quatro principais categorias:

1. **Objetos de Fluxo**
2. **Objetos de Conexão**
3. **Piscinas e Raias**
4. **Artefatos**

Dentro das categorias listadas, muitas variantes foram adicionadas, especialmente na migração da versão 1.2 para a 2.0, possibilitando melhor determinação nas tarefas descritas.

2.4.1 Objetos de Fluxo

Principais elementos da notação, os objetos de fluxo são essencialmente divididos em: Evento de Início, Gateway/Decisão, Evento Intermediário, Atividade e Evento Finalizador, conforme a figura 2.1.

Figura 2.1: Exemplos de Objetos de Fluxo do BPMN

Eventos

Eventos são essencialmente representações de início de um processo, de fim de processo e também de eventos intermediários que podem ocorrer. Tais eventos indicam qual conjunto de tarefas deve ser realizado ao receber uma mensagem, ao passar determinado período de tempo, ao ocorrer algum cancelamento, entre outros.

Atividades

As atividades, em essência, podem ser determinadas como qualquer atividade de trabalho executada durante o processo de negócio em andamento. São sub-dividas em Processo, Sub-Processo e Tarefa, esta última considerada a ação atômica (não mais divisível) da notação, que pertence a um Processo.

Gateways

Gateways, por sua vez, determinam o fluxo do processo através de decisões a serem tomadas dependendo da situação corrente (controle do fluxo).

2.4.2 Objetos de Conexão

Como diz o próprio nome, os Objetos de Conexão ligam os elementos existentes no diagrama de processo, de modo a indicar o fluxo de funcionamento deste. Os tipos de Objetos são: Fluxo de Sequência, Associação e Fluxo de Mensagem, conforme a figura 2.2.

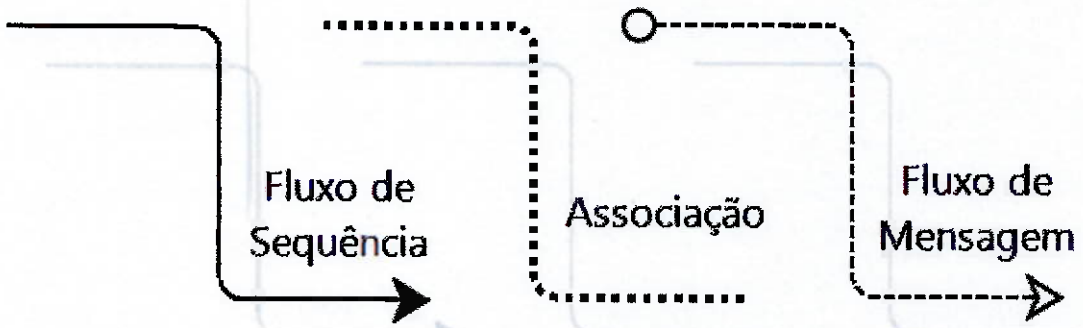

Figura 2.2: Exemplos de Objetos de conexão do BPMN

Fluxo de Sequência

Fluxos de Sequência são utilizados para indicar a próxima tarefa ou próximo processo a ser executado durante o processo.

Fluxo de Mensagem

Os Fluxos de Mensagem geralmente indicam que um documento ou outro tipo de mensagem está sendo enviado de um agente executor de tarefas (ou uma organização, por exemplo) a outro.

Associação

Associações são utilizadas para comentários e outras indicações que não afetam o curso das tarefas a executar. Devem sempre associar um Artefato a um Objeto de Fluxo.

2.4.3 Piscinas e Raias

Piscinas e Raias constituem a tradução livre de *Pools* e *Lanes*, respectivamente. A seguir apresenta-se uma pequena descrição e suas representações, exemplificadas na figura 2.3.

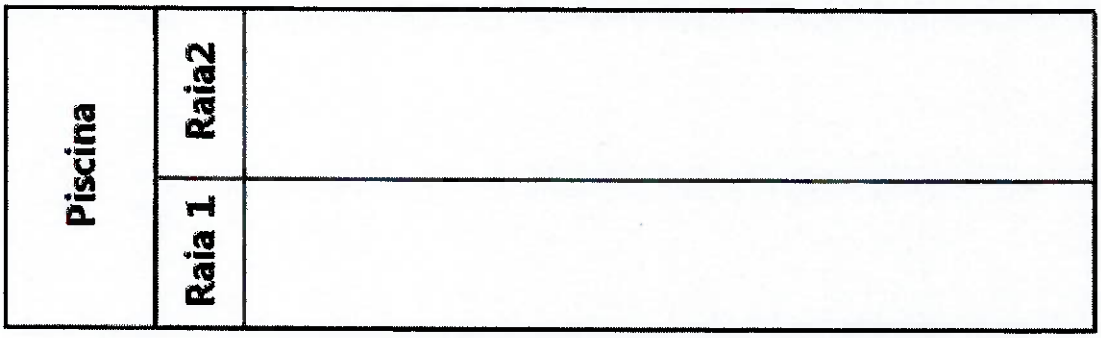

Figura 2.3: Exemplos de Piscinas e Raias do BPMN

Piscinas (*Pool*)

Piscinas são a representação gráfica para um participante ou conjunto de participantes qualquer que esteja envolvido no processo de negócio diagramado. Pode ser uma organização ou um setor, por exemplo.

Raias (*Lanes*)

As Raias indicam uma subdivisão da piscina, para melhor entendimento do que pode acontecer em um determinado ambiente. Pode, por exemplo, representar um cargo dentro de um setor ou ou setor dentro de uma empresa, como programador ou equipe de desenvolvimento, por exemplo.

2.4.4 Artefatos

Artefatos são componentes que não estão diretamente relacionados ao fluxo de sequências e de mensagens dos diagramas BPMN, mas que representam informações adicionais necessárias para um melhor entendimento do negócio proposto, conforme mostra a figura 2.4.

Objeto de Dados

Objeto de dados podem ser parte importante no fluxo de ações descritas nos diagramas, podendo representar um arquivo, documento de texto, relatório ou qualquer outro elemento com informações relevantes para os processos.

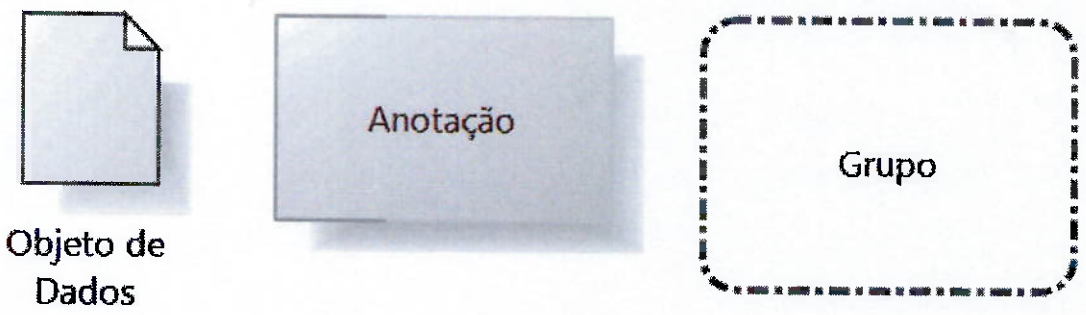

Figura 2.4: Exemplos de Artefatos do BPMN

Grupo

Para propiciar uma melhor visualização de elementos que pertencem a um conjunto similar de tarefas, é aconselhável a utilização de Grupos, cujo formato é um retângulo com cantos arredondados e contorno tracejado.

Anotação

Anotações são utilizadas apenas em caso de necessidade de um comentário ou observação acerca de um determinado elemento no diagrama, sendo representado por um retângulo aberto e linha contínua.

2.5 EXEMPLOS

A notação também possibilita criar um tipo próprio de Artefato ou mesmo Objeto de Fluxo, proporcionando melhor entendimento do processo e, por consequência, um diagrama mais compreensível.

Na figura 2.5, pode-se ter a noção de como são ordenados os elementos dentro do diagrama. Na versão atual da notação existem muitas variantes dos elementos, cuja utilização em um único exemplo torna-se dificultada por não caber em apenas uma folha, de modo didático.

Para ilustrar melhor a utilidade da notação, segue um diagrama simples: a figura 2.6 retrata uma organização com ao menos um coordenador e um programador (ou conjunto de programadores), executando um conjunto tarefas com a finalidade de oferecer

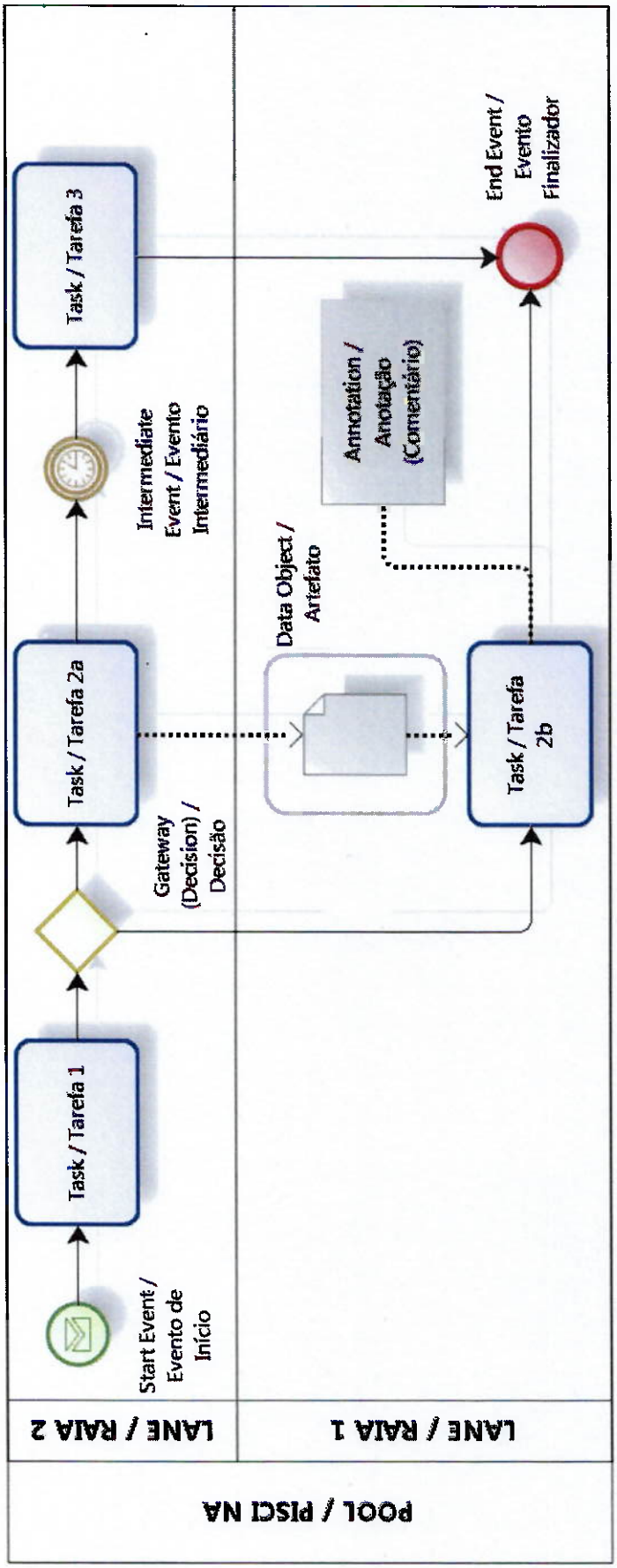

Figura 2.5: Exemplo de Fluxo de diagrama BPMN

suporte a um cliente. À esquerda, uma notificação por *e-mail* é recebida pelo coordenador (símbolo verde com uma carta dentro). Após a recepção é feita uma avaliação do tipo de suporte necessário para a dúvida. Caso já seja uma tarefa recorrente, é interessante repassar o contato do programador para auxiliar segundo as normas já estipuladas. Em caso contrário, o coordenador cria uma guia para a correta comunicação durante o contato com o cliente. Havendo disponibilidade, dependendo de quem está a cargo do atendimento, são realizadas várias iterações até que haja êxito na ajuda e a tarefa seja finalmente concluída.

Caso seja notado que alguma melhoria possa ser adicionada ao fluxo, é permitido adicionar ou eliminar uma ou mais tarefas e fluxos de decisão para cobrir casos não esperados, por exemplo.

2.6 CONSIDERAÇÕES FINAIS DO CAPÍTULO

A notação BPMN, por sua vasta variedade de símbolos, especificação detalhada, flexibilidade na montagem dos diagramas e possibilidade de criação de elementos novos para suprir necessidades específicas é adequada para aplicação na especificação de processos de áreas distintas, como *software* embarcado. A favor ainda, há a possibilidade de gerar o diagrama final em formatos populares de documento. A ferramenta *Bizagi Process Modeler* mostrou-se agradável de se utilizar, por sua interface intuitiva, bastante visual e detecção de erros a cada vez que o arquivo é salvo, proporcionando ao autor criar e alterar os diagramas apresentados no trabalho sem consumo de tempo para aprendizado da ferramenta ou de suas opções mais avançadas. Como contraponto da ferramenta, tem-se a obrigatoriedade de utilizar o sistema operacional Windows para sua instalação e utilização, apesar de ser uma ferramenta gratuita.

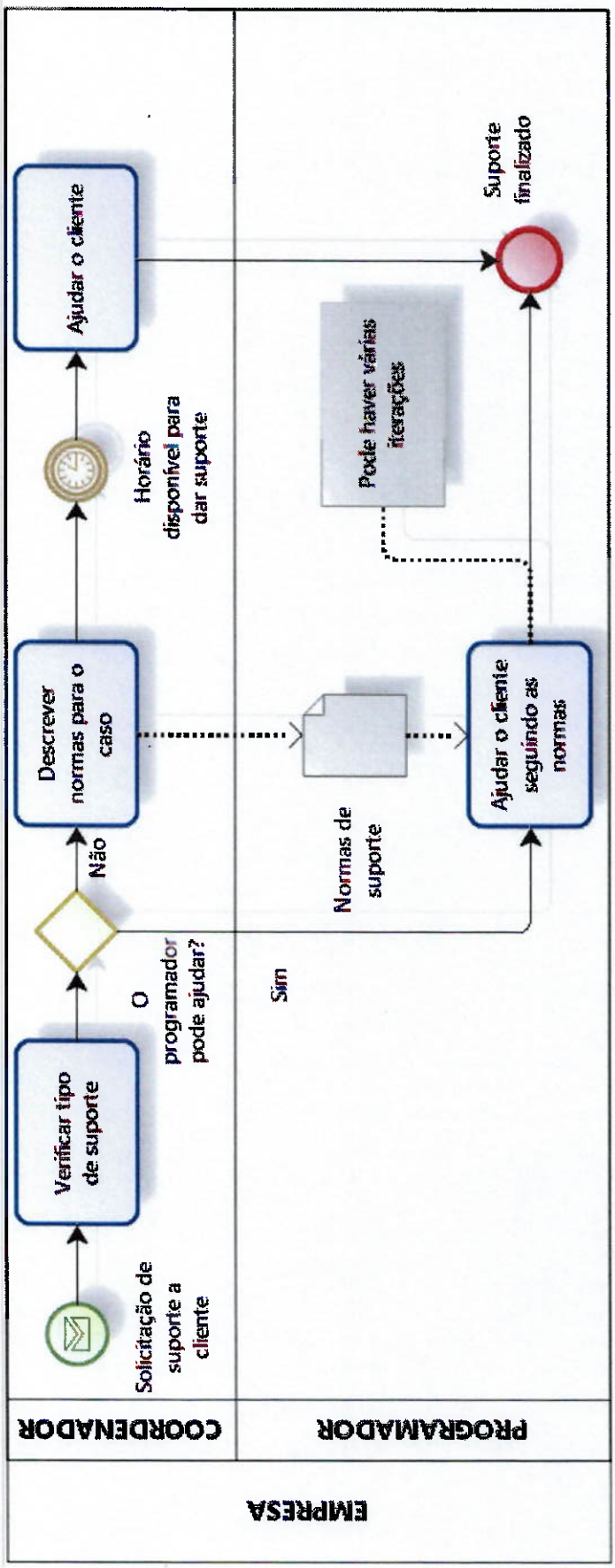

Figura 2.6: Exemplo Prático de Fluxo de diagrama BPMN

3 *SOFTWARE* EMBARCADO

3.1 HISTÓRICO DE *FIRMWARE* E *HARDWARE*

Componentes essenciais dos sistemas embarcados de alto desempenho e presente em grande partes dos aparelhos de telefonia celular, os microcontroladores e microprocessadores possibilitam diversas personalizações de equipamentos eletrônicos devido à sua flexibilidade e rapidez em receber alterações.

Processadores são a principal unidade funcional das placas, cuja missão consiste em executar instruções e manipular dados (GANSSLE et al., 2008a). Equipamentos eletrônicos possuem ao menos um processador mestre, podendo coordenar o funcionamento de outros processadores secundários, denominados também como escravos, que executam tarefas mais específicas.

Inicialmente, microprocessadores continham um mínimo de memória e componentes de entrada e saída de sinais, enquanto microcontroladores possuíam maior memória e um número maior de portas para tratamento de sinais. Atualmente, torna-se difícil diferenciar ambos devido à melhora significativa nas características do componente de menor porte, ou seja, os microcontroladores.

De acordo com Muranko e Drechsler (2006), sistemas embarcados (ou embutidos) são caracterizados pela presença de componentes de *software* e de *hardware* integrados, por exemplo, em carros e equipamentos domésticos. Deve-se enfatizar que cerca de 79% de todos os processadores utilizados estão na área dos sistemas microprocessados e que atualmente, uma casa comum possui, aproximadamente, 50 sistemas embarcados funcionando nela (SRINIVASAN; DOBRIN; LUNDQVIST, 2009). Foi observado também o aumento da participação do *software* nas indústrias de telecomunicações, defesa (GUSTAVSSON; AXELSSON, 2011), entretenimento (câmeras digitais), máquinas industriais e sensores (MITSUI; KAMBE; KOIZUMI, 2009), agora sendo constituído por sistemas e sub-sistemas, em parte devido à melhora do *hardware* e das necessidades dos clientes.

Meliones (2010) menciona que a terminologia de sistemas embarcados refere-se a sistemas digitais que incluem desenvolvimento simultâneo de *hardware* e de *software*

com aplicabilidades especiais, normalmente controle, automação e comunicação em rede. Torna-se cada vez mais complicada a sua concepção e desenvolvimento devido a suas características individuais de arquitetura e componentes, pois recentemente os avanços tecnológicos na área de sistemas embarcados têm requerido menor tempo de desenvolvimento, maior desempenho, novos métodos, linguagens e ferramentas computacionais (RICCOBENE et al., 2007). A utilização de um paradigma correto no seu desenvolvimento induz à maior eficácia no reuso das plataformas.

Mitsui; Kambe e Koizumi (2009) representam de forma simplificada, na figura 3.1, a evolução do *Software* Embarcado (SE) nos últimos tempos, podendo ser notado, inclusive, pela representação que a complexidade dos sistemas aumentaram consideravelmente, assim como seu desempenho e escala de produção. No início, a configuração utilizada possuía um computador de uso genérico (com capacidade de processar os dados) o qual obtinha sinais vindos de uma placa controladora (com limitado poder de processamento para as necessidades da época) e seus periféricos instalados (interfaces serial e paralela, por exemplo).

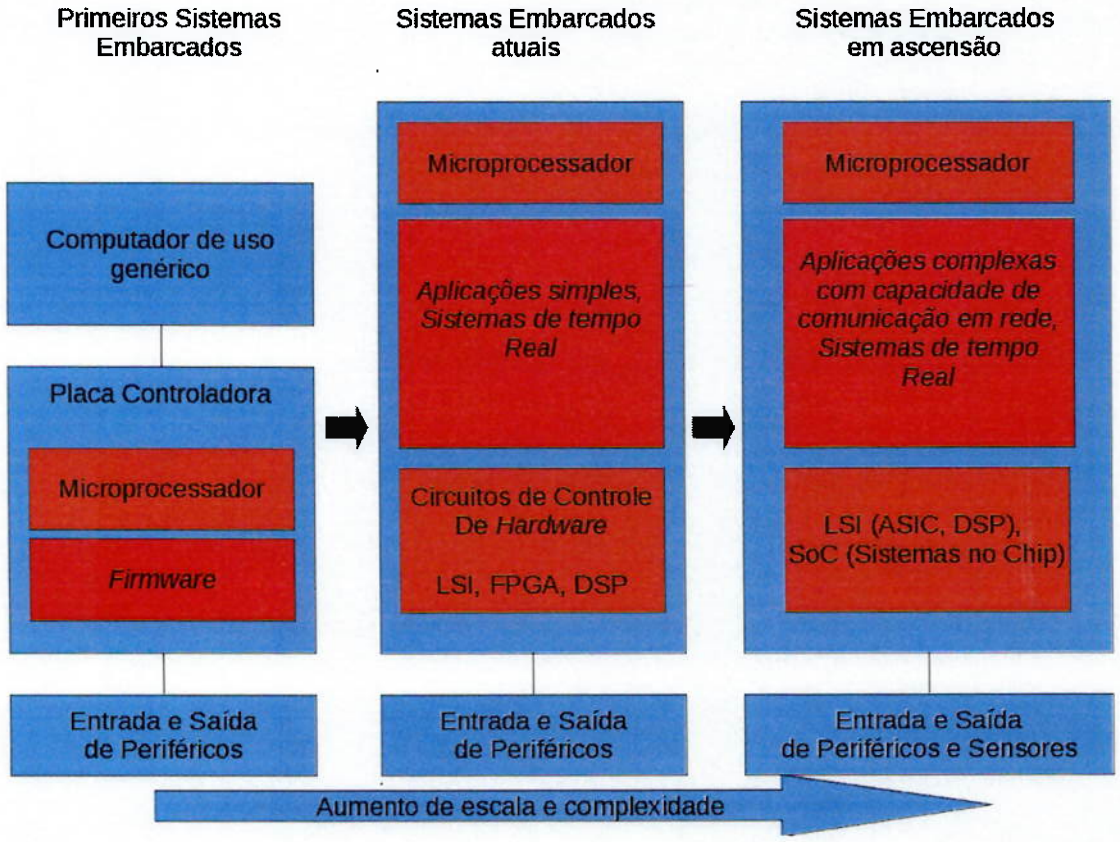

Figura 3.1: Breve história da evolução dos sistemas embarcados de Mitsui; Kambe e Koizumi (2009)

Atualmente já é realidade a presença de autonomia nos componentes, sem precisar de um computador auxiliar para processamento (dependência de aplicações externas). Além disso, alguns periféricos e sensores também possuem inteligência para se comunicar com o microcontrolador, de modo estável e confiável.

Segundo a Lei da Funcionalidade Observada mencionada por Woodward (2007), enquanto a utilidade de um eletrônico aumenta linearmente, a quantidade de transistores em um processador aumenta exponencialmente, resultando em uma lacuna de criação (*Design Gap*), na qual aumentam os custos devido à diferença de evoluções e de rápida obsolescência dos equipamentos.

3.2 TIPOS, CARACTERÍSTICAS E APLICAÇÕES

Das várias centenas de processadores disponíveis no mercado (em que não há clara dominância de qualquer que seja), em várias disposições e formatos (encapsulamentos), existem agrupamentos chamados de arquitetura. A denominação arquitetura associa componentes que obedecem ao mesmo formato do conjunto de instruções (código de máquina), podendo ser executado o mesmo código por todos os elementos do grupo, desde que respeitadas as características de formato.

Algumas das arquiteturas existentes e aos menos um de seus fabricantes são listadas a seguir (GANSSLE et al., 2008a):

AMD - AMD

Possui diferentes arquiteturas, baseadas principalmente em 32 e 64 *bits*.

ARM - ARM

Criada em 1983, baseada em 32 *bits*.

C16X - Infineon

Criada em 1993, baseada em 16 *bits*.

ColdFire - Freescale

Criada em 1994, baseada em 8 *bits*.

I960 - Vmetro

Criada em 1984 e fabricado até o começo da década de 1990, era baseada em 32 *bits*

e sua velocidade podia variar entre 10MHz e 100MHz.

M32/R - Renesas

Criada especialmente para ambientes industriais e veiculares, é baseada em 32 *bits*.

M Core - Freescale

Criada em 1997, baseada em 32 *bits*.

MIPS32 e MIPS64 - MIPS Technologies

Criada em 1981, baseada em 32 e 64 *bits*.

NEC - NEC Corporation

Criada em 1983, passou por várias gerações.

PowerPC - IBM

Criada em 1991, atualmente é baseada em 64 *bits*.

68k - Freescale

Criada em 1979, baseada em 16 e 32 *bits*.

SuperH (SH) - Hitachi

Criada no início da década de 90, baseada em 32 *bits*.

SHARC - Analog Devices

Criada no início da década de 90, baseada em 32 *bits*.

strongARM - Intel

Criada no final da década de 90, seu processamento chega a 300 MHz.

SPARC - Sun Microsystems

Criada em 1987, baseada atualmente em 64 *bits*.

TMS320C6xxx - Texas Instruments

Criada em 1983, é atualmente baseada em 32 *bits*.

x86 - Intel e AMD

Criada em 1978, era baseada em 16 *bits*, tendo evoluído posteriormente para 32 e agora 64 *bits*.

TriCore - Infineon

Criada em 1999, baseada em 32 *bits*.

Podendo trabalhar em frequências que vão desde dezenas de *kiloHertz* (kHz) a unidades de *gigaHertz* (GHz), os microprocessadores também têm diversas capacidades em suas memórias internas (*Random Access Memory* (RAM) e *Read Only Memory* (ROM)), podendo possuir módulos já embarcados nas próprias pastilhas (*Ethernet, Universal Serial Bus* (USB) com funcionalidade *On-The-Go* (OTG), Serial, Paralelo e *Inter-Integrated Circuit* (IIC) (mais conhecido como I^2C), entre outros).

O autor destaca neste momento a arquitetura ARM, por sua crescente presença nos equipamentos que exigem alto desempenho, utilizados hoje em dia. Algumas de suas características reconhecidas pelo mercado são:

- Desempenho;
- Tamanho diminuto;
- Baixo consumo de energia.

Da mesma forma, unidades de controle eletrônico são aplicadas hoje em dia em aplicações críticas, como controle de baterias ou controle de energia de veículos híbridos (KRAMMER et al., 2010). Qualquer falha pode resultar em danos ao ambiente ou propriedades e acidentes com pessoas, por isso é necessário assegurar a qualidade dos componentes eletrônicos, além da implementação de *hardware* e *software* (como por exemplo através das normas *International Electrotechnical Commission* (IEC) 61508 - indústria em geral - e *International Organization for Standardization* (ISO) 26262 - adaptação para o setor automotivo), possibilitando ao mesmo tempo que seja flexível a ponto de poder ser adaptado a outras plataformas com baixo custo. Inclusive na indústria automotiva existe o dilema para os sistemas: alta complexidade, muitas funcionalidades, várias possibilidades de produtos (personalizações) e confiabilidade cada vez maiores, enquanto os custos e tempo de projeto devem ser menores.

Além disso, os sistemas de tempo real - incluindo-se os Sistemas Operacionais de Tempo Real - *Real-Time Operating System* (RTOS) - são sistemas nos quais a resposta esperada a um estímulo deve ser apresentada dentro de um tempo pré-estipulado para tal. Os sistemas embarcados são os que mais se aproximam desta realidade atualmente, sendo capazes de receber um estímulo qualquer e tratá-lo em tempo hábil (GANSSLE-89, 2008b). Os tipos de sistemas de tempo real devem ser mencionados como *Hard-RTOS* (do inglês, 'duro') e *Soft-RTOS* (do inglês, 'macio'), sendo que o primeiro possui tempos totalmente

restritos para resposta (sistemas de alta criticidade e confiabilidade, com índices de falha menores que um em milhões) e o segundo possui maior tolerância quanto ao tratamento dos sinais (podendo em alguns casos até perder informações sem comprometer a operação ou objetivo final).

Para facilitar o monitoramento dos eventos que podem desencadear respostas, são implementados os tratamentos de interrupção (muitas vezes com relação de prioridade entre elas, caso haja mais de uma) ao invés de constante monitoramento em laços dos periféricos do equipamento (*loop* principal). Tal arquitetura, apesar de necessitar de melhor controle no fluxo da aplicação, complexa elaboração de projeto e capacitação dos desenvolvedores, permite obter um código mais enxuto, mais rápido de depurar e desenvolver, mais econômico energeticamente e portanto, menos custoso para desenvolvimento e produção, mais rápido para entrega ao mercado e melhor aceito pelos clientes por seus benefícios apresentados.

Os variados sistemas embebidos podem ser categorizados em (MITSUI; KAMBE; KOIZUMI, 2009):

Isolado

sem necessidade de comunicação com servidores: câmera digital, aparelhos caseiros (autônomos);

Terminal com interface para comunicação

telefone celular, televisão com acesso à rede;

Distribuído em rede

rede caseira, rede interna de um carro, sensores em rede.

Por mais que os componentes tenham evoluído, para utilizá-los é necessário difundir os conceitos às novas gerações de desenvolvedores. Mitsui; Kambe e Koizumi (2009), percebendo a grande defasagem entre o ensino nas escolas e faculdades e os desafios encontrados na indústria, propuseram uma atualização na forma de passar o conhecimento aos alunos. A figura 3.2 representa um ambiente para a programação de componentes *Field-Programmable Gate Array* (FPGA) instituído durante o experimento realizado. À esquerda está ilustrado o ambiente em computadores de laboratório e à direita pode-se observar o equipamento e interfaces utilizadas durante as aulas.

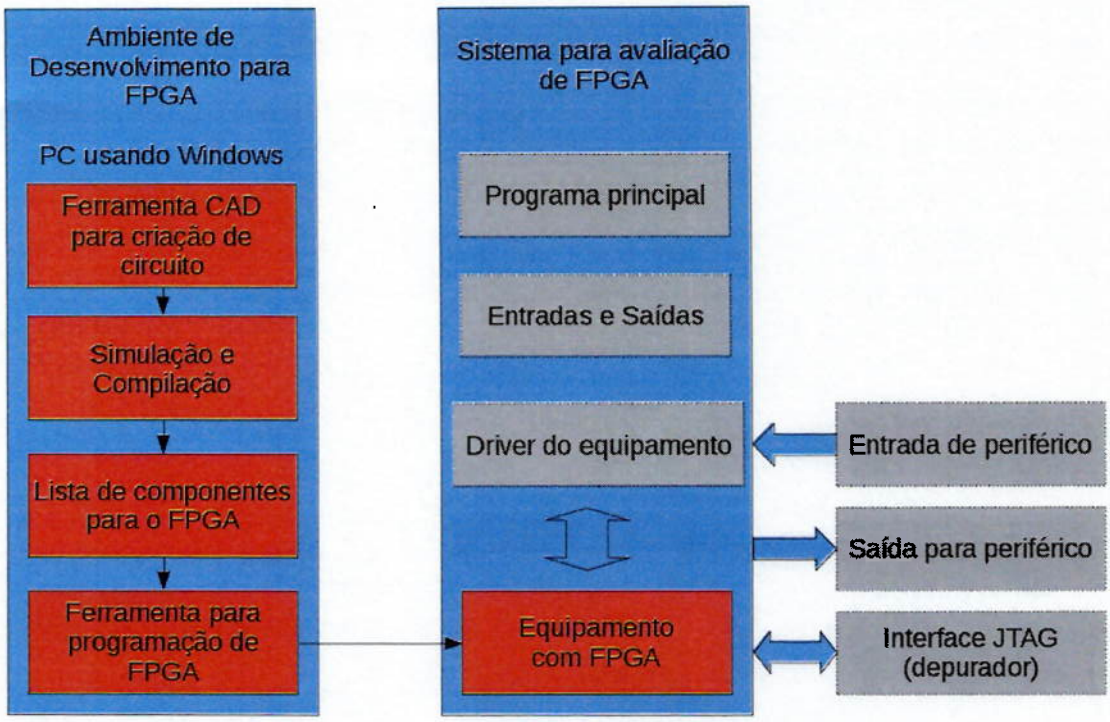

Figura 3.2: Experiência de desenvolvimento de *hardware* embarcado de Mitsui; Kambe e Koizumi (2009)

Em seguida, através da figura 3.3 é apresentado um modo de se conceber *software* para os sistemas embarcados. À esquerda ilustrando o ambiente em computador de laboratório para o desenvolvimento de aplicações em Linguagem C e à direita decompondo o microprocessador em blocos para melhor visualização de sua estrutura.

Finalmente, por meio da figura 3.4, mostra-se a visão simplificada dos experimentos de *software*, de *hardware* e o experimento final. Nos modelos apresentados, desenvolver com RTOS provou-se mais vantajoso, notou-se melhora ao utilizar FPGA e ambas as interfaces e coordenação de seu desenvolvimento foram sincronizadas em tempo e informações. Problemas de especificação ou situações de problema de integração dos módulos não foram observados sob essa arquitetura.

A maioria dos sistemas são programados em linguagens de baixo nível (entre elas *Assembler* e C), principalmente por causa de necessidade de desempenho ou consumo de memória (BEHALEK; SALOUN, 2010), o que demanda maior tempo de desenvolvimento. Ao mesmo tempo, o desejo para a rápida criação destes aumenta a cada dia, devido à necessidade de colocar produtos no mercado, o quanto antes e a menores preços.

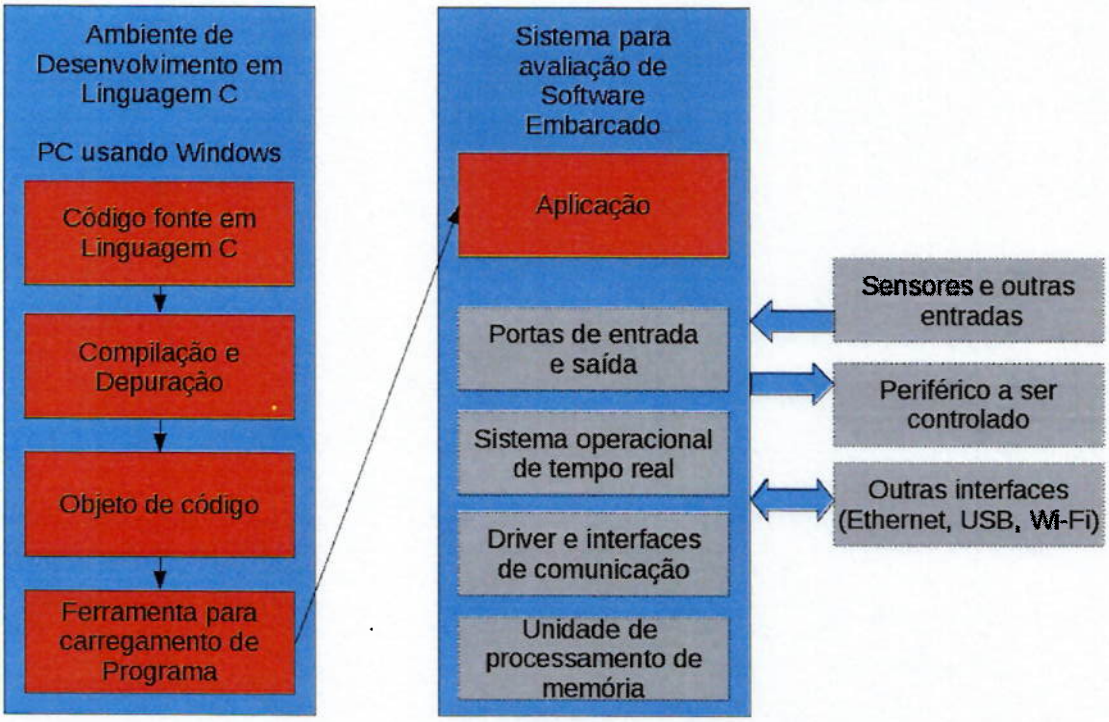

Figura 3.3: Experiência de desenvolvimento de *software* embarcado de Mitsui; Kambe e Koizumi (2009)

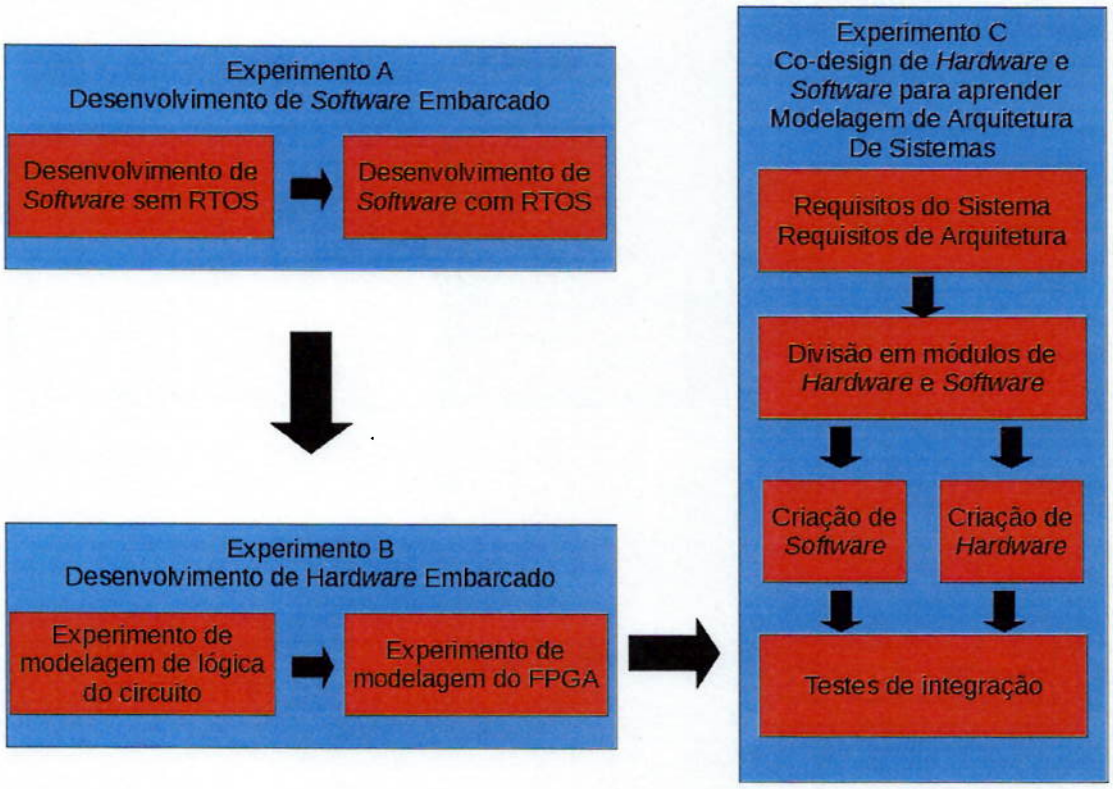

Figura 3.4: Experiência de ensido de *software* embarcado de Mitsui; Kambe e Koizumi (2009)

3.3 PROCESSOS DE PRODUÇÃO DE SISTEMAS EMBARCADOS

As indústrias automotiva, de telecomunicações e de automação de processos possuem o *software* embarcado como parte de seus processos de criação (AXELSSON, 2011). Sua evolução possibilitou um enorme número de computadores menores ligados em rede, acumulando milhões de linhas de código para sua concepção.

A criação de sistemas embarcados demanda considerável e variada gama de especializações para seu desenvolvimento, entre elas (MELIONES, 2010):

- Análise e projeto de arquitetura de sistemas;
- Análise e projeto de *hardware*;
- Adaptação de sistemas operacionais para o *hardware* alvo;
- Programação de adaptadores para os periféricos (do inglês, *driver*);
- Desenvolvimento de aplicativos (*software*).

Enquanto isso, Riccobene et al. (2007) declaram que a área de sistemas embarcados envolve diferentes visões: modelagem de aplicações, montagem de componentes físicos, simulação e verificação de todos os componentes envolvidos (tanto de *hardware* quanto de *software*) e o conjunto da obra. Parte do desafio é incluir tudo isto em um único componente (*chip*) ou no mínimo espaço possível. Uma outra vertente para a qual está se direcionando o desenvolvimento de aplicações em sistemas é o guiado por modelagem - *Model-Driven Development* (MDD) - (LIGGESMEYER; TRAPP, 2009), no qual são identificadas - através de abordagem iterativa - características, desafios, testes e soluções para assegurar qualidade, robustez e confiabilidade ao produto final.

Igualmente necessários, tanto para engenheiros da área de *software* e de *hardware* embarcados (GANSSLE et al., 2008a), o aprendizado e entendimento de padrões para diagramas e de símbolos são primordiais para que a arquitetura criada pela equipe possa ser passada a outras equipes (sejam internas ou externas à empresa) sem danos ou demora nas interpretações. Deve ser citado então, como exemplo, a equipe que concebe um *hardware* (entenda-se: ao menos uma placa com microprocessador ou microcontrolador que possa interagir com o meio externo a ela) e passar para a equipe de *software*

desenvolver o programa (conjunto de instruções) desta plataforma sem haver problemas de compatibilidade. Para auxiliar a tarefa, alguns diagramas podem ser incluídos nesta categoria, por exemplo:

Diagramas de bloco

Envolvem atividades macro de modo a facilitar a visualização generalizada.

Diagramas Esquemáticos

Descrevem os componentes utilizados e com quais outros devem estar conectados, mas representado ainda com abstrações.

Diagramas de Conexões

Apresentam maior detalhe que os diagramas esquemáticos, mas ainda com abstração sobre os desenhos reais de implementação da placa.

Diagramas Lógicos

Ilustram o fluxo de execução do programa gravado no microprocessador.

Diagramas de Temporização

Representação de ações que devem tomadas conforme é passado o tempo e ocorrem estímulos no sistema.

Segundo Garcia e Andrea (2010), existem outros fatores a se considerar que podem interferir no sucesso de qualquer projeto da área:

- Coordenação dos subprocessos envolvidos (mecânicos, elétricos e de programação) com qualidade;
- Antigamente os sistemas eram inicialmente baseados no *hardware* (mecânica e elétrica). Neste caso, o *software* era iniciado apenas quando mudanças necessárias na parte elétrica já se tornariam dispendiosas.

Entre as limitações impostas pelos sistemas embarcados estão o tempo de desenvolvimento escasso, eficiência necessária do código e eficácia dos testes e depuração, que devem ser realizados dentro de suas restritas capacidades (BARISIC et al., 2007). Ao mesmo tempo, empresas envolvidas na área de *Software* (SW) embarcado buscam cada vez mais gerenciar o aumento de funcionalidades e suas complexidades, enfrentando

competitividade dos concorrentes a prazos cada vez menores e orçamentos mais enxutos (SRINIVASAN; DOBRIN; LUNDQVIST, 2009).

Como ocorre em parte das profissões da área, o desenvolvimento possui desvantagens. Entre os contra-pontos - que já podem ter sido modificados pelo tempo - levantados pelos engenheiros de sistemas embarcados, devem ser mencionados (GRENNING, 2007):

- Impossibilidade de testar o código sem o *hardware*;
- Programação feita em Linguagem C (mais trabalhosa);
- Tempo de carga e compilação demorado;
- *Hardware* caro e às vezes em menor número do que o necessário;
- O código deve ter tamanho pequeno e ser eficiente;
- Falta de tempo para escrever os testes;
- Tempo necessário para escrever a camada adaptadora para os periféricos (do inglês, *driver*).

Por necessitar de uma plataforma confiável para ser efetivamente testado, há adicional demora pela espera de um protótipo e pela necessidade de dividir o equipamento entre os colegas. Muito tempo de desenvolvimento pode ser perdido caso a placa não esteja correta. As dificuldades neste momento são descobrir onde há erro (se no *hardware* ou no *software*) e como proceder para sua correção (manutenção da placa e ajustes para a produção das próximas versões).

Enquanto isso, na seção de mecânica e eletrônica, os desafios são cada vez mais relacionados a (LIGGESMEYER; TRAPP, 2009):

- Minimizar o custo;
- Reduzir o consumo de energia;
- Reduzir o tamanho;
- Diminuir o peso do produto final;
- Aumentar a eficiência dos recursos aparelho;

- Aumentar o poder de processamento para o caso de uso de plataformas e sistemas operacionais complexos.

Durante o processo de criação de um sistema embarcado, a solução pode, preferencialmente, ter um primeiro modelo funcional ou protótipo que contenha as partes mais críticas do projeto e a partir deste acrescentar melhorias e outras funcionalidades. Durante todos os testes, o núcleo crítico será o mais testado, tornando-se mais estável (BEHALEK; SALOUN, 2010).

É registrado por Pillat; Oliveira e Fonseca (2012) que a qualidade de um sistema de *software* é, por muitas vezes, associada com a qualidade de seu processo gerador. Tal processo descreve todas as atividades necessárias para a criação do produto final, além de especificar os papéis de todos os envolvidos nas tarefas, assim como os documentos desejados ao final delas, definindo o melhor fluxo para tal. Os fluxos de tarefas, largamente observados e utilizados nos processos de negócio, se desenvolveram rapidamente nos últimos anos (WU; ZHANG, 2009) sendo, a partir de uma análise mais detalhada, possível automatizar os processos já existentes, permitindo inclusive, também coordenar pessoas e ferramentas para a finalização de uma tarefa qualquer (por muitas vezes em ambiente computacional), em menor tempo.

Por poder integrar vários departamentos, cada um com suas complexidades próprias, sistemas empresariais requerem que o fluxo das operações estejam em acordo para o sucesso no resultado, ainda permitindo flexibilizações para o caso de mudanças nas operações ou decisões estratégicas. Devido ao aumento de pressões vindas dos setores comercial e técnico (WOODWARD, 2007), o desenvolvimento de sistemas embarcados, que a cada dia aumenta em muito sua complexidade, precisa de otimização para o cumprimento das metas.

Em seu trabalho, Mitsui; Kambe e Koizumi (2009) apresentam uma estimativa de custos indicando que o desenvolvimento do *Software* Embarcado de um projeto de sistema é responsável por aproximadamente 40% de seu custo total, representado também por materiais, protótipos, testes em laboratórios e licenças de programas, por exemplo. Levando-se em conta as cifras atualmente investidas em desenvolvimento e pesquisa na área de tecnologia, qualquer possibilidade de redução de custos representa uma economia considerável para qualquer negócio, podendo ser aplicada em novos equipamentos, aplicações, ferramentas ou qualificação de pessoal, por exemplo.

De acordo com Antonio; Ferrari e Fabbri (2012), representar as arquiteturas e adequar o processo de desenvolvimento de *software* é um passo fundamental para a melhora na qualidade dos produtos entregues. Apesar dos avanços registrados nos últimos anos na área de sistemas embarcados críticos (a citar aviação e medicina, que podem resultar em fatalidades), ainda falta a definição de um modelo adequado para seu desenvolvimento. Se, por um lado, os engenheiros necessitam criar os produtos cada vez mais rápido por causa da demanda mercadológica, por outro, também torna-se necessário garantir a qualidade do mesmo na operação em campo. O sucesso nesta área está no correto balanceamento entre essas atividades, por meio da eliminação de atividades desnecessárias e foco nas tarefas prioritárias.

Em 46,5% dos casos estudados pelos mesmos Antonio; Ferrari e Fabbri (2012) é feita a menção da linguagem *Unified Modeling Language* (UML) como técnica para modelagem da arquitetura de sistema embarcados, ainda que seja considerada uma modelagem informal para esse propósito.

Pode ser declarado que já existem diversas tentativas para a criação de fluxos de desenvolvimento e metodologias para o *sofware* embarcado (KASHIF et al., 2009), entre elas merecem destaque alguns passos para se implementar um processo, devendo-se prestar atenção especialmente nos itens a seguir (JUN; RUI; YI-MIN, 2007):

- Investigar;
- Especificar;
- Escolher as ferramentas adequadas;
- Treinar os funcionários;
- Realizar tarefas, sob monitoramento e acompanhamento.

3.4 CONSIDERAÇÕES FINAIS DO CAPÍTULO

Presente em considerável parte dos sistemas utilizados atualmente, os componentes eletrônicos necessitam de desenvolvimento focado e orquestrado para redução do tempo de mercado, da incidência de falhas elétricas e de comportamentos instáveis em condições de operação.

Por experiência do autor, o mau gerenciamento de sua produção leva a atrasos consideráveis no cronograma inicial, podendo tornar o projeto um fracasso por ser atrasado em relação à concorrência ou às necessidades mercadológicas. Outro ponto a ser notado, é a portabilidade da plataforma, que pode tornar-se obsoleta, para acompanhamento das evoluções do mercado. Uma vez que os elementos de *hardware* e *software* estejam fortemente associados à plataforma, sua evolução pode tornar-se inclusive mais demorada que o início de um novo projeto.

4 ESPECIFICAÇÃO DO PROCESSO DE DESENVOLVIMENTO DE SISTEMAS EMBARCADOS

A partir dos pontos fortes, pontos fracos, benefícios e recomendações já apresentadas, o autor verificou a necessidade de especificar um processo para Sistemas Embarcados, utilizando a notação BPMN por sua versatilidade e facilidade de entendimento.

Neste capítulo é apresentada uma proposta para o projeto para sistemas embarcados, a qual é sujeita a evoluções e refinamentos por meio de melhorias e técnicas obtidas da literatura e da experiência profissional do autor. Os passos seguidos para sua criação estão divididos entre as seções:

1. PASSO INICIAL: CONCEPÇÃO DO *CO-DESIGN* NO BPMN
2. PRIMEIRA EVOLUÇÃO: PAPÉIS E ATRIBUIÇÃO DE TAREFAS
3. SEGUNDA EVOLUÇÃO: METODOLOGIAS ÁGEIS E ORIENTAÇÃO A TESTES
4. TERCEIRA EVOLUÇÃO: LEVANTAMENTO DE REQUISITOS
5. PRIMEIRO REFINAMENTO: GERENCIAMENTO E MONITORAMENTO DO PROCESSO E SEUS RISCOS
6. SEGUNDO REFINAMENTO: REFATORAÇÃO
7. TERCEIRO REFINAMENTO: DOCUMENTAÇÃO
8. PROPOSTA DE MODELO FINAL

Após todos os incrementos justificados e aplicados, chega-se ao modelo especificado final proposto (item 4.8), que pode ser instanciado por meio de sua adequação aos processos já instaurados no ambiente de trabalho.

4.1 PASSO INICIAL: CONCEPÇÃO DO *CO-DESIGN* NO BPMN

Tradicionalmente, o desenvolvimento de *hardware* e de *software* em sistemas embarcados era dissociado logo em seu princípio (GUPTA, 2001), podendo os engenheiros e arquitetos de cada área criarem as soluções para seus problemas sem interagir com as áreas afins. Ao final de ambos os desenvolvimentos pode-se ter a desagradável situação na qual os sistemas não se integram satisfatoriamente, acarretando em custos altíssimos, baixa confiabilidade, reuniões para readequação dos módulos, além da demora na entrega do produto final.

Para solucionar, ou ao menos minimizar os problemas causados por tal procedimento, tanto Gupta (2001) - figura 4.1 - quanto Chong e Xingchuan (2009) - figura 4.2 - propõem, de forma simplificada, tarefas nas quais tanto desenvolvedores de *hardware* quanto de *software* tenham suas individualidades respeitadas, contudo tornam necessária a convergência de ambas as equipes na forma com que as partes se conversam. Algumas tarefas de modelagem podem e devem ser executadas por ambas as equipes em conjunto, para simplificar procedimentos e acoplamentos, além de definir melhor as metas.

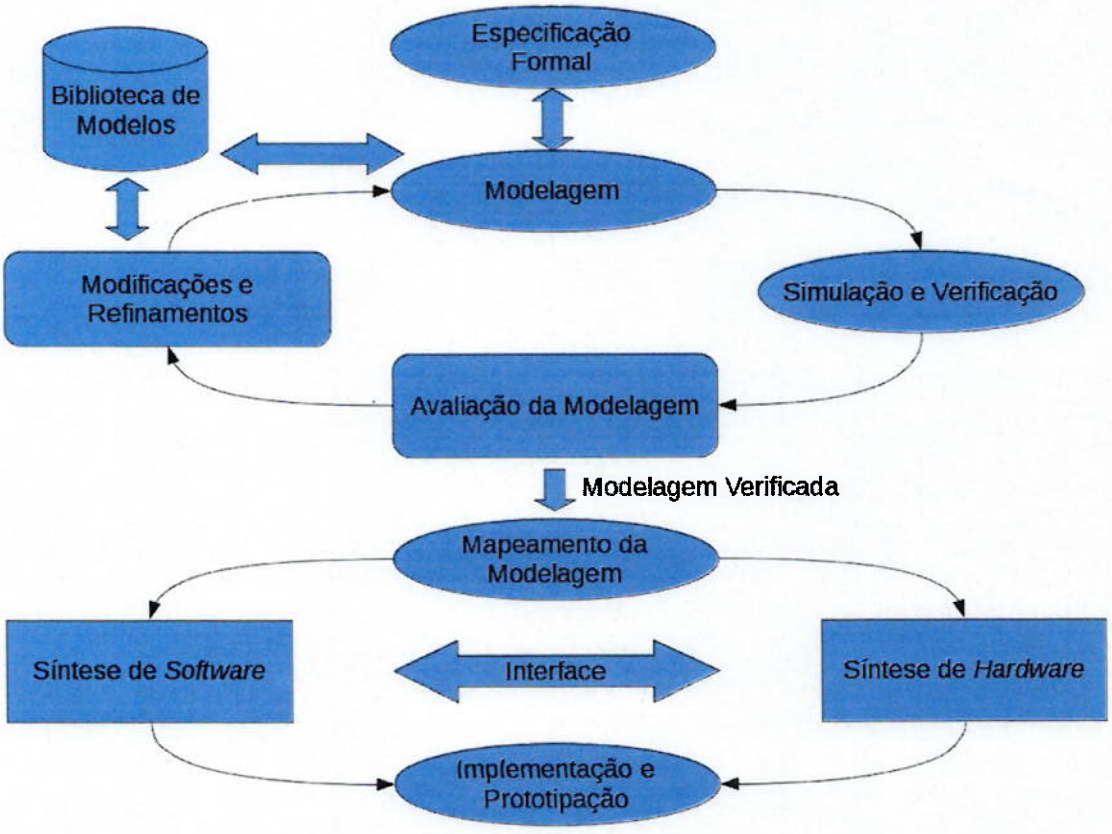

Figura 4.1: Proposta de *Co-design* de Gupta (2001)

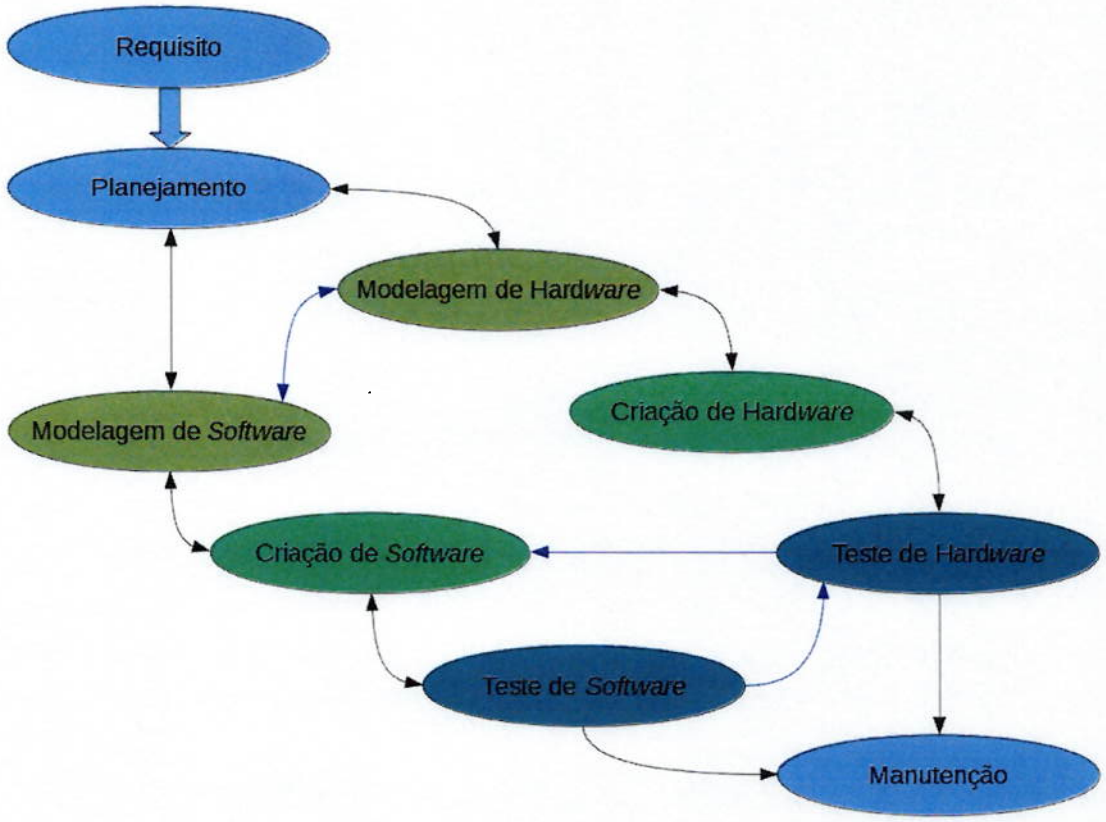

Figura 4.2: Proposta de *Co-design* de Chong e Xingchuan (2009)

Adicionalmente, para diminuir os custos de desenvolvimento a figura 4.3 mostra o método adotado por Mitsui; Kambe e Koizumi (2009) de *co-design* entre *hardware* e *software*, no qual módulos são criados e testados em paralelo.

A partir da união destas propostas mencionadas (que não estão na notação BPMN, objetivo final deste trabalho) - para normalizar as interpretações, facilitando o entendimento dos diagramas e fluxogramas - cria-se o processo embrião ilustrado na figura 4.4, que propõe a concepção de projetos de sistemas embarcados.

No processo inicial o autor une as ideias vistas nesta seção de modo a ter tarefas em comum (compartilhadas pelos membros dos respectivos setores) como Levantamento de Requisitos do cliente, Planejamento e Especificação do sistema, Integração e Validação do mesmo e Manutenção do ambiente junto ao cliente, em suportes pontuais. Simultaneamente, ambos os setores têm suas tarefas individuais derivadas das inicialmente realizadas. A Modelagem a partir da Especificação, sua Implementação e Testes são realizados com base em informações de projetos anteriores, que deixam seu legado armazenado em forma de Bibliotecas para consulta. Os Relatórios de testes são utilizados para verificar erros na implementação e poder corrigir para uma nova bateria de

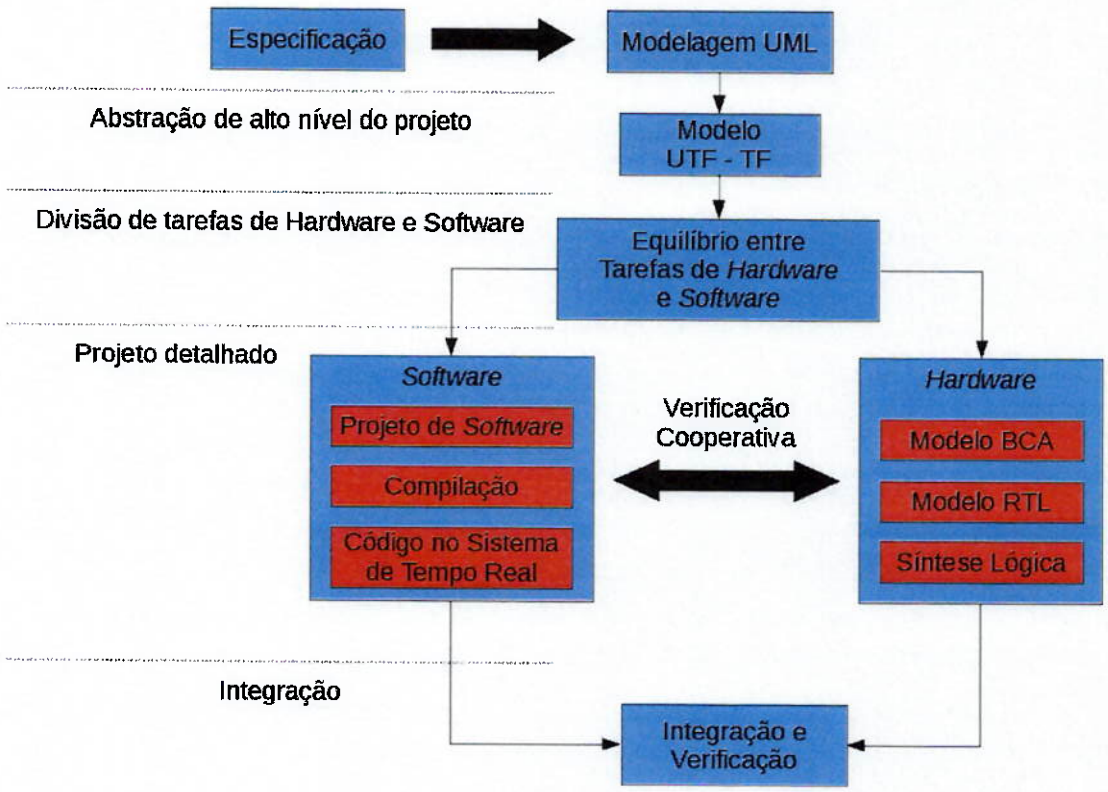

Figura 4.3: Método de *Co-design* de Mitsui; Kambe e Koizumi (2009)

testes. Sua concepção em duas partes, *Hardware* (HW) e SW, sugere diferentes modos de teste, um específico para o nível elétrico (comunicação com o mundo externo, entre módulos, resistência a variações térmicas) e um para o nível lógico (temporização de comunicação, resposta em tempo adequado ao usuário, usabilidade da interface), cada um com critérios de aceitação dependentes da aplicação (rígido se envolver risco de acidentes, ou mais brando caso não seja prejudicial, sempre definido pela análise da equipe acerca da aplicação).

4.2 PRIMEIRA EVOLUÇÃO: PAPÉIS E ATRIBUIÇÃO DE TAREFAS

Cerca de 50% do tempo gasto por arquitetos é gasto comunicando-se com os envolvidos no projeto (AXELSSON, 2011). É considerável a possibilidade de obter benefícios caso sejam criadas rotinas para assegurar a qualidade, pois aumentam a eficiência e eficácia, além de diminuir partes mais trabalhosas de realinhamento de informações por conter elementos similares em vários projetos. Um procedimento bem estipulado informa a todos quais são

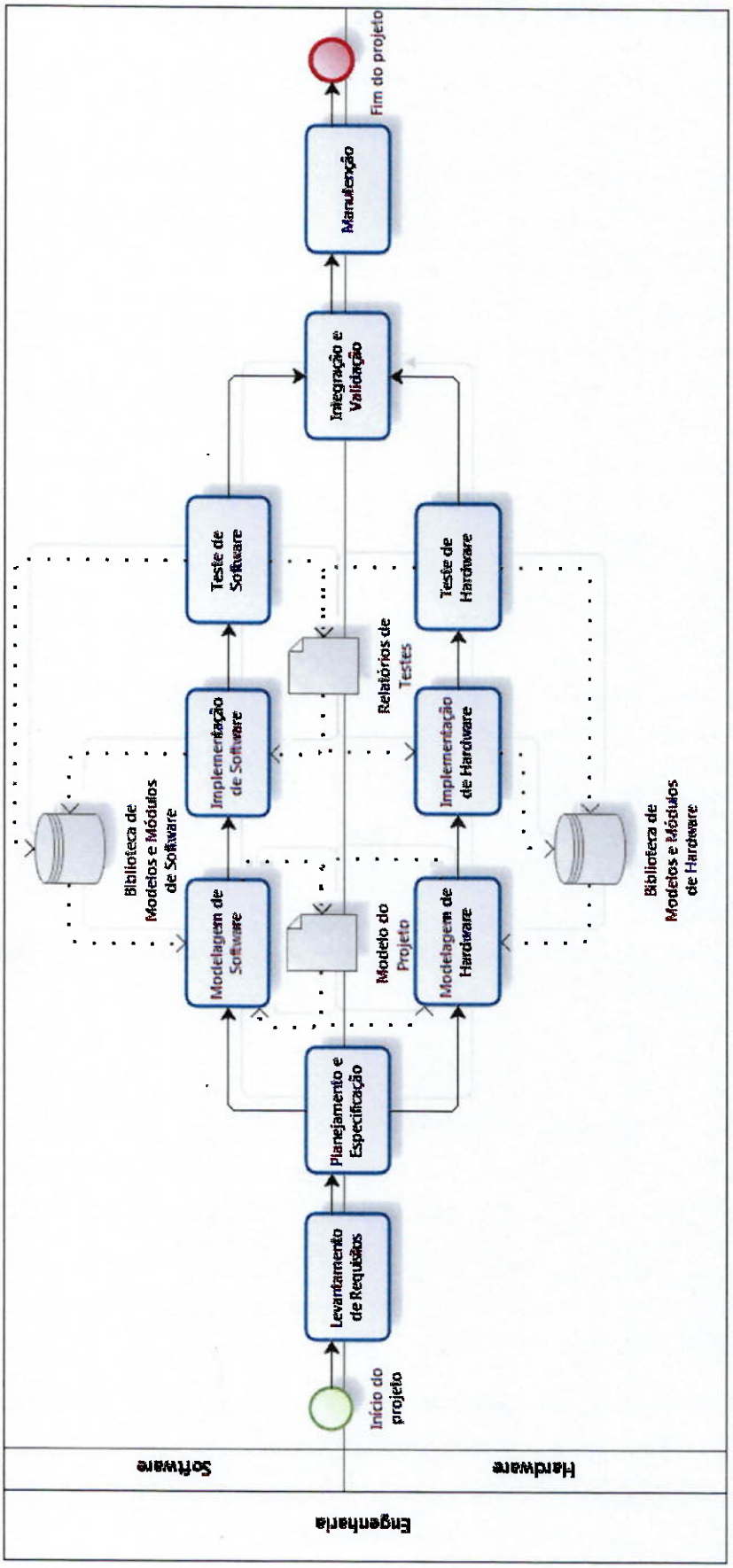

Figura 4.4: Versão inicial de *Co-design* adaptada

seus papéis, possibilitando a inserção e proposta de melhorias. Deste modo, os envolvidos podem agilizar as etapas, ou ao menos aumentar o retorno de informações solicitadas.

Primordial fundamento em qualquer projeto com diversas equipes - em um mesmo local de trabalho ou espalhadas pelo mundo inteiro - nos dias atuais, a comunicação entre as mesmas faz-se necessária para que todos os requisitos sejam cumpridos por completo e em total sincronismo, reduzindo também o tempo de entrega do projeto, um dos fatores determinantes para seu sucesso.

O grande aumento na complexidade dos *softwares* tem exigido cada vez mais a presença de engenheiros desta área para a criação e manutenção dos sistemas, segundo Mitsui; Kambe e Koizumi (2009). Para estar preparado para tal desafio, o engenheiro que atuar nesta área deve ter conhecimento também de arquiteturas de *hardware* e de sistemas, o mesmo valendo para os engenheiros das outras áreas.

Em seu estudo (figura 4.5), Mitsui; Kambe e Koizumi (2009) declaram os papéis de Arquiteto de Sistemas embarcados e de Engenheiros de *Hardware* (HW) e de *Software* (SW) cada qual com suas atribuições a experiências necessárias para dar andamento ao projeto.

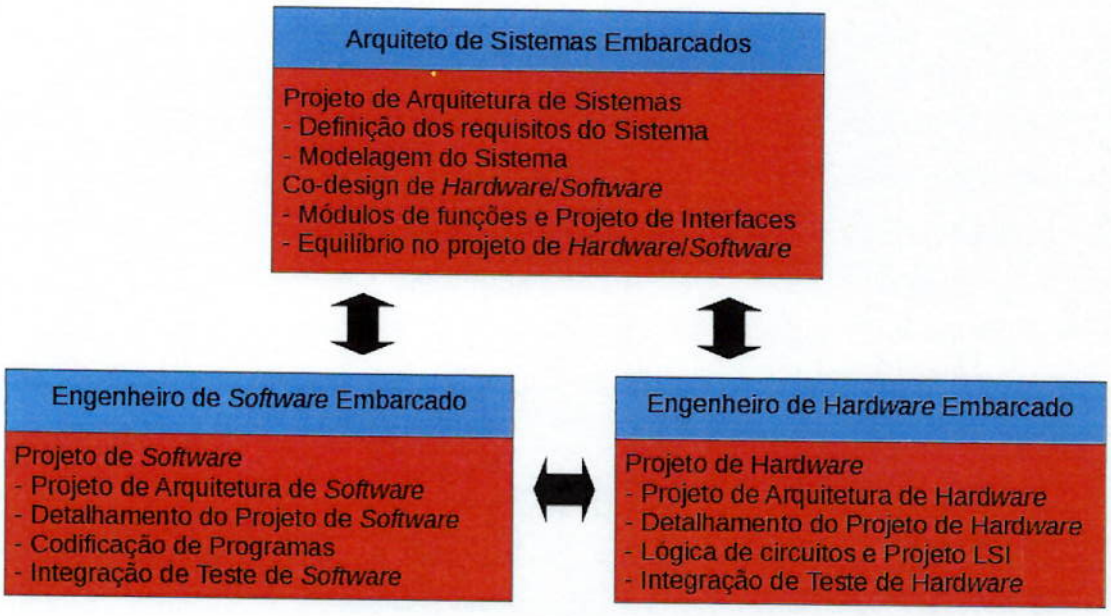

Figura 4.5: Ensino de sistemas embarcados para engenheiros segundo Mitsui; Kambe e Koizumi (2009)

A partir da figura 4.5 apresentada, os engenheiros de sistemas embarcados podem ser classificados, resumidamente, em:

Arquiteto de sistema

experiente em ambas as tecnologias de *hardware* e *software*, analisa os requisitos coletados e realiza a modelagem do sistema e da arquitetura.

Engenheiro de *hardware*

compreende a modelagem da arquitetura de HW e a detalha, criando os circuitos elétricos e elementos relacionados a estes.

Engenheiro de *software*

compreende a modelagem da arquitetura de SW e realiza seu detalhamento, criando os módulos computacionais, funções necessárias e formas para teste.

É possível reconhecer então, a necessidade de ter na equipe pessoas com conhecimentos em diferentes áreas, como levantamento de requisitos, modelagem de sistemas, codificação, lógica de circuitos e integração e testes. Tais tarefas podem ser divididas em inúmeros papéis e serem executados por várias pessoas diferentes ou até pela mesma pessoa, dependendo do tamanho e disponibilidade da equipe.

Para fins de melhor visualização, o modelo proposto pelo autor adiciona diversos papéis, granularizando as funções. O nível de detalhe não é aprofundado, uma vez que muitos detalhes são inerentes ao tipo de equipe.

Os papéis foram criados já dentro de um setor da empresa, no caso Engenharia, para facilitar também a visão gerencial do processo na figura 4.6. Cargos podem ser criados ou multiplicados, dependendo do quadro funcional da fábrica de *software* ou organograma. Assim como aumentar, diminuir e concentrar funções para menos integrantes também é válido, sendo recomendado pelo autor o cuidado para não sobrecarregar os recursos.

Além dos já citados cargos de sistemas, *software* e *hardware*, a experiência em projetos permite ao autor adicionar o Engenheiro de Aplicações, o qual valida o equipamento final por meio de testes (de *hardware* e *software* em separado - mencionados no item 4.1 - e do conjunto final) e interação com o cliente, o Analista de Negócio que representa e avalia as necessidades do cliente e o Gerente de Projetos que monitorará o andamento do processo, fazendo também o controle de quando o projeto deve ser iniciado e finalizado.

Uma característica que deve ser notada neste passo e em todos os outros apresentados é que as equipes de *hardware* e *software* são clientes e fornecedores uma da

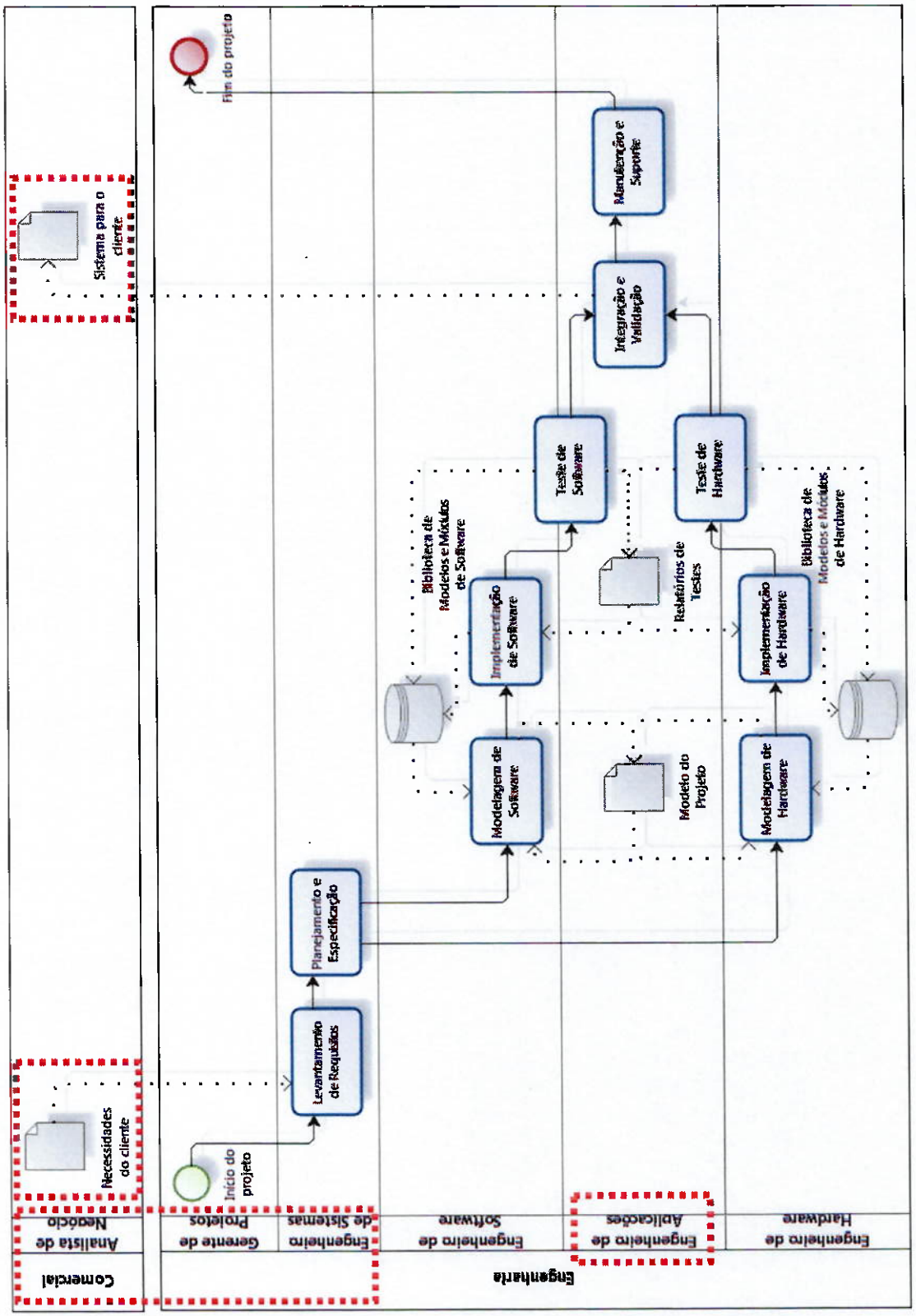

Figura 4.6: Proposta de *Co-design* com a adição dos papéis

outra (fornecem placas funcionais e programas para testes de acordo com os requisitos), sendo necessária a completa simbiose entre todos (SRINIVASAN; DOBRIN; LUNDQVIST, 2009) para o sucesso dos projetos.

4.3 SEGUNDA EVOLUÇÃO: METODOLOGIAS ÁGEIS E ORIENTAÇÃO A TESTES

Entre as diferentes soluções possíveis para problemas de demora nos projetos e notificar sua evolução, pode ser adotada como solução uma metodologia ágil (popular nos últimos tempos, segundo Behalek e Saloun (2010)), na qual deve-se eliminar o quanto antes os riscos e futuros problemas logo no começo do projeto ou em suas primeiras iterações (na criação do modelo ou protótipo).

Segundo Srinivasan; Dobrin e Lundqvist (2009), métodos ágeis têm como promessa reduzir o tempo dos ciclos, propiciando mais rápida agregação de valor para todos os envolvidos no andamento do projeto. Apesar da grande vantagem proporcionada, seu uso ainda é bastante tímido em sistemas embarcados devido a alguns impedimentos, como os de ordem técnica (gerenciamento de requisitos e testes) e de ordem organizacional (adaptação de processos, transferência de conhecimento, mudança na cultura interna e infra-estrutura).

Apesar das evidências em artigos e publicações científicas que empresas de todo o mundo estão rapidamente adotando a metodologia ágil de desenvolvimento de *software* padrão, pouco tem sido notado no que tange ao seu uso no complexo ambiente da indústria de sistemas embarcados. Há mais de uma década, surgiram várias vertentes de métodos ágeis de desenvolvimento que agora estão disponíveis para uso das empresas, as quais são desafiadas devido ao seu modo mais tradicional de concepção de *software*. Pesquisas mencionadas por Salo e Abrahamsson (2008) indicam que aproximadamente 14% das empresas norte-americanas e européias usam processos ágeis na concepção de sistemas embarcados, enquanto outros 19% possuem planos futuros para sua utilização na área. Ao mesmo tempo que é notado seu crescimento, também aumentam as dúvidas sobre o que se trata tal metodologia. Enquanto isso, devido à sua relativa simplicidade e facilidade de implementação, o tradicional processo em cascata é um dos mais aplicados (KASHIF et al., 2009).

Tratando-se de certificações da área, as regras do *Capability Maturity Model* (CMM)/*Capability Maturity Model Integration* (CMMI) podem ser instanciadas para o desenvolvimento de sistemas embarcados, assim como suas práticas sem grandes custos (JUN; RUI; YI-MIN, 2007). Para isso, é necessário amadurecer bem os passos a ser

seguidos, adequando à situação atual do desenvolvimento. Devido à atual dinâmica de requisitos existentes em grande parte dos projetos (parte por levantamento de requisitos mal realizada, parte por mudança de comportamento de mercado e parte por criação de novas ideias), faz-se mais que necessária a utilização de uma metodologia ágil para poder realizar a entrega em tempo hábil, conforme solicitado. Para promover a agilidade nos projetos, Huang; Darrin e Knuth (2012) recomendam algumas técnicas, entre elas:

- Delegar a algumas pessoas a responsabilidade de se envolver com o solicitante do projeto, envolvendo-o e mantendo-o informado a respeito do projeto;
- Delegar responsabilidades a todo líder de cada sub-projeto (módulo) envolvido, tornando-o responsável por seu andamento e qualidade, além da integração com os outros módulos;
- Reunir todos os dias (ou com grande frequência), por um curto tempo, os responsáveis pelas partes para atualizar o andamento das tarefas, os problemas encontrados e assim verificar possíveis alterações no cronograma;
- Revisar os detalhes do projeto para verificar inconsistências e incluir melhorias;
- Adaptar os processos aos requisitos do projeto, focando apenas no que é essencial para a entrega, sem deixar de lado o gerenciamento;
- Efetuar testes e análises o mais cedo possível para evitar possíveis falhas críticas ao final do projeto, comprometendo a data final de entrega.

Liggesmeyer e Trapp (2009) propõem, em seu trabalho, um processo de desenvolvimento orientado a modelagem (MDD), no qual existem várias iterações - com o cliente final - que verificam mudanças a serem realizadas no modelo entregue, mas que ao final encontram-se estabilizadas por meio dos testes realizados a cada validação. Pode-se notar na figura 4.7 a grande quantidade de iterações com o cliente através dos protótipos e que são refinados a cada rodada de desenvolvimento, tornando o ciclo cada vez menor, o que indica menos tempo gasto entre as entregas.

Enquanto isso Tackett e Doren (1999), através da figura 4.8, indicam o uso de processos curtos, que consistem em iterações rápidas e recorrentes, e mais detalhados em outra representação gráfica, mas com o mesmo objetivo.

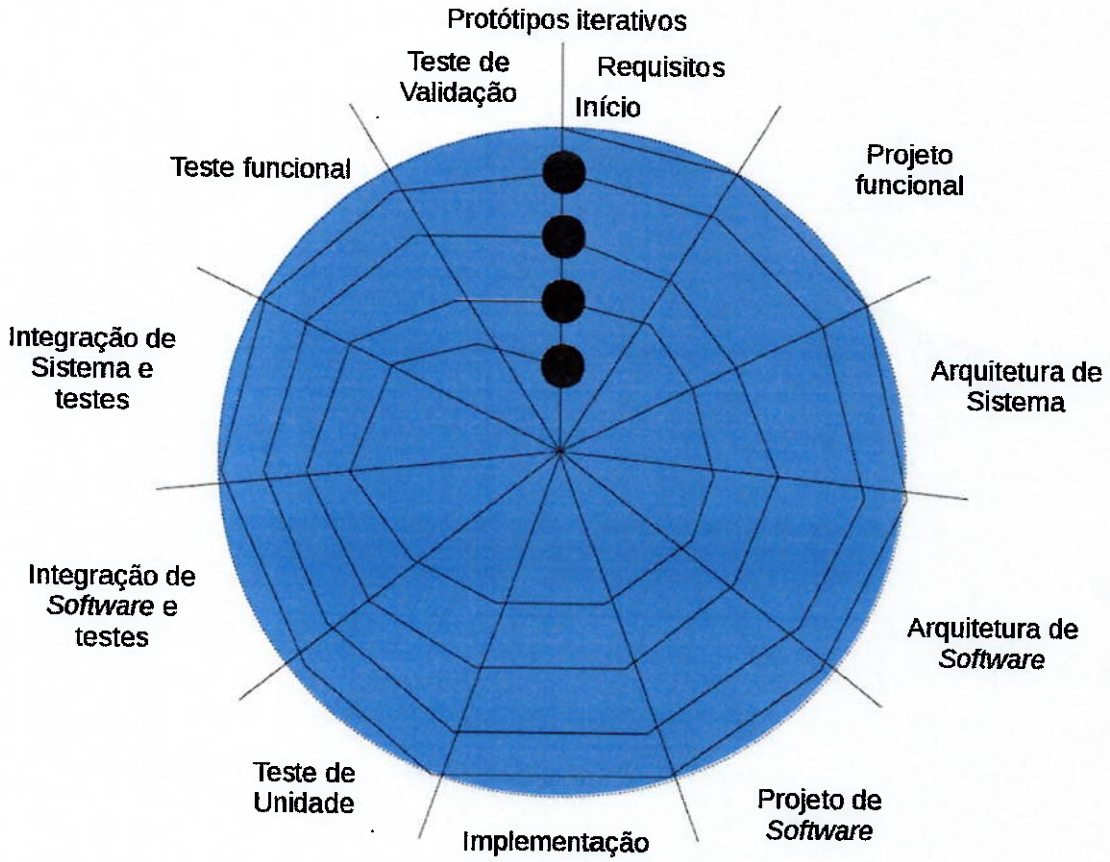

Figura 4.7: Modelo de desenvolvimento de Liggesmeyer e Trapp (2009)

Existem, contudo, recomendações de um processo mais linear com recomendação de monitoramento: na figura 4.9 apresentada por Garcia e Andrea (2010), o desenvolvimento segue o estilo cascata, sendo uma tarefa realizada por vez, mas com possibilidade de monitoramento.

Apesar de estar crescendo em aplicações e desenvolvimento de produtos, o Desenvolvimento Orientado a Testes (*Test Driven Development* (TDD)) não tem sido aplicado tão amplamente em sistemas embarcados (GRENNING, 2007). Um dos primeiros desafios para tal é a adaptação do padrão para o mundo dos processadores embarcados. Ciclo do desenvolvimento TDD, que envolve:

- Criar um teste;
- Gerar o produto ou unidade a ser testada, realizar todos os testes e verificar se o novo teste detecta falhas;
- Escrever o código de modo a corrigir eventuais falhas;
- Gerar novamente o produto ou unidade a ser testada;

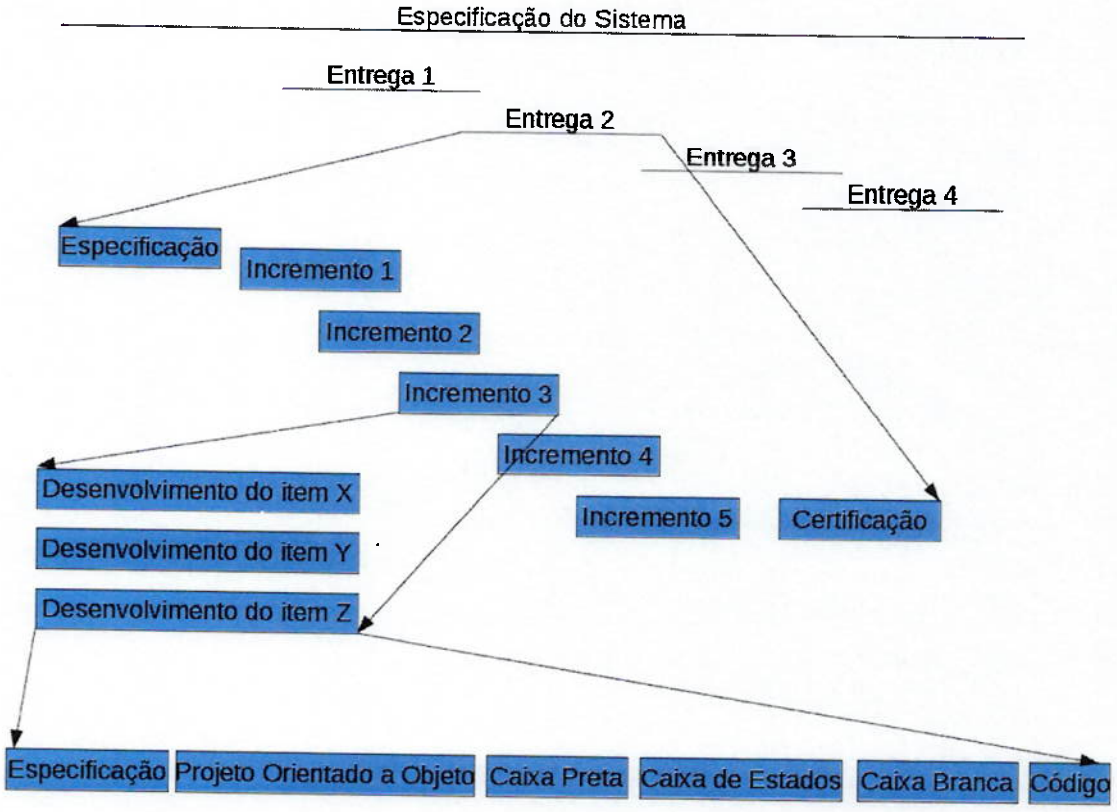

Figura 4.8: Desenvolvimento evolutivo e incremental de Tackett e Doren (1999)

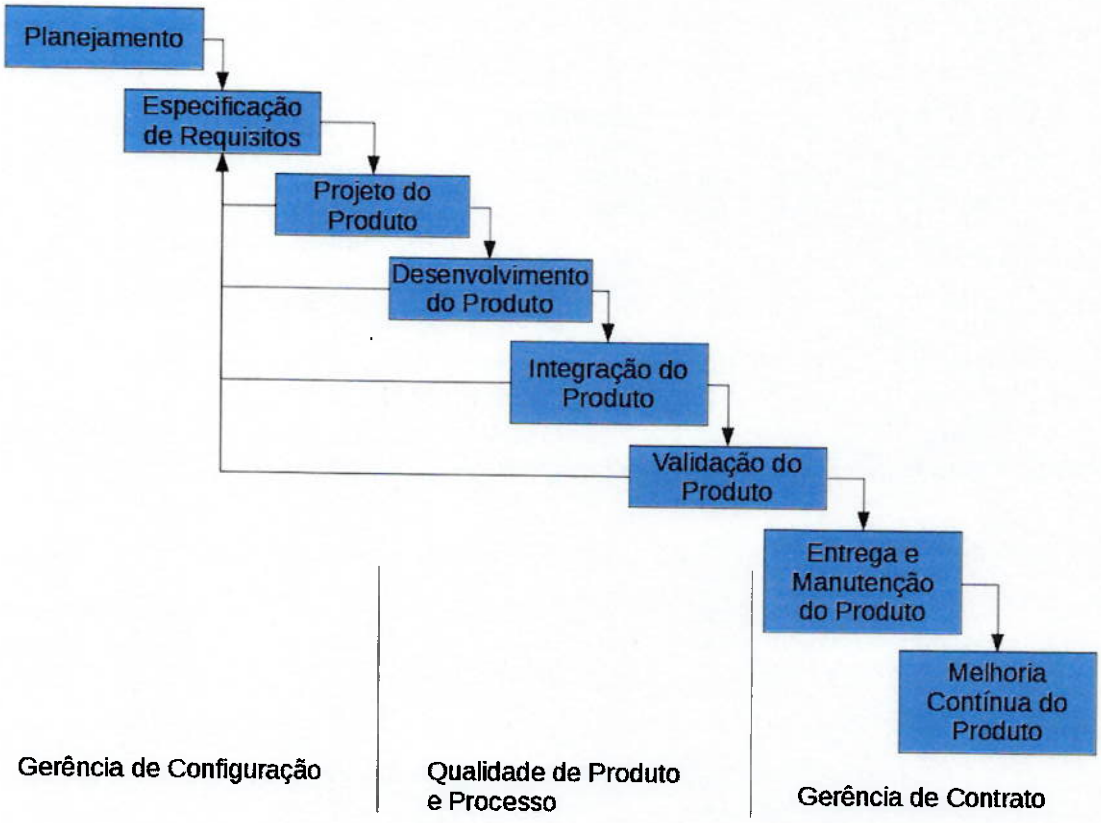

Figura 4.9: Método SPIES sugerido por Garcia e Andrea (2010)

- Testar e verificar se o novo resultado é o sucesso;
- Refazer para remover duplicações;
- Repetir os passos anteriores.

Ao mesmo tempo, as seguintes atividades devem ser realizadas para seu desenvolvimento:

- Identificar os colaboradores;
- Criar um conjunto de testes a que o módulo em questão deve ser submetido;
- Escolher uma parte dos testes para poder explorar as interfaces do módulo;
- Aplicar o ciclo do TDD a fim de definir a interface e como utilizá-la. Aumentar a lista de interfaces de acordo com o necessário;
- Ordenar os testes a serem feitos de acordo com o nível de complexidade, sendo que os mais difíceis utilizam o sucesso dos testes que já obtiveram êxito;
- Aplicar o ciclo do TDD a estes testes, acrescentando novos testes, se necessário;
- Conferir se todas as possibilidades foram cobertas.

Além disso, o TDD provê benefícios à equipe tais como: melhora de previsões de testes (menores e melhor mensuráveis), repetibilidade dos testes (várias iterações) e tempo de depuração menor (testes menores). Por outro lado, alguns desafios do TDD aplicado em sistemas embarcados envolvem:

- Compatibilidade entre o sistema utilizado para desenvolvimento e compilador;
- Facilidade para testar, inclusive o *hardware* e ambientes com pouca memória;
- Tempo para teste e ferramentas para desenvolvimento.

Kashif et al. (2009), de outro modo, propõem a criação de dois modelos, sendo o primeiro para a geração de módulos e o segundo para a realização dos testes dos mesmos, ao invés do tradicional com um fluxo único destas duas tarefas. Nos modelos propostos, os requisitos são interpretados pelo desenvolvedor (para modelagem da aplicação) e também

pelo testador (para criar os casos de teste). Por serem tarefas separadas, suas interpretações podem ser diferentes, ajudando na descoberta de erros na especificação e no entendimento por parte dos envolvidos.

Com todas as ideias anteriormente citadas, tendo em mente a grande concorrência do mercado e a necessidade de apresentar a evolução dos projetos aos clientes e gestores, são aplicadas a metodologia ágil de modo iterativo e incremental com características de TDD. A figura 4.10 representa então as várias iterações e tarefas em ciclos de teste que devem ser executados em curto espaço de tempo, tentando sempre reduzi-lo a cada evolução do processo.

No diagrama foram acrescentados também o documento com os requisitos do cliente (muitas vezes representado pelo analista de negócio) para poder avaliar os riscos e tempo do projeto, assim como uma base de dados com os problemas levantados em testes com o cliente para análise em casos futuros.

4.4 TERCEIRA EVOLUÇÃO: LEVANTAMENTO DE REQUISITOS

Por serem necessários para desenvolvimento, implementação, teste e validação do sistemas, a função de levantamento de requisitos é crítica no processo (KRAMMER et al., 2010). Uma ideia proposta para os processos de especificação, que ajudam a eliminar imprecisões e ambiguidades é bem aceita pela indústria, uma vez que o desenvolvimento avança ao mesmo tempo que novos requisitos, arquiteturas e plataformas aparecem (KASHIF et al., 2009).

Levantar requisitos é uma das mais difíceis tarefas na criação de sistemas (TACKETT; DOREN, 1999), tanto por descobrir se é um verdadeiro requisito, quanto traduzi-lo em comportamento computacional. Requisitos ambíguos ou mal levantados possuem consequências graves no andamento de um projeto, podendo levar à necessidade de mudanças de paradigma, arquitetura e implementação custosas em termos de tempo de entrega, trabalho desnecessário e falta de confiabilidade por parte do cliente.

Segal e Morris (2008) citam como exemplo funcionários de recursos humanos e contadores, cuja grande maioria sabe o que é desejado de um *software* ou sistema devido

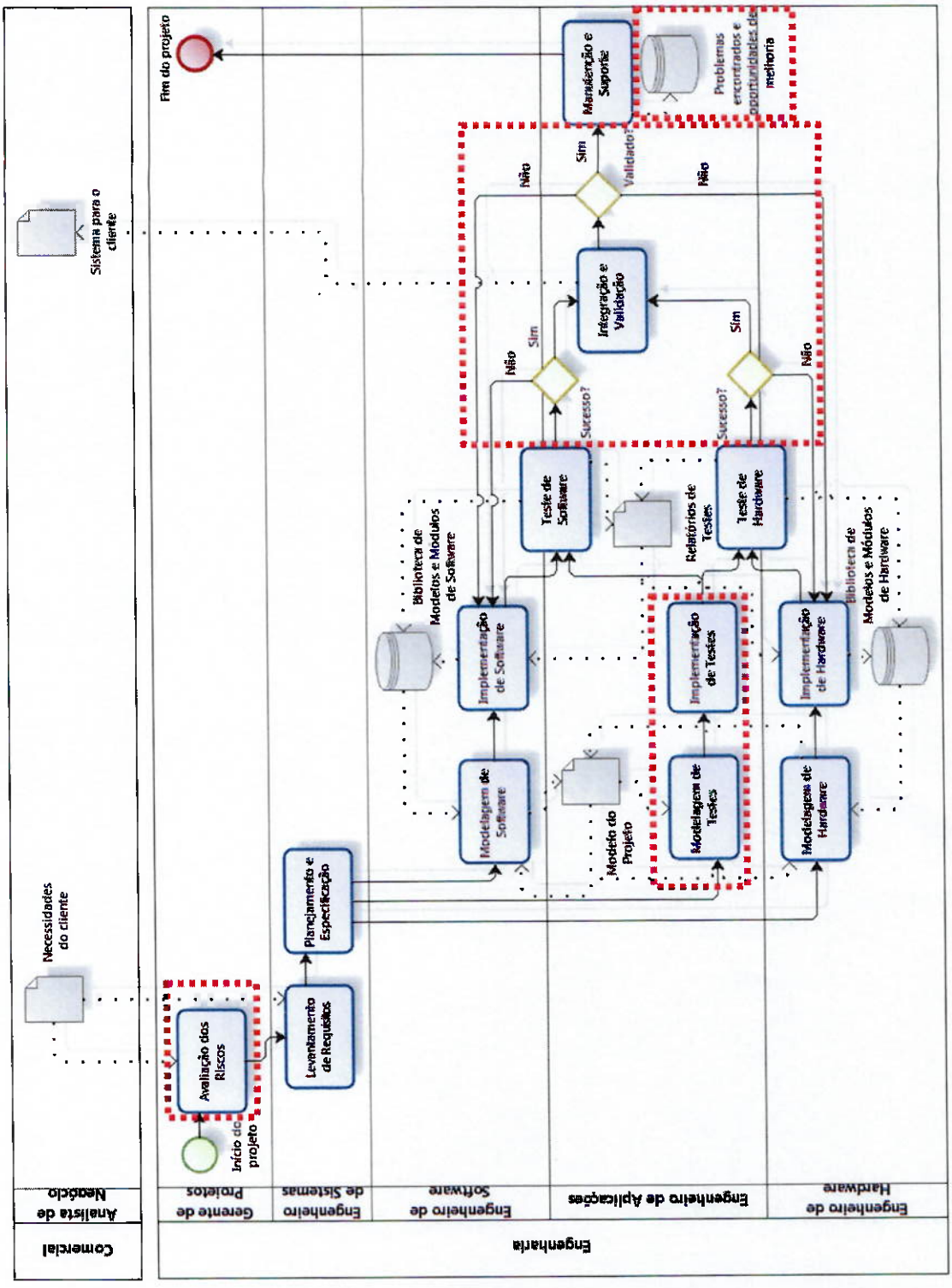

Figura 4.10: Proposta com metodologia ágil

ao entendimento do domínio no qual estão trabalhando. Parte de seus requisitos podem ser alterados, mas não o conhecimento do domínio. Já no caso de cientistas a situação é diferente, porque o objetivo das aplicações por eles utilizadas é justamente passar a conhecer e entender, ou ao menos ampliar o que já se conhece sobre, o domínio objeto de estudo. Parte dos procedimentos realizados são simulações e a partir dessas que surgem

novos requisitos ou mudanças de outros prévios, uma vez que os resultados alteram a percepção do evento sob observação. Outro problema neste caso também é a presença de dados reais de resultados para poder efetuar comparações e testes de validação. A figura 4.11 ilustra uma situação conflitante na qual os engenheiros de *software* esperam que os requisitos sejam comunicados, mas o cientistas não os possuem e precisam do sistema para que os primeiros requisitos possam aparecer e ser refinados.

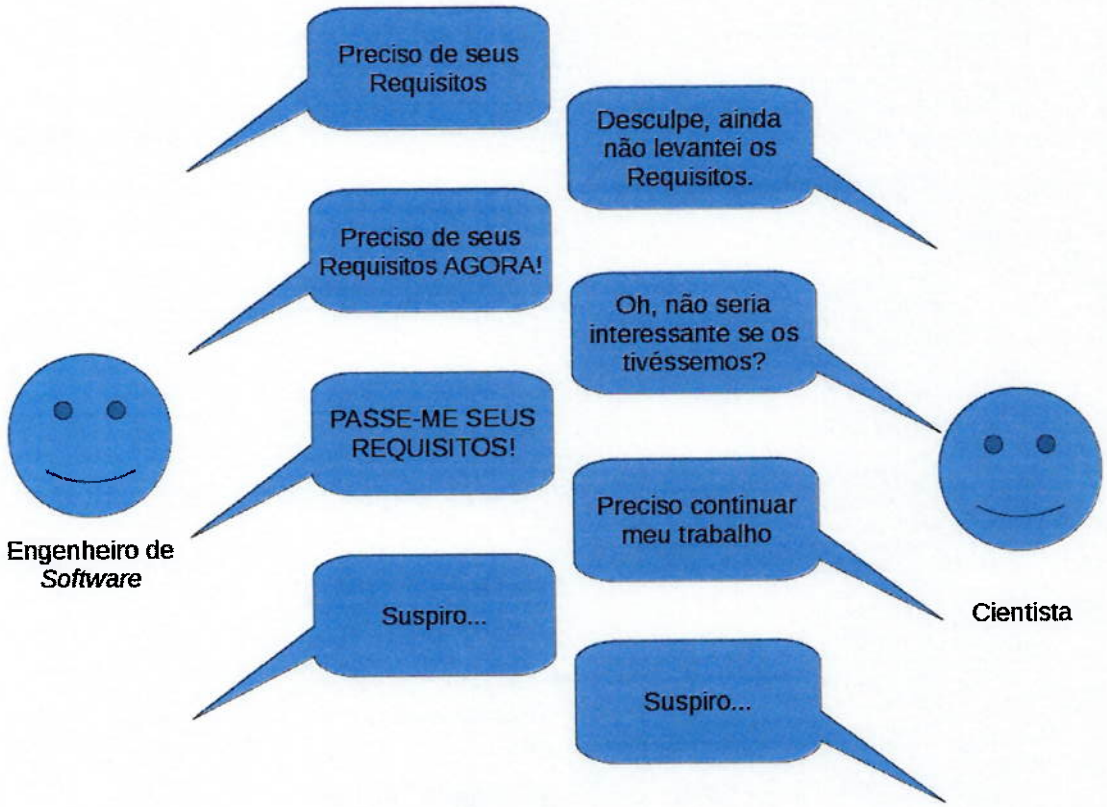

Figura 4.11: Exemplo de levantamento de requisitos sem sucesso por Segal e Morris (2008)

Para evitar erros no levantamento de requisitos (que tornam-se prejuízos ao decorrer das etapas), é primordial que os documentos sejam precisamente escritos e sem ambiguidades (KRAMMER et al., 2010). Devido ao grande número de envolvidos nos projetos, cada qual com diferente conhecimento técnico em diferentes níveis e áreas além dos fatores culturais, criar tal documento é uma tarefa difícil. Para formalizar requisitos levantados sem erros, é necessário gastar considerável tempo, possuir prévio conhecimento técnico na área (domínio) e em ferramentas computacionais. Na prática, é utilizada a linguagem natural, de mais fácil entendimento para todos. Por meio da figura 4.12 o autor acrescenta e detalha o levantamento de requisitos junto ao cliente, o qual não deve ser negligenciado por seus justificados impactos negativos no projeto em caso de erro.

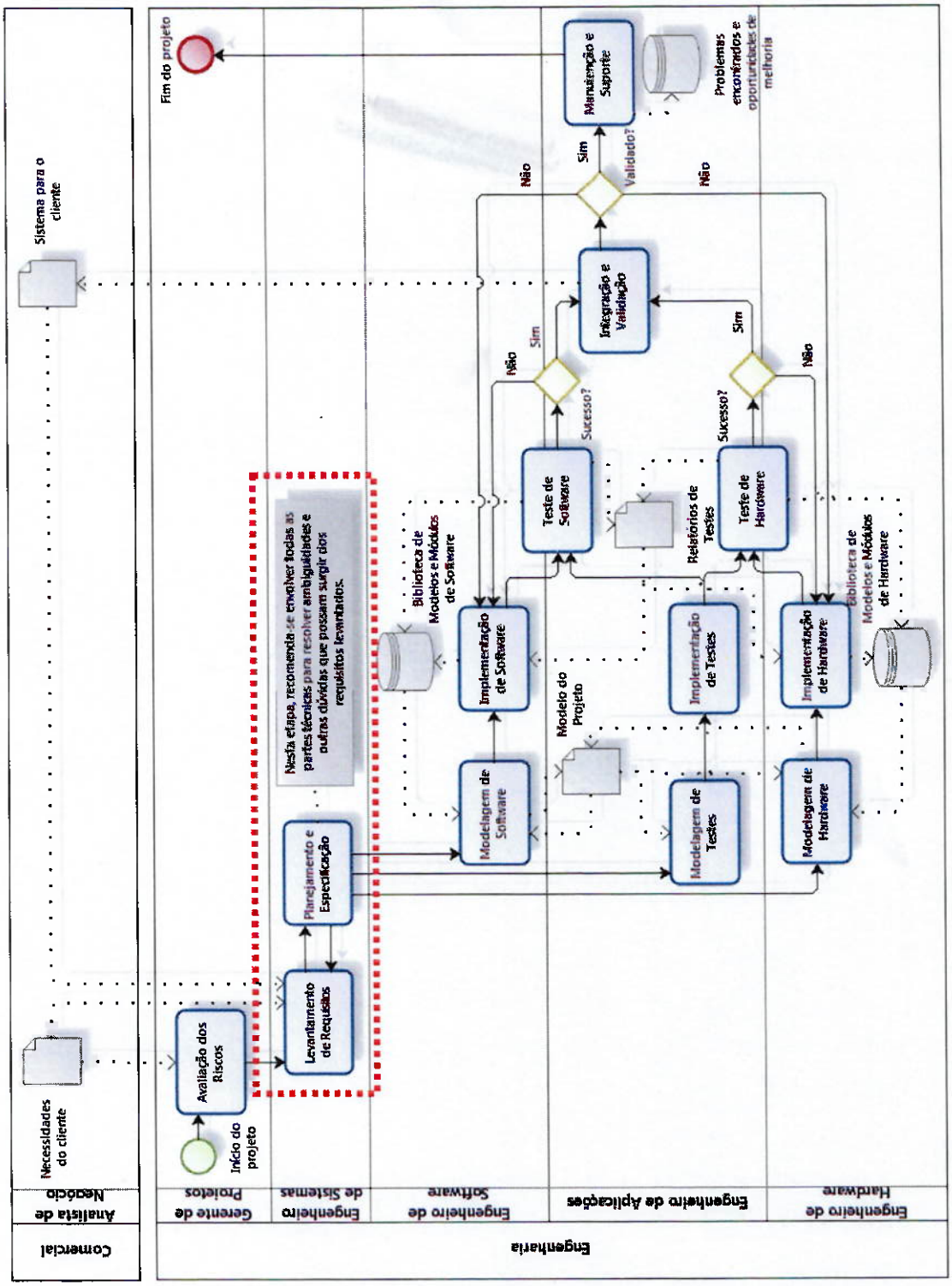

Figura 4.12: Proposta com levantamento de requisitos

Para o levantamento dos requisitos deve-se levar em conta, além dos requisitos originais, os resultados das avaliações intermediárias do sistema em desenvolvimento, podendo esclarecer alguma característica mal levantada. Ao mesmo tempo, criar e atualizar o planejamento em conjunto com os engenheiros competentes reduz o nível de ruídos na comunicação e qualquer outro mal-entendido que possa ocorrer durante o processo.

4.5 PRIMEIRO REFINAMENTO: GERENCIAMENTO E MONITORAMENTO DO PROCESSO E SEUS RISCOS

Não somente o levantamento de requisitos é uma etapa que merece grande atenção dos gestores, mas apenas uma das diversas competências que merecem foco. Segundo o *Project Management Body of Knowledge* (PMBOK) (PROJECT MANAGEMENT INSTITUTE, 2012), o gerenciamento de projetos deve monitorar as seguintes áreas do conhecimento:

Integração do Projeto

Execução de projeto e suas alterações.

Tempo do Projeto

Cronograma e suas atividades.

Escopo do Projeto

Levantamento de requisitos e verificação do escopo.

Custo do Projeto

Estimativas e controle de custos e orçamentos.

Recursos Humanos do Projeto

Equipe (pessoas).

Qualidade do Projeto

Qualidade do projeto.

Riscos do Projeto

Análise de riscos e respostas em caso de problemas.

Comunicações do Projeto

Comunicação entre os envolvidos.

Aquisição do Projeto

Aquisição de elementos vindos de terceiros.

Os projetos de desenvolvimento de *software* por vezes ignoram a avaliação de grandes riscos ao seu sucesso durante todo o processo. Para evitar tal problema (projetos

atrasados e de má qualidade, melhorar produtividade, processos e mitigação de riscos), foi criado o Processo de Desenvolvimento Baseado em Estados (do inglês *State-Based Development Process*) por Tackett e Doren (1999), uma sistemática de projeto baseada em revisões de parceiros, cuja aplicação, comprovadamente, concebeu um sistema crítico em curto espaço de tempo e com bastante qualidade. A sua aplicação não consegue ser indicada em um diagrama de processos, mas é válido para sua eficácia. O sistema utilizado na figura 4.13, indica uma segmentação de etapas para a criação dos SE em partes gerenciáveis, sendo que cada transição de estado deve ser notificada (seja por meio de relatório, ferramenta computacional de acompanhamento de projeto ou por outro meio).

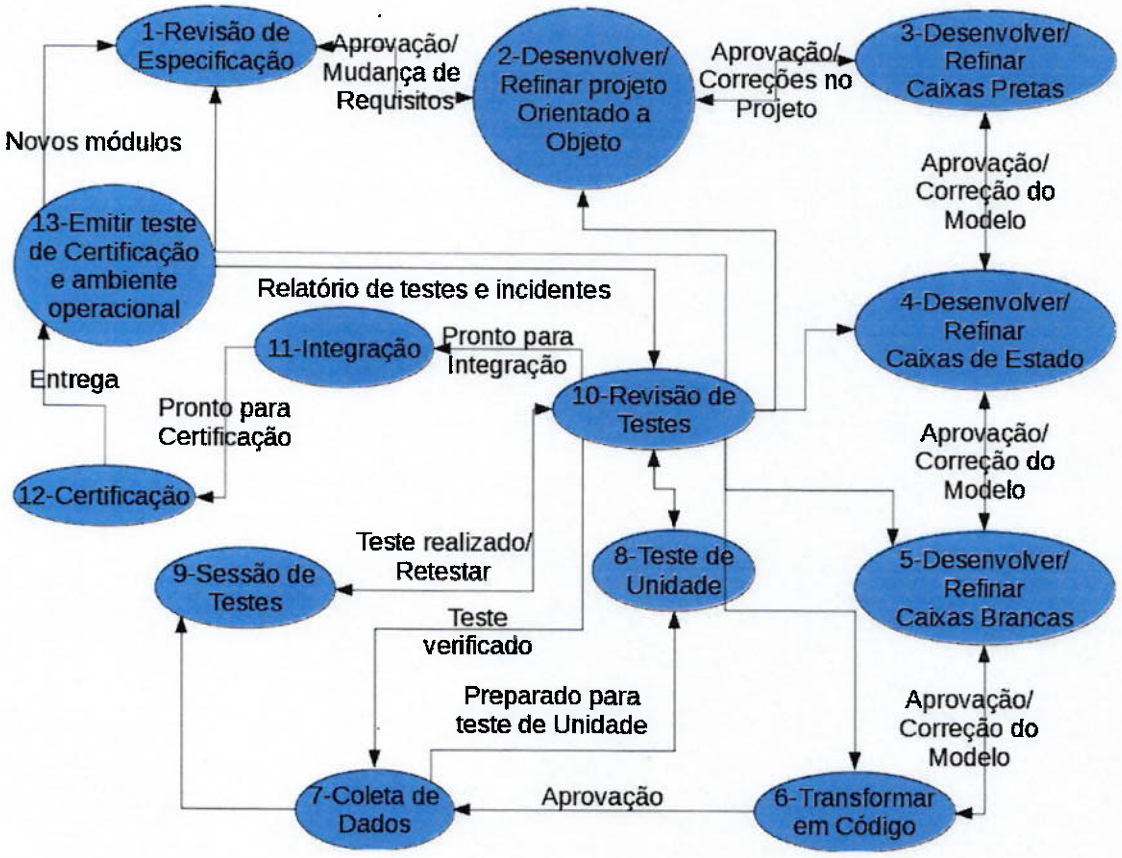

Figura 4.13: Processo baseado em estados de Tackett e Doren (1999)

Axelsson (2011) declara que encontros semanais frequentemente utilizados para coordenar atividades paralelas de diferentes arquitetos é uma prática considerada efetiva, contudo pode ser melhorada através do uso de ferramentas computacionais para poder detectar casos em que duas ou mais pessoas estão executando as mesmas tarefas do contexto. Para um acompanhamento melhor do andamento do projeto, são então adicionados os meios para controle como reuniões periódicas e documentos adicionados como saída de

tarefas da figura 4.14 que são os utilizados para alimentar as estatísticas de gerenciamento e facilitadores do monitoramento dos mesmos.

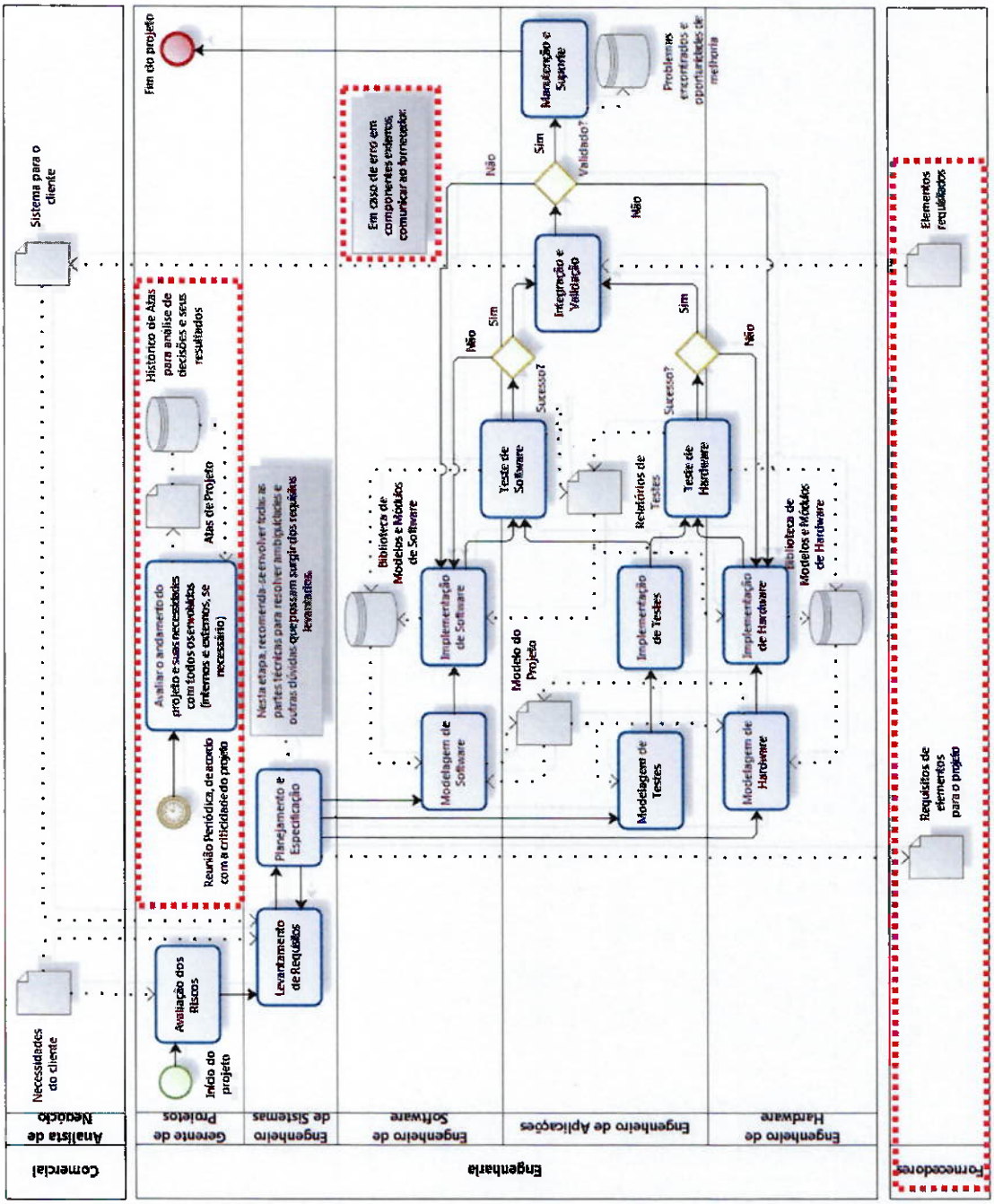

Figura 4.14: Proposta com monitoramento do processo

No que tange a rastreamento de informações, o autor considera essencial acrescentar, além de reuniões periódicas com suas devidas atas, o monitoramento da comunicação com fornecedores cujos prazos podem atrasar ou estar incompletos. Por se tratar de um sistema encadeado, todas as partes podem sofrer com prazos contratuais e forma como é vista no mercado.

4.6 SEGUNDO REFINAMENTO: REFATORAÇÃO

A refatoração é um processo que envolve melhoria de HW e/ou SW através de uma mudança na sua arquitetura ou em componentes-chave que melhore significativamente seu desempenho, diminua seu custo de produção e proporcione maior flexibilidade e facilidade de manutenção. Por vezes, tal procedimento visa atualizar um projeto antigo para uma linguagem e/ou paradigma mais atual.

Outro benefício é não depender da memória de desenvolvedores que não documentaram o projeto ou que já não estão presentes na empresa, pois também não raros são os casos em que mudanças nas últimas etapas de um projeto foram necessárias. Além de acontecer em situações em que há pressão pela entrega, a falta de tempo para implementação adequada das etapas de um ciclo normal é comum. Neste caso, evoluções/refatorações futuras do mesmo projeto podem ser de difícil execução, sem a devida documentação original, com todas a possibilidades e características da plataforma em desenvolvimento, perdendo informações que podem se tornar valiosas para outros projetos.

Em muitas empresas há tentativas de se identificar oportunidades de melhoria para que se possa refazer o projeto com um pretexto, mas isto não é uma norma instituída. Das companhias pesquisadas por Axelsson (2011), não houve uma sequer que tornasse refatoração uma prática constante.

A sugestão de uma refatoração periódica (podendo ser realizada após meses ou anos, por exemplo) da figura 4.15 tem como objetivo tornar instituída sua prática, possibilitando usufruir dos benefícios mencionados anteriormente.

A possibilidade de uma melhoria pode ser percebida por indicação de pessoal externo ou interno ao setor e à própria empresa, sendo sua ideia avaliada pelos gestores por sua viabilidade ou não. Adicionalmente a isto, é recomendado pelo autor armazenar, de modo simples e prático, as sugestões e problemas encontrados em projetos de versões anteriores para não haver os mesmos erros nas próximas versões.

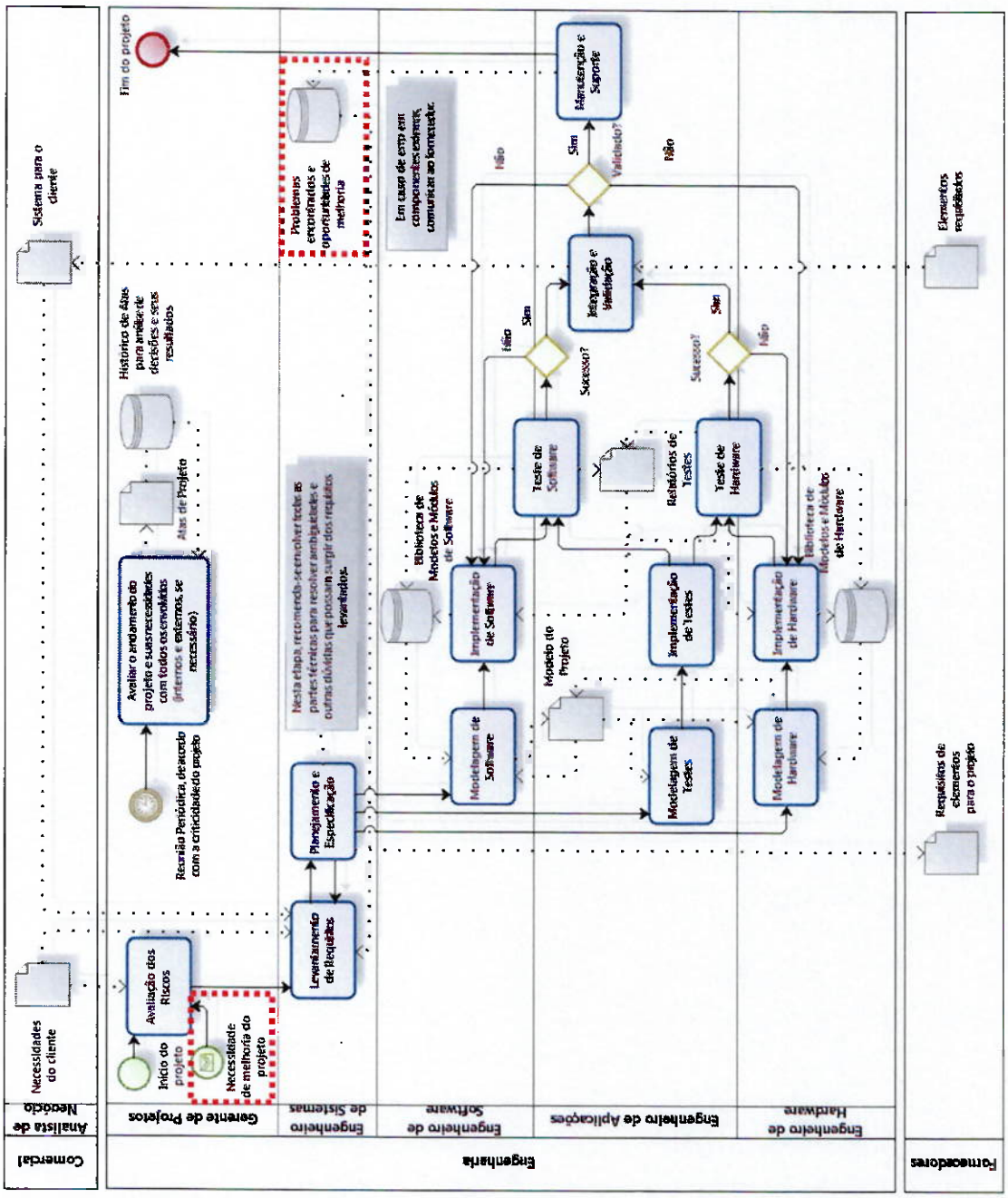

Figura 4.15: Proposta adicionada de refatorações periódicas

4.7 TERCEIRO REFINAMENTO: DOCUMENTAÇÃO

Devido à pratica do reuso de diversos componentes e módulos (especialmente de *software*), a documentação dos mesmos deve ser atualizada constantemente, possibilitando sincronismo entre o que o usuário do módulo pode esperar deste e o que realmente está implementado (MURANKO; DRECHSLER, 2006). Apesar da vital importância da prática, por muitas vezes esta tarefa é negligenciada, inclusive pelo, algumas vezes, inexistente processo interno das empresas.

Infelizmente, a documentação atual de muitos dos aparelhos e módulos não está em sintonia com o produto final, causando complicações de entendimento, inclusive por ser complexa, incompleta e extensa. Por várias vezes, a documentação incompleta mostrou-se fonte de erros cometidos na etapa de planejamento e análise de projetos maiores. Em seu artigo Muranko e Drechsler (2006) sugerem, de forma abreviada, as seguintes etapas para criar documentação para o projeto:

- Análise de Requisitos;
- Planejamento;
- Análise do Projeto;
- Manutenção.

Nessas etapas as saídas devem ser divididas de acordo com seu tipo:

Documentação técnica

Relacionada às informações necessária para o produto e seu uso. Será utilizada durante a vida útil do produto.

Documentação interna

Instruções e informações exclusivas das equipes internas ao desenvolvimento, como especificações de requisitos, construção, produção, testes e qualidade.

Documentação externa

Relativos informações entregues ao clientes finais, como instruções de uso e manuais, por exemplo.

E suas áreas:

Documentação de projeto

Envolve a parte gerencial do projeto, sendo considerada não-técnica.

Documentação de desenvolvimento

Descrição de problemas, soluções, contornos e quaisquer ferramentas utilizadas nos procedimentos a estes relacionados.

Documentação do produto

Visão geral do produto, assim como maiores detalhes de seu funcionamento em forma de fluxogramas, por exemplo.

Documentação de usuário

Refere-se à forma de operação do produto pelo usuário, como manuais, instruções de manutenção e documentação da venda.

Tais elementos documentais da figura 4.16 podem e devem ser atribuídos ao refinamento proposto, novamente ressaltando que sua adoção em forma completa ou simples varia da cultura da fábrica de *software* e dos tempos disponíveis para tal atividade.

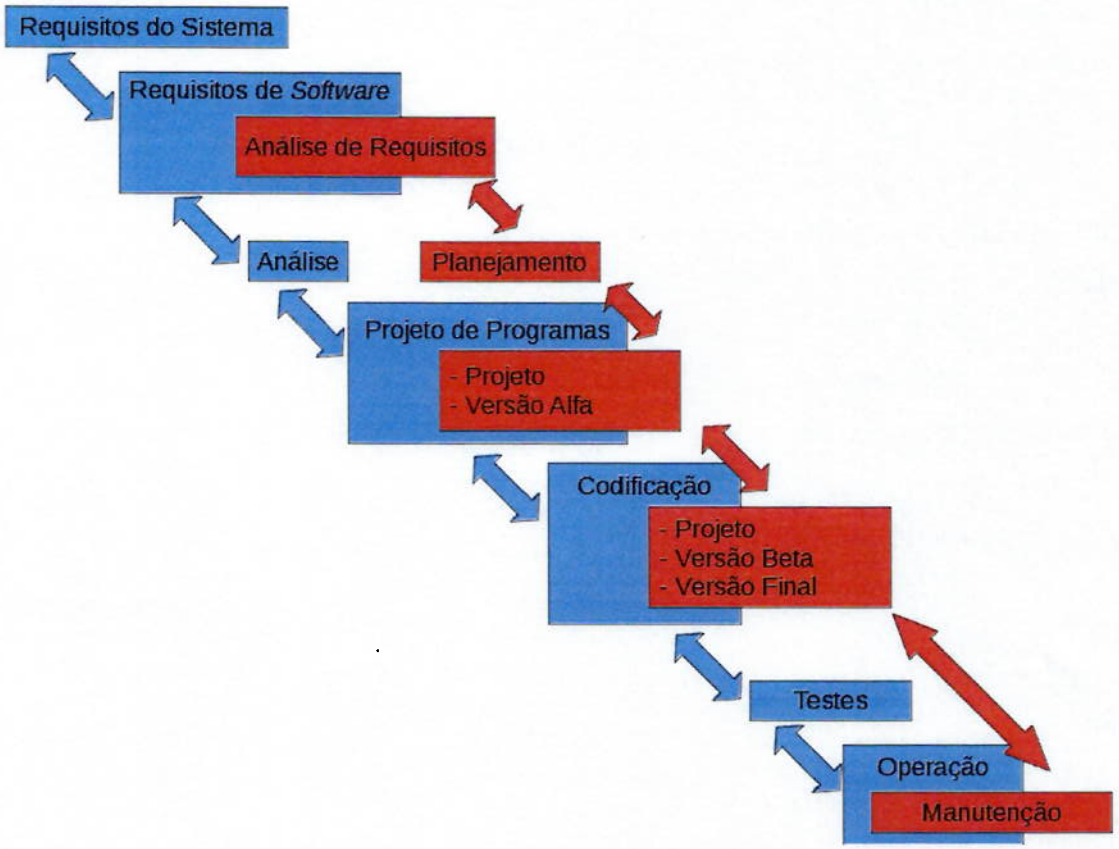

Figura 4.16: Proposta de documentação para sistemas embarcados de Muranko e Drechsler (2006)

Acrescentando ao modelo em especificação, tem-se a proposta de documentação da figura 4.17, visando orientar as equipes a produzir saídas complementares aos subprodutos criados nas tarefas, podendo ser utilizados em consultas futuras.

Há o acréscimo por parte do autor de um repositório de manuais e instruções para o cliente final, assim como características técnicas, um banco de dados com Requi-

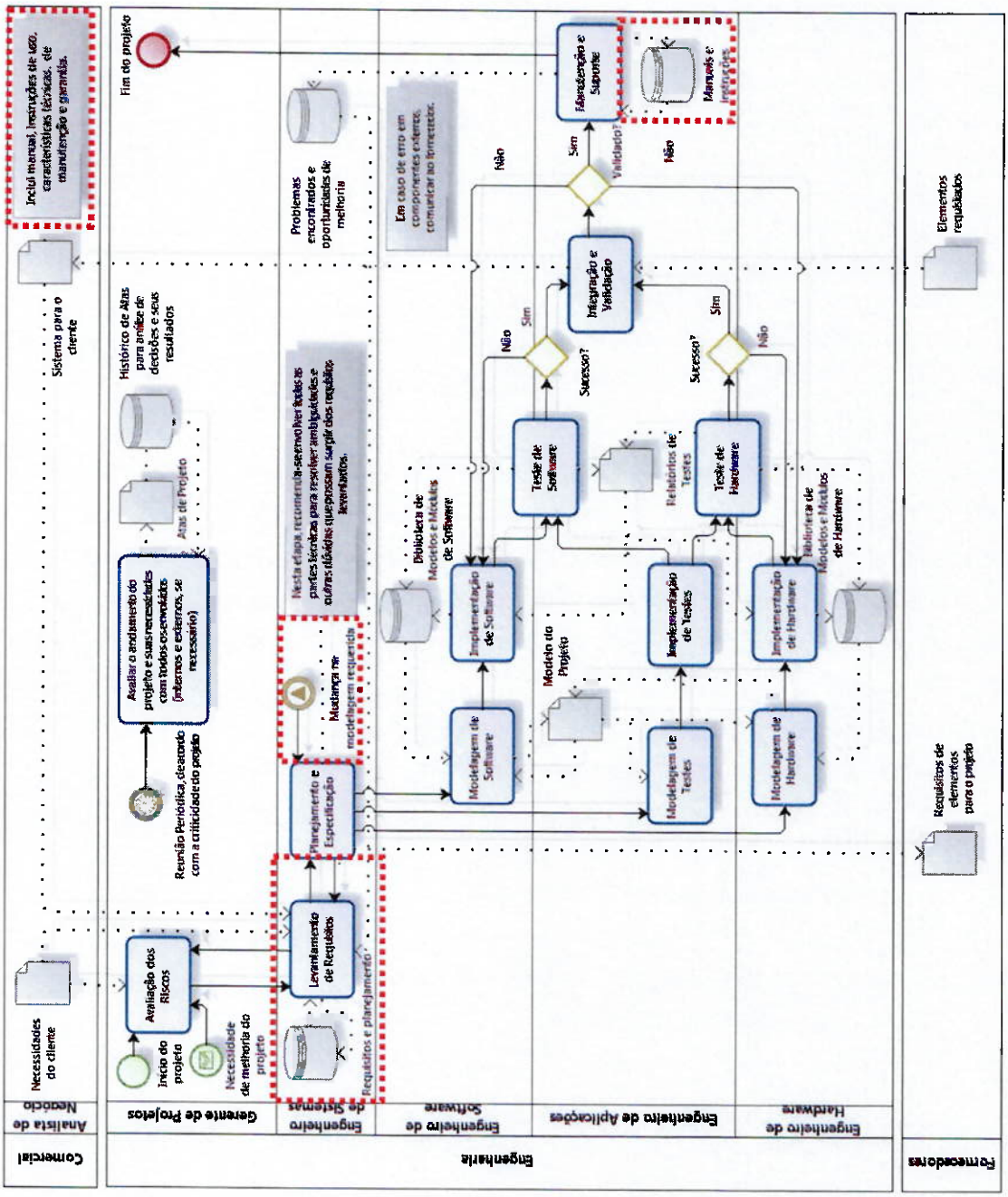

Figura 4.17: Proposta com criação de documentações

sitos e Planejamentos já efetuados para consulta futura e finalmente um sinalizador para a necessidade de mudança na modelagem caso informações não estejam coesas ou seja necessária sua refatoração.

4.8 PROPOSTA DE MODELO FINAL

Justificados a necessidade da criação de um modelo em notação BPMN, com seus benefícios e variantes de utilização possíveis, assim como as melhorias implementadas para

a área de sistemas embarcados durante sua concepção, o modelo final é apresentado em seu formato final pela figura 4.18, que não é um modelo rígido para ser seguido nos mínimos detalhes. Este modelo é apenas uma proposta de passos a seguir durante a criação de um sistema com SE, podendo ser adequado às necessidades atuais da empresa que a utilizar.

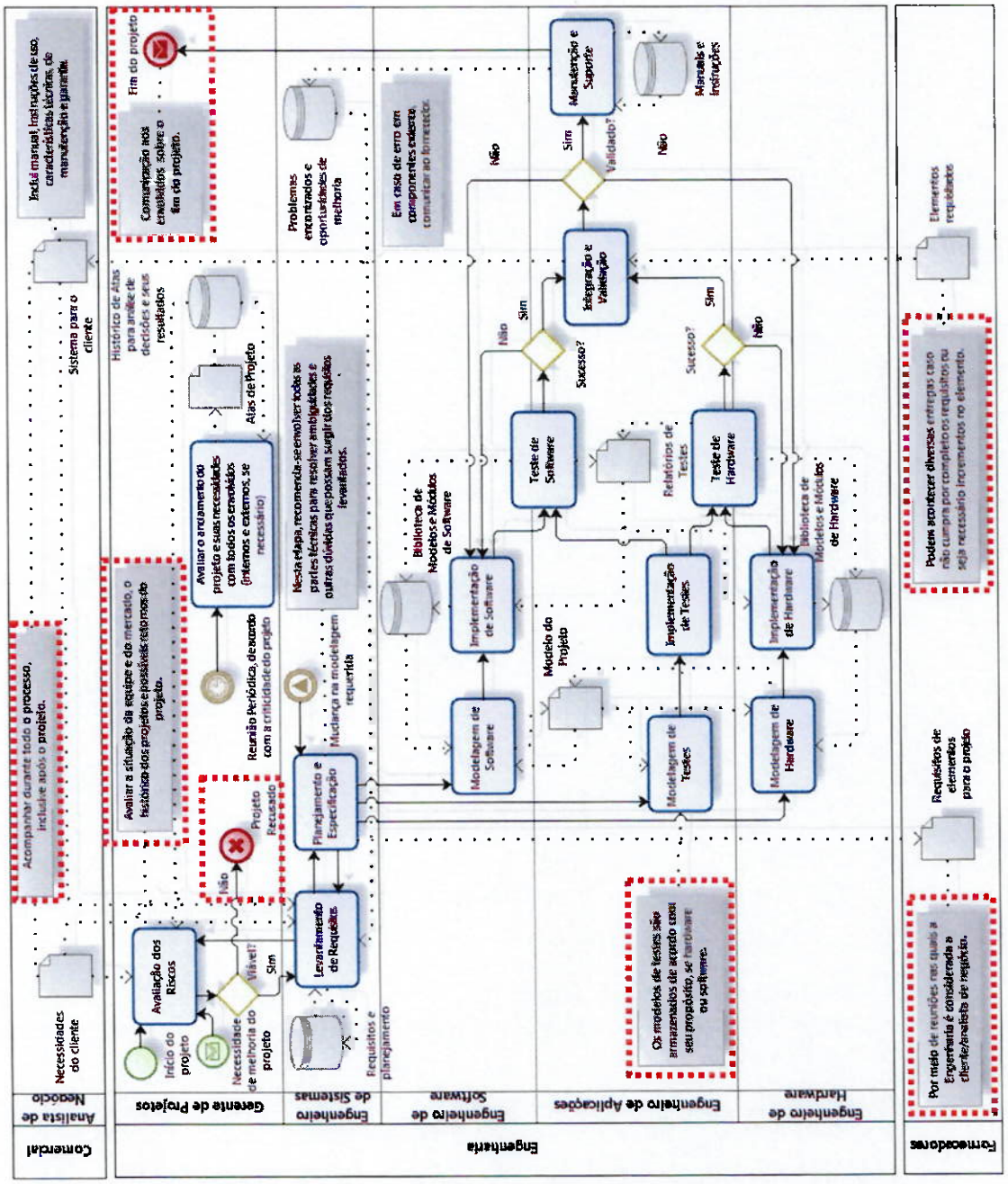

Figura 4.18: Proposta final de um processo em BPMN para *software* embarcado

Para melhor acompanhamento, o Analista de Negócio deve verificar periodicamente o andamento do processo (reuniões periódicas ou por mensagens), inclusive com acompanhamento junto ao cliente após a conclusão do projeto. Cabe também entre as atribuições do gerente comunicar aos envolvidos o final de um projeto, assim como avaliar

sua recusa. Ao final, fica também explícito por meio de comentários o acompanhamento junto a terceiros para evitar problemas críticos durante as entregas.

Pela experiência do autor do trabalho, não basta apenas criar o produto e validar com o cliente. Se faz também necessário poder produzi-lo em escala e com baixo índice de falhas. Sistemas eletrônicos tendem a apresentar menos falhas de verificação que seres humanos, portanto todo este processo de desenvolvimento de SE é refeito adaptando seu modelo para um cliente diferente: a fábrica juntamente com a qualidade de produto, tendo que recolher seus requisitos, suas limitações e passar pelos mesmos processos como se fosse um desenvolvimento padrão.

Por ser um ambiente complexo e multi-facetado, uma empresa possui componentes que não são relacionados a processos, mas sim à sua execução. No capítulo 5, são discutidos elementos que podem ser agregados ao processo tornando-o mais vantajoso em termos de tempo e rastreabilidade.

4.9 CONSIDERAÇÕES FINAIS DO CAPÍTULO

Através da aplicação de diversas características que podem impactar o desenvolvimento de *software* embarcado no modelo, partindo de simples modelos concebidos por diversos autores em notações diversas, foi criada a especificação de uso genérico para desenvolvimento de sistemas embarcados. A aplicação de cada uma das etapas (evoluções e refinamentos) no modelo principal, mantendo a essência da notação proposta necessitou de estudos e algumas tentativas para que sua utilização não comprometesse a uniformidade dos processos, tampouco removesse a possibilidade de tornar o processo genérico.

O acréscimo gráfico das informações no modelo especificado foi realizado através do programa interface utilizado, sendo possível também gerar as figuras presentes no trabalho sem contratempos. O processo proposto visa, apoiado em sua bibliografia, poder ser utilizado e adequado aos ambientes produtivos sem que haja grandes intervenções no modelo original, garantindo a estabilidade e simplicidade da especificação original.

5 CONSIDERAÇÕES ALÉM DO PROCESSO PROPOSTO

Propor um novo processo, ou mesmo qualquer mudança em um ambiente empresarial, está sempre atrelado à resistência de células de trabalho que não se sentem confortáveis com as alterações inseridas no seu dia-a-dia. Para vencer os desafios impostos, é necessário avaliar alguns dos aspectos de diversas áreas. Sua análise não deve ser limitada a apenas aos elementos abordados neste trabalho, dada a variedade de contextos encontrada no mercado e as especificidades de cada local de trabalho. Alguns dos elementos a se levar em conta - que não foram inseridos no diagrama de processos por sua complexidade ou impossibilidade de aplicação de modo gráfico - são:

1. PESSOAL E AMBIENTE
2. FÁBRICA DE *SOFTWARE*
3. DEFINIÇÃO DE LINGUAGENS E ARQUITETURA
4. FERRAMENTAS COMPUTACIONAIS
5. PROJETO MECÂNICO
6. CLIENTES E FORNECEDORES

Esses elementos são considerados pelo autor como críticos por sua relavância no resultado final dos projetos de sistemas embarcados.

5.1 PESSOAL E AMBIENTE

Axelsson (2011) declara em seu artigo que pessoas para arquitetar e executar o trabalho são indispensáveis. Por muitas vezes as experiências da carreira dos especialistas são essenciais na prevenção de erros e planejamento dos projetos a iniciar ou em execução. Para suprir a ausência de um desenvolvedor, arquiteto ou gerente é necessário que o

substituto tenha conhecimento mínimo do assunto tratado para poder dar continuidade ao processo. Ter apenas pessoas jovens e inexperientes leva a um considerável tempo gasto na curva de aprendizado, por vezes sacrificando projetos lucrativos e de boa visibilidade.

Na figura 5.1 de Mitsui; Kambe e Koizumi (2009), são relacionadas algumas das competências que devem ser desenvolvidas pelos profissionais envolvidos com *software* embarcado. Assim como os engenheiros de HW e SW devem conhecer partes específicas de suas próprias áreas, tanto eles quanto o arquiteto/engenheiro de sistemas precisam conhecer fundamentos da integração entre as partes do sistema e modos eficientes de teste e manutenção da qualidade.

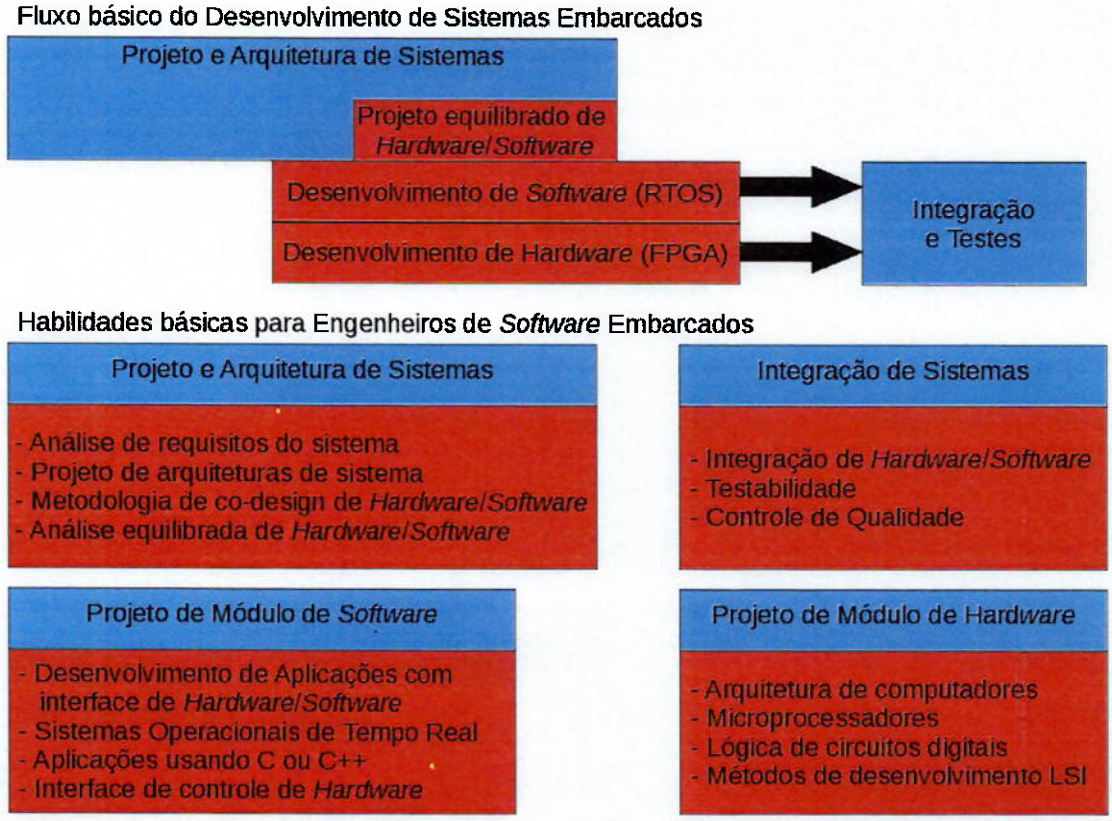

Figura 5.1: Habilidades essenciais de um engenheiro de *software* segundo Mitsui; Kambe e Koizumi (2009)

Além dos recursos humanos, a melhora dos processos de *software* depende também dos outros recursos disponibilizados pela empresa (como materiais, ferramentas, investimentos na especialização de pessoal...) e seus objetivos (JUN; RUI; YI-MIN, 2007). Por experiência própria, sem o apoio do corpo gestor torna-se insustentável o desenvolvimento dentro dos padrões que qualidade e tempo desejados, pois todos os elementos humanos participantes dos processos estão sobrecarregados pelo excesso de tarefas e de

falta de ferramentas para sua automatização, além de insatisfação se não for mostrado comprometimento em suprir as demandas sem ao menos avaliar os benefícios que podem ser trazidos.

Gustavsson e Axelsson (2011) mencionam uma pesquisa sobre o atual estado das práticas de arquitetura de sistemas embarcados em diversas indústrias, que se provou bem focado na área de *software*. Pode-se determinar que a diferença dos métodos utilizados e analisados são reflexo dos diferentes ambientes aos quais os projetos estão submetidos. Aspectos internos e externos são observados como influenciadores dos resultados observados, entre eles a adaptação do modelo CMMI em uma das organizações observadas na pesquisa. Muitas das empresas consultadas identificaram a necessidade de treinamento para o novo modo de trabalho utilizado, sendo de grande valia fornecer informações para os arquitetos e outros envolvidos abraçarem o processo instituído. Tal prática poderia reduzir as consequências de descrições de processos mal feitas e melhorar a aceitação dos componentes humanos através de um melhor entendimento da arquitetura proposta (AXELSSON, 2011).

Como forma de avaliar os projetos foram observadas diferentes métricas em todas as empresas, sendo difícil avaliar todas sob a mesma perspectiva, pois as informações levantadas estão de acordo com o que é considerado crítico pelos gerentes no processo. As diferentes ópticas são frutos das decisões tomadas de acordo com a política da indústria (GUSTAVSSON; AXELSSON, 2011), que nem sempre é a mesma para todas.

Em suma, o apoio de todos os elementos de uma equipe harmonizada, adicionado de políticas internas de estímulo por meio de salários e ferramentas produtivas leva a considerável melhora no ambiente e seus resultados, sendo o resultado obtido por diversos meios, como indicados nos estudos apresentados.

5.2 FÁBRICA DE *SOFTWARE*

O conceito de fábrica de *software* engloba, entre outros, os recursos humanos, materiais, processos e metodologias agrupados em um processo robusto e conciso que envolve a modelagem, desenvolvimento, testes, validações e manutenções de *software*. Sua análise através de indicadores de produtividade e qualidade em todas as etapas presentes no ciclo de desenvolvimento possibilita, por exemplo, orientar a re-utilização de componentes,

estimar prazos de projetos, alocar recursos conforme a demanda e detectar necessidade de mudanças.

Tackett e Doren (1999) citam sobre um projeto acompanhado no qual a cada passo dado havia um diálogo sobre os resultados obtidos e opiniões a partir da revisão do código, buscando sempre descobrir algum defeito novo. A partir desta programação em pares, um benefício foi notado: os desenvolvedores menos experientes puderam amadurecer mais rapidamente devido à interação com pares mais experientes, que ajudavam através da revisão do código, dicas e retornos sobre os passos dados. Neste mesmo ambiente, encontros semanais para troca de experiências permitiram discussões sobre a melhora dos processos.

Além da comunicação, o entendimento do projeto por parte de todos os desenvolvedores é importante pois os leva juntos a atingir um mesmo objetivo, cada qual com sua contribuição bem definida. Existem estudos (SCOZZI et al., 2008) que indicam um grande prejuízo devido a lacunas de desenvolvimento nas quais as equipes não tinham as tarefas bem atribuídas, sendo necessário gastar tempo com comunicação para chegar a um concenso durante o andamento do projeto, modificando partes do que já foi feito. A eficácia do desenvolvimento está intimamente atrelada ao quão alinhados estão os modelos mentais dos desenvolvedores acerca do projeto, caso contrário haverá muito tempo gasto para conciliar os modelos. Diagramas podem ajudar a unificar a visão dos membros da equipe, podendo também estimular diferentes opiniões sobre o modelo atual, proporcionando melhora do mesmo.

Grenning (2007) recomenda algumas práticas para uma equipe de desenvolvimento para sistemas embarcados que utilize o desenvolvimento TDD:

- Desenvolver as unidades do sistema e seus testes independentemente do *hardware*;
- Avaliar os mesmos independentemente do *hardware*;
- Criar aplicação para testar a plataforma;
- Criar aplicação final para o produto e testá-lo integralmente.

É comprovado na prática, pelo autor, que uma fábrica de *software* funcional, coerente, com boa comunicação interna, definição adequada de papéis e bem organizada traz grandes benefícios, pois os projetos tendem a ser finalizados com maior qualidade, com

melhor definição em seus modelos, maior facilidade de futuras manutenções e profissionais satisfeitos por notar sua evolução em menores prazos.

5.3 DEFINIÇÃO DE LINGUAGENS E ARQUITETURA

Segundo Muranko e Drechsler (2006), na área de produtos de cliente final o montante de código duplica a cada dois anos. Por muitas vezes este grande montante pode se tornar inutilizável dependendo da técnica de programação utilizada e sua arquitetura. Ao mesmo tempo, Axelsson (2011) declara que várias indústrias baseiam seus complexos produtos com sistemas embarcados em uma boa arquitetura. A partir desta, pode-se criar uma sinergia entre produtos da mesma família, facilitando a evolução de todos os elementos da arquitetura com o passar do tempo. Por mais que seja às vezes necessário refatorar uma família de produtos, por vezes é válido o esforço na tarefa.

Gustavsson e Axelsson (2011) indicam que um dos grandes desafios dos arquitetos é a constante necessidade de decidir entre a evolução do sistema para um novo ou manter sua estabilidade atual, pois ambas as propriedades são contrastantes nos projetos. Com grande frequência, uma linha de produtos mantém a mesma plataforma (representação abstrata de um conjunto de implementações possíveis de uma arquitetura de *hardware* e de serviços prestados pelas aplicações (*software*), segundo Riccobene et al. (2007)) na qual foi criada, mas com modificações adequadas aos seus vários clientes. Por ser tão essencial poder manter uma mesma plataforma (reusabilidade), uma arquitetura bem concebida é primordial para menor duração dos projetos de personalização e facilidade de alteração de vários projetos ao mesmo tempo em caso de erro, afinal o código-fonte possui grande parte unificada. A concepção de uma arquitetura deve ser pensada ainda nos primeiros passos de um projeto para que mudanças durante sua implementação não ocasionem demoradas e custosas alterações de características (AXELSSON, 2011).

O processo de arquitetura é intimamente relacionado às outras atividades do desenvolvimento, podendo por exemplo seguir o modelo em 'V'. Apesar de importante, a esmagadora maioria das empresas não possui um processo de arquitetura documentado. As atividades de arquitetura devem ser relacionadas ao princípio da análise e projeto dos sistemas, etapa na qual requisitos e funções são transformados em sistemas técnicos.

Deve-se também mencionar que os requisitos não-funcionais são necessários para a criação da arquitetura. São incluídos, mas não exclusivos: desempenho, facilidade de modificação, ciclo de vida do produto, futuros desenvolvimentos, produção, serviços e operação. Entradas de características no sistema podem ser realizadas através de um cliente ou de um analista de negócio (por meio de mudança no requisito ou solicitação de nova funcionalidade/melhora de desempenho) ou outros envolvidos com objetivos parecidos para mudanças (AXELSSON, 2011). Como saída do processo tem-se pré-requisitos do sistema a serem entregues aos desenvolvedores. No material entregue podem constar a análise detalhada, definições de interface, restrições de recursos, regras para criação e todo o contexto para o desenvolvimento.

Ainda segundo Axelsson (2011), ao se criar uma nova plataforma, há a oportunidade de se revisar a arquitetura, evoluindo a partir dos erros cometidos nos projetos anteriores. Mudanças típicas desta tarefa são mudanças na forma de comunicação, na estrutura e no núcleo básico da aplicação. Para o sucesso de um projeto, as etapas devem ser:

- Eficientes: consumir o mínimo de recursos possível, tanto em número de arquitetos, quanto de envolvidos.
- Eficazes: entregar resultados com máximo valor agregado aos seus usuários (desenvolvedores e clientes finais por exemplo).
- Entregues em tempo: Ser possível entregar produtos de qualidade em tempo hábil. O levantamento de requisitos deve ser preciso para não haver necessidade de mudanças em cima da hora.
- Equilibrar curto e longo prazos: Entregar o projeto a tempo é necessário por causa do curto tempo para chegar ao mercado, mas é interessante manter a visão focada para poder reutilizar os resultados em projetos futuros. A otimização entre desempenho e custo se faz necessária nesta virtude, ao mesmo tempo que o equilíbrio entre esforços e facilidade de modificação deve ser mantido.
- Ter aceitação: Por ter influência direta no desenvolvimento a longo prazo, decisões arquiteturais de desenvolvimento devem propiciar bons resultados de modo a motivar sua continuação, tanto por gerentes como por desenvolvedores.

Adicionalmente, são sugeridas as seguintes atividades como boas práticas para um processo de desenvolvimento (AXELSSON, 2011):

- Planejamento e priorização de tarefas: o processo de arquitetura evolucionário é ativado por mudanças nos requisitos originais, normalmente uma nova funcionalidade ou solicitação de melhora de desempenho;
- Análise de requisitos: quando um pedido de mudança é identificada pelo arquiteto, o primeiro passo é realizar a análise do mesmo a partir das necessidades dos envolvidos do projeto e transformar em um requisito novo. Em grandes empresas, tais resultados são armazenados em bancos de dados com controle de versão;
- Desenvolver soluções arquiteturais: o próximo passo é desenvolver uma arquitetura atualizada que supra as necessidades da nova mudança, satisfazendo os requisitos antigos;
- Gerar pré-requisitos arquiteturais: quando houver a necessidade de pré-requisitos para os desenvolvedores do sistema ou produtos, arquitetos devem obtê-los a partir dos já existentes na descrição da arquitetura;
- Planejar refatoração estratégica: Criar uma nova arquitetura sem modificar o comportamento externo, normalmente fruto da necessidade de otimização;
- Gerenciamento de Arquitetura: há a necessidade de gerenciar todos esses processos e os arquitetos que os fazem.

Outros modos formais para modelagem de arquiteturas investigados foram (ANTONIO; FERRARI; FABBRI, 2012):

- Diagramas de estado (*Statechart*);
- Máquina de estados finitos *Finite-State Machine* (FSM);
- Redes de Petri;
- Linguagem de análise e projeto de arquitetura (*Architecture Analysis & Design Language* (AADL));
- MetaH.

Uma técnica interessante a se citar para a melhoria dos fluxos operacinais é a rede de Petri, que ajuda na verificação dos aspectos, evita ambiguidades, incertezas (como duplicidade) e contradições (WU; ZHANG, 2009).

Quando a arquitetura é escolhida, ainda que no nível de arquitetura de sistema, sua influência na aplicação que fará parte do projeto ainda é desconhecida, podendo resultar em necessidade de grandes mudanças nos requisitos devido à impossibilidade de atender os desempenho estipulado (SRINIVASAN; DOBRIN; LUNDQVIST, 2009). Em conjunto, a capacidade do *hardware* se faz crítica, pois afeta os esforços de *software* para o funcionamento adequado do produto final.

O sucesso dos sistemas vem da facilidade de modificações e manutenções. Por muitas vezes é mais fácil atualizar uma plataforma de *hardware* do que efetuar sua manutenção. Assim como para o *software* mais funcionalidades são requisitadas com o passar do tempo. Uma vez que muitas mudanças tornem-se necessárias ao mesmo tempo, sua criação poderá tornar-se cada vez mais demorada e onerosa (SRINIVASAN; DOBRIN; LUNDQVIST, 2009).

Riccobene et al. (2007) apresentam um processo de desenvolvimento unificado específico para a plataforma de sistemas embarcados. Seu objetivo visa aumentar o nível de abstração para facilitar o desenvolvimento de sistemas mais complexos através da manipulação de modelos. Uma eficaz implementação pode levar mais tempo para se fazer, porém a personalização e criação de sub-projetos derivados desta são criados em menos tempo e, mesmo que se descubra algum problema na plataforma, sua correção já é aplicada a todos os projetos que se utilizam do ecossistema, pela própria experiência do autor deste trabalho. O modelo de uma plataforma genérica pode ser reusado e ter como objetivo embarcar um sistema específico com relativa facilidade. Sua arquitetura é construída de modo incremental, criando subsistemas de acordo com suas necessidades e partir destas, refiná-los.

Além das arquiteturas citadas e outras mais existentes, a arquitetura *Service-Oriented Architecture* (SOA) tornou-se um paradigma bastante utilizado por empresas atualmente, segundo o artigo de Barisic et al. (2007). A chave de seu sucesso deve-se ao suporte a reuso e escalabilidade, aumentando eficiência no seu desenvolvimento, entrega e tempo de execução. O requisito das aplicações SOA baseia-se na implementação de um serviço ou conjunto de serviços, muitos com documentação normalmente já defi-

nida. Em caso de erros, a própria arquitetura SOA auxilia na descoberta dos problemas rapidamente.

Padronizar a *Application Programming Interface* (API) de equipamentos ou módulos menores para facilitar o reuso é uma prática que deve ser realizada (RICCOBENE et al., 2007), assim como mapear tanto *Hardware* (HW) quanto *Firmware* (FW) (na parte comum entre elas, como interrupções por exemplo) para poder trabalhar cada um individualmente e em seguida tornar sua junção mais efetiva. Além de padronizar as interfaces, é interessante unificar a linguagem utilizada para mais fácil manutenibilidade de código. Algumas das linguagens utilizadas para SE são: C e C++ para a camada mais baixa de gerenciamento e Qt, C# e VB.NET quando se tornam necessárias aplicações gráficas e de rápida criação.

5.4 FERRAMENTAS COMPUTACIONAIS

Devido ao aumento de complexidade na arquitetura, manutenção e tamanho dos programas atuais, torna-se extremamente raro a autoria de um *software* pertencer a apenas uma única pessoa (SCHROEDER; IBÁÑEZ; MARTIN, 2004). Tão necessários quanto a criação do sistema são os testes realizados no mesmo, além de uma documentação concisa para seu devido rastreamento. Mesmo tornando custoso o projeto por inteiro, é recomendável persistir nestas práticas uma vez que os benefícios são vistos a longo prazo, como em futuras manutenções, por exemplo.

Baseado nos princípios de programação ágil e programação extrema (*Extreme Programming* (XP)), foram estipuladas as seguintes diretrizes para desenvolvimento padrão na área de *Firmware* (FW) e *Software* (SW), segundo Schroeder; Ibáñez e Martin (2004):

- Levantamento de requisitos;
- Projeto e análise do *software*;
- Gerenciamento de versões do código fonte e atualizações;
- Configuração de projetos para plataformas específicas;
- Compilação e junção de código (*linkage*);
- Teste do código em tempo de execução;

- Validar as saídas;
- Documentar o código;
- Rastrear e consertar erros descobertos.

Geralmente o desenvolvimento de sistemas ou componentes é iniciado com a implementação de um núcleo funcional, evoluindo por vários lados, dentre as diretrizes citadas. Neste período, todos os desenvolvedores devem ser encorajados a corrigir problemas nos códigos dos colegas.

Para avaliar o andamento de um projeto existem diversas ferramentas pagas (*Project*, da *Microsoft* e *Enterprise Architect*, da *Sparx Systems*) e gratuitas (*Planner* e *OpenProj*), com possibilidade de criação de gráficos de evolução, previsão, armazenamento de documentos entre outros. Algumas inclusive integram-se com servidores de *e-mail*, navegadores e nomes de usuário.

O desenvolvimento pode ser feito inclusive por meio do uso de um programa simples de edição de arquivos, contudo um ambiente integrado de desenvolvimento (*Integrated Development Environment* (IDE)) é de grande importância por agregar todos os arquivos utilizados, podendo configurar facilmente a plataforma objetivo do código, seus compiladores e a forma de depuração de código durante a observação de erros. A citar, *Visual Studio* (*Microsoft*, normalmente utilizado para *software* para PC), *IAR* (para processadores ARM) e *CodeWarrior* (para processadores *Freescale*).

Programas de gerenciamento de código *Concurrent Version System* (CVS) mantêm registradas todas as versões de código produzidas, incluindo 'o que', 'quem', 'quando' e 'porque' as alterações foram realizadas. Alguns dos sistemas popularmente utilizados são o *Apache Subversion* (SVN) (em sistema Windows) e o *Global Information Tracker* (GIT) (em sistemas *Portable Operating System Interface* (POSIX), como *GNU is Not Unix* (GNU)/Linux). Ambos funcionam também em outros sistemas operacionais, assim como possuem diversas aplicações com interface gráfica disponível.

Durante o ciclo de vida do desenvolvimento ocorrem, principalmente no início do desenvolvimento de uma plataforma, diversas instabilidades que devem ser registradas e corrigidas. Uma das interfaces utilizadas para tal são os chamados *bug trackers*, nos quais as evidências de problemas são encaminhadas diretamente à equipe desenvolvedora responsável pela correção. Como exemplo são citados *RedMine* e *Mantis*.

Schroeder; Ibáñez e Martin (2004) indicam, na figura 5.2, uma descrição simples de funcionamento do desenvolvimento de aplicações de código aberto (*open source*), no qual usuários e desenvolvedores de todo o mundo possuem acesso ao código de um projeto qualquer, mas somente os desenvolvedores podem realizar atualizações no mesmo. Ao serem realizadas alterações/correções em uma ou várias funcionalidades, a plataforma de teste realiza a compilação (geração do código final) e seus testes, assim como sua devida documentação instantaneamente. Seus resultados são por sua vez utilizados para atualizar os colaboradores com as novas mudanças efetuadas.

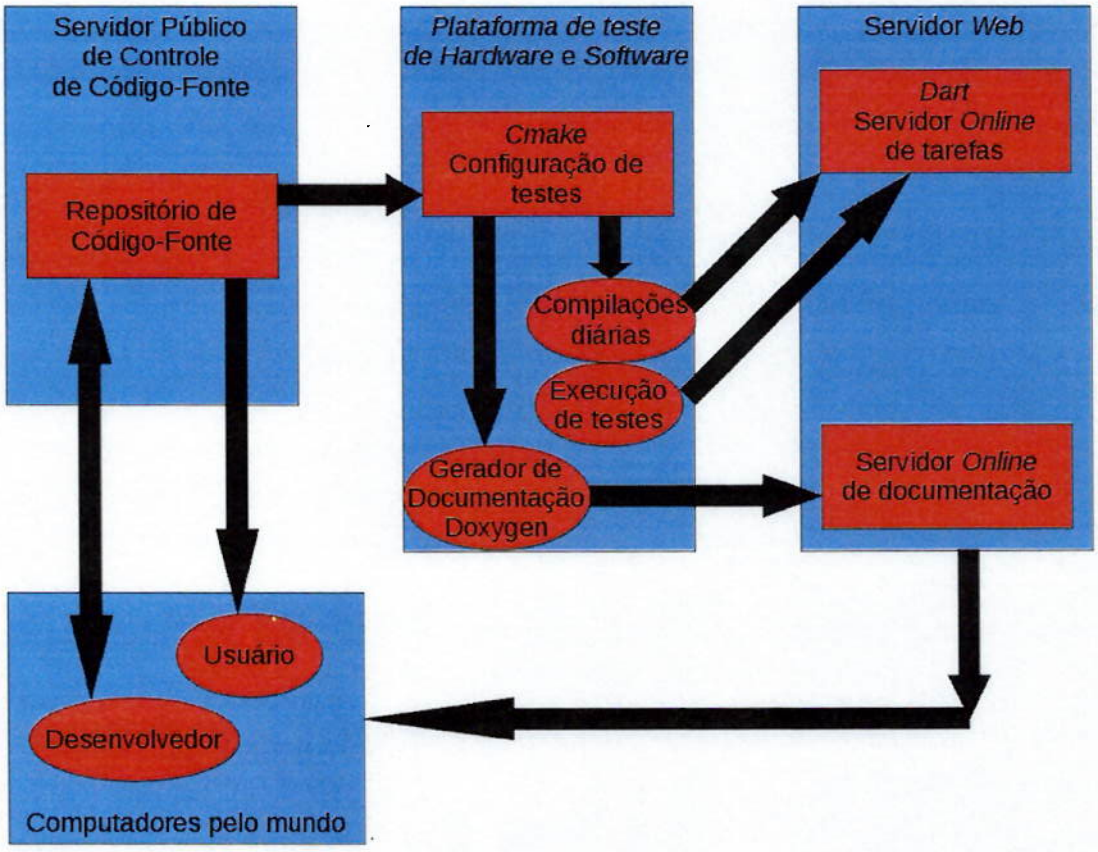

Figura 5.2: Ferramentas para *software* de Schroeder; Ibáñez e Martin (2004)

Ao mesmo tempo em que existe uma vasta gama de programas e IDE para facilitar o desenvolvimento, por vezes é preciso criar ferramentas próprias para adequação à política empresarial ou agilizar os processos. Acima de tudo, é necessário escolher com bom julgamento a implementação de uma ferramenta, pois sua configuração, treinamento de funcionários e adaptação ao novo sistema são demorados.

5.5 PROJETO MECÂNICO

O projeto mecânico de um sistema embarcado é um elemento crítico para a sua produção com sucesso, pois é o produto final deste processo que receberá a atenção visual dos usuários finais. A concepção de uma mecânica esteticamente insatisfatória, frágil ou então perigosa para utilização pode impactar fortemente as vendas do produto final. Por ser composto de diversos elementos físicos de diversas origens (gabinetes, parafusos, cabos e conectores), que não necessariamente são obtidos de um único fornecedor, ou por requerer diversas iterações para a aprovação do objeto final, seu processo deve ser observado com atenção para evitar problemas maiores.

Devido à grande variedade de sistemas legados, domínios e cenários diferentes, é necessário adaptar o fluxo de concepção do *software* e do *hardware* (PILLAT; OLIVEIRA; FONSECA, 2012). Técnicas como o reuso de componentes e módulos (lógicos ou físicos) são consideradas boas práticas, principalmente por evitar criar cada módulo novamente do zero, o que reduz consideravelmente o tempo de entrega do projeto.

Toda mudança nas aplicações executadas em SE não requerem necessariamente alterações de placas ou desenho do produto. Contudo, a descoberta de problemas na concepção de uma placa, a programada obsolescência de um ou mais componentes e/ou plataformas pode levar à tona a necessidade de mudanças físicas, por algumas vezes no SW e por outras na apresentação do produto.

Se as alterações envolvem o projeto físico, é preciso gerenciar corretamente como se dará a implementação das etapas, entre elas: possíveis diferenças no esquema elétrico, tamanho e quantidade de camadas das placas e gabinete do produto. Nestes casos, a revisão do projeto mecânico é primordial, pois a concepção de uma nova versão pode levar inclusive mais tempo que o próprio desenvolvimento das aplicações, atrasando o projeto como um todo.

5.6 CLIENTES E FORNECEDORES

Um problema comum nos projetos é o raro envolvimento do usuário final durante o desenvolvimento do produto (TACKETT; DOREN, 1999). Sem sua presença, não se consegue obter a especificação técnica de modo determinado, de acordo com o domínio e sem am-

biguidades, que são necessários para o seguimento do projeto com facilidade (JUN; RUI; YI-MIN, 2007).

Ao chegar uma nova requisição, é necessário planejar uma tarefa para arquitetá-la, mas nem sempre o planejamento do produto e seu desenvolvimento estão sincronizados (AXELSSON, 2011). Não é comum uma empresa que elicita e documenta os requisitos arquiteturais significantes de modo explícito, sendo, na maioria das vezes, apenas baseado na experiência dos desenvolvedores e arquitetos. Este é mais um exemplo da falta de documentação dos processos e requisitos. Entre as fraquezas encontradas neste tipo de processo, está a dependência de indivíduos mais experientes que trabalham com as informações de modo não documentado. Confiar na memória das pessoas pode resultar em grandes problemas se os requisitos forem necessários, novamente, após um longo período de tempo, sendo às vezes necessário recorrer a anotações imprecisas ou inexistentes (AXELSSON, 2011).

No outro lado do processo, o alinhamento com fornecedores de qualquer tipo de material (físico ou serviços) é essencial para o cumprimento de prazos de entrega e evitar problemas contratuais. A falha em qualquer das partes envolvidas na cadeia de fornecimento pode desencadear quebras de contrato com multas e comprometer a credibilidade de todos perante as vistas do mercado. Neste contexto, a própria empresa torna-se o cliente, sendo essencial e primordial elicitar corretamente os anseios para não passar pelos mesmos problemas citados.

5.7 CONSIDERAÇÕES FINAIS ALÉM DO PROCESSO

Na literatura há muito foco em atividades a se implementar, ao invés de como criá-las. Torna-se então o maior problema, não a falta de padrões, mas sim estratégias para aplicá-las na empresa (JUN; RUI; YI-MIN, 2007). As estratégias para a instanciação do processo variam de acordo com a disposição da equipe de desenvolvimento, além de seu tamanho, localização geográfica, tempo de convívio e limitações técnicas. Exemplos de descrições não foram encontrados pelo autor, contudo uma pesquisa sobre adequação de processos em outras áreas pode ser utilizado no domínio de sistemas embarcados.

Woodward (2007) acredita que, entre os diversos desafios que devem ser encarados durante os próximos 15 anos na área de SE, podem ser citados:

- Complexidade;
- Otimização;
- Interdependência;
- Verificação;
- Ferramentas.

Os desafios declarados são também parte do que o autor acredita ser necessário para a evolução dos sistemas embarcados, pois a utilização de ferramentas adequadas (próprias ou não) de verificação e desenvolvimento aumenta a eficiência, proporcionando mais tempo aos desenvolvedores para sua missão de refletir sobre melhores técnicas para otimização, gerenciamento de complexidade e interdependência de módulos e pessoas. Sua implementação com o sucesso desejado exige perseverança, base técnica e habilidades inter-pessoais.

5.8 CONSIDERAÇÕES FINAIS DO CAPÍTULO

Assim como os diversos elementos mencionados no capítulo 4, existem outras áreas, além das expostas neste capítulo, que necessitam de estudo aprofundado e foco para que sejam implementadas em um ambiente produtivo. Indica-se refletir, também, se é válida a utilização de suas propriedades nos projetos e se este deve ser incluído, quando possível, nos diagramas de referência para o desenvolvimento.

Ainda que as políticas de todas as áreas abordadas estejam estabelecidas, detalhadas e à disposição de todos, é recomendável que revisões para melhoria ou apenas para reciclagem de conhecimento sejam realizadas com todos os membros de todas as equipes submetidos a estes processos. As seções que foram referenciadas neste capítulo estão entre as atividades que merecem atenção.

6 CONSIDERAÇÕES FINAIS

As propostas apresentadas nos capítulos anteriores devem englobar ainda outras variáveis não relatadas neste trabalho, que podem interferir no seu modelo final aplicado. O levantamento dos diversos e inúmeros elementos que podem e devem ser alterados em um ambiente de trabalho deve ser detalhado e criterioso, sempre estabelecendo prioridades de utilização durante sua implementação e evitando conflitos de interesses. Instalações físicas para a realização do trabalho, controle de prazos, controle de tráfego de informações e ferramentas computacionais de comunicação interna e externa são exemplos que poderiam ser investigados e estudados em uma implementação dos processos seguindo este modelo.

6.1 CONCLUSÃO

A especificação de desenvolvimento de sistemas embarcados apresentada, para qualquer tipo e em qualquer ambiente de trabalho é factível, apesar da demanda de intenso e considerável investimento de tempo por parte de vários colaboradores especialistas em diversas competências - cada uma específica, mas essencial ao andamento do projeto - para verificar sua viabilidade e adaptabilidade às regras a que devem ser submetidas. Equipamentos da área médica, por exemplo, demandam rigorosos e demorados testes por se tratar do envolvimento com a vida de pacientes. Ao mesmo tempo, a indústria automotiva demanda confiabilidade por possuir um público exigente, cuja falha em determinados sistemas pode representar catástrofes. No entanto, ambientes de entretenimento mais simples podem possuir pequenos erros sem comprometer o resultado final. A adequação do modelo apresentado é viável para muitos ambientes, ainda que a cobertura de todas as possibilidades seja impraticável. Para contextos peculiares, faz-se necessária a criação de uma nova especificação que, a exemplo da apresentada, exige o estudo do domínio no qual é inserida, podendo mobilizar desde a pessoa focada somente neste projeto a uma equipe numerosa com divisão de tarefas.

6.2 CONTRIBUIÇÃO DO TRABALHO

O trabalho contribui ao coletar e condensar, de forma lúdica e gráfica, diversas experiências de autores na área de *software* embarcado e ambientes similares e unificar as diversas ópticas encontradas em uma proposta direcionada de especificação, com refinamentos acrescentados pelo autor, a ser seguida como guia para o desenvolvimento de sistemas embarcados, que por vezes não é sequer documentado. Sua base teórica consistente obtida da literatura técnica, aliada às experiências práticas do autor sugerem robustez na sua concepção, ainda que não seja possível utilizar integralmente o modelo em todos os ambientes envolvidos com o assunto devido às próprias peculiaridades.

6.3 TRABALHOS FUTUROS

Trabalhos similares na área técnica podem ser encontrados com algum esforço, ainda que não se consiga obter uma forma genérica que agrade a todos os gestores de processos. Trabalhos na área acadêmica com o foco deste (utilização do BPMN) não foram encontrados pelo autor, ainda que sua existência seja provável. Adaptações, generalizações, detalhamentos ou ainda aplicações práticas do modelo apresentado neste trabalho são formas de dar continuidade a este, assim como confrontá-lo com algum documento similar.

REFERÊNCIAS

ANTONIO, E. A.; FERRARI, F. C.; FABBRI, S. C. P. F. A Systematic Mapping of Architectures for Embedded Software. In: BRAZILIAN CONFERENCE ON CRITICAL EMBEDDED SYSTEMS, 2., 2012, Campinas, São Paulo: *Institute of Electrical and Electronics Engineers* (IEEE), 2012. Disponível em: <http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=6226949>. Acesso em: 08 ago. 2012.

AXELSSON, J. Improving the Evolutionary Architecting Process for Embedded System Product Lines. In: IEEE INTERNATIONAL SYSTEMS CONFERENCE, 2011, Montreal, Canada: IEEE, 2011. Disponível em: <http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=5881604>. Acesso em: 08 ago. 2012.

BARISIC, D.; *et al.* Making Embedded Software Development More Efficient with SOA. In: INTERNATIONAL CONFERENCE ON ADVANCED INFORMATION NETWORKING AND APPLICATIONS WORKSHOPS, 21., 2007, Niagara Falls, Canada: IEEE, 2007. Disponível em: <http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=4221179>. Acesso em: 08 ago. 2012.

BEHALEK, M.; SALOUN, P. Usage of Embedded Process Functional Language as a Modeling Tool for Embedded Systems Development. In: INTERNATIONAL CONFERENCE ON INTELLIGENT SYSTEMS, MODELLING AND SIMULATION, 2010, Liverpool, United Kingdom: IEEE, 2010. Disponível em: <http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=5416108>. Acesso em: 08 ago. 2012.

BIZAGI. Buckinghamshire, United Kingdom. **Bizagi Process Modeler**. Disponível em: <www.bizagi.com>. Acesso em: 16 abr. 2012.

CHINOSI, M.; TROMBETTA, A. Modeling and Validating BPMN Diagrams. In: IEEE CONFERENCE ON COMMERCE AND ENTER-

PRISE COMPUTING, 2009, Vienna, Austria: IEEE, 2009. Disponível em: <http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=5210774>. Acesso em: 08 ago. 2012.

CHONG, T.; XINGCHUAN, Y. Workflow-based Embedded System Procedure Management. In: INTERNATIONAL CONFERENCE ON WIRELESS COMMUNICATIONS, NETWORKING AND MOBILE COMPUTING, 5., 2009, Beijing, China: IEEE, 2009. Disponível em: <http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=5303318>. Acesso em: 16 abr. 2012.

GANSSLE, J. G.; *et al.* **Embedded hardware**. Oxford, United Kingdom: Elsevier, 2008. 520 p.

GANSSLE, J. G. **The Art of Designing Embedded Systems**. 2. ed. Oxford, United Kingdom: Elsevier, 2008. 298 p.

GARCIA, I.; ANDREA, I. Using the Software Process Improvement approach for defining a Methodology for Embedded Systems Development using the CMMI-DEV v1.2. In: IEEE INTERNATIONAL CONFERENCE ON COMPUTER AND INFORMATION TECHNOLOGY, 10., 2010, Bradford, United Kingdom: IEEE, 2010. Disponível em: <http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=5578216>. Acesso em: 08 ago. 2012.

GRENNING, J. Applying Test Driven Development to Embedded Software. **IEEE Instrumentation & Measurement Magazine**, v. 10, n. 6, p. 20-25, Dec. 2007. Disponível em: <http://ieeexplore.ieee.org/xpl/tocresult.jsp?isnumber=4428566&punumber=5289>. Acesso em: 08 ago. 2012.

GUPTA, P. Hardware-software codesign. **IEEE Potentials**, v. 20, n. 5, p. 31-32, Dec/Jan. 2001/2002. Disponível em: <http://ieeexplore.ieee.org/xpl/tocresult.jsp?isnumber=21197&punumber=45>. Acesso em: 08 ago. 2012.

GUSTAVSSON, H.; AXELSSON, J. Architecting Complex Embedded Systems: An Industrial Case Study. In: IEEE INTERNATIONAL SYSTEMS

CONFERENCE, 2011, Montreal, Canada: IEEE, 2011. Disponível em: <http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=5929054>. Acesso em: 08 ago. 2012.

HUANG, P. M.; DARRIN, A. G.; KNUTH, A. A. Agile Hardware and Software System Engineering for Innovation. In: IEEE AEROSPACE CONFERENCE, 2012, Big Sky, Montana, USA: IEEE, 2012. Disponível em: <http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6187425>. Acesso em: 08 ago. 2012.

JUN, D.; RUI, L.; YI-MIN, H. Software Processes Improvement and Specifications for Embedded Systems. In: ACIS INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING RESEARCH, MANAGEMENT & APPLICATIONS, 5., 2007, Busan, South Korea: IEEE, 2007. Disponível em: <http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=4296898>. Acesso em: 08 ago. 2012.

KASHIF, H. *et al.* Model-Based Embedded Software Development Flow. In: INTERNATIONAL DESIGN AND TEST WORKSHOP, 4., 2009, Riyadh, Saudi Arabia: IEEE, 2009. Disponível em: <http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=5404096>. Acesso em: 08 ago. 2012.

KOUS, K. Comparative analysis versions of BPMN and its support with Control-flow patterns. In: INTERNATIONAL CONVERTION MIPRO, 33., 2010, Opatija, Croatia: IEEE, 2010. Disponível em: <http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=5533326>. Acesso em: 08 ago. 2012.

KRAMMER, M. *et al.* Improving Methods and Processes for the Development of Safety-Critical Automotive Embedded Systems. In: IEEE CONFERENCE ON EMERGING TECHNOLOGIES AND FACTORY AUTOMATION, 15., 2010, Bilbao, Spain: IEEE, 2010. Disponível em: <http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=5641177>. Acesso em: 08 ago. 2012.

LIGGESMEYER, P.; TRAPP, M. Trends in Embedded Software Engineering.

IEEE Software, v. 26, n. 3, p. 19-25, May/Jun. 2009. Disponível em: <http://ieeexplore.ieee.org/xpl/tocresult.jsp?isnumber=4814945&punumber=52>. Acesso em: 13 nov. 2012.

LIU, H. *et al.* Comparison between Collaborative Business Process Tools. In: INTERNATIONAL CONFERENCE ON RESEARCH CHALLENGES IN INFORMATION SCIENCE, 5., 2011, Gosier, France: IEEE, 2011. Disponível em: <http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6006864>. Acesso em: 08 ago. 2012.

MELIONES, A. N. A Comprehensive Development Guide for Network Embedded Systems. In: WORKSHOP ON INTELLIGENT SOLUTIONS IN EMBEDDED SYSTEMS, 8., 2010, Heraklion, Greece: IEEE, 2010. Disponível em: <http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=5540513>. Acesso em: 08 ago. 2012.

MITSUI, H.; KAMBE, H.; KOIZUMI, H. Use of Student Experiments for Teaching Embedded Software Development Including HW/SW Co-Design. **IEEE Transactions on Education**, v. 52, n. 3, p. 436-443, Aug. 2009. Disponível em: <http://ieeexplore.ieee.org/xpl/tocresult.jsp?isnumber=5191288&punumber=13>. Acesso em: 08 ago. 2012.

MORIMOTO C. E. **Hardware - o Guia Definitivo**. 1. ed. [S.I.]: GDH Press; [S.I.]: Sul Editores, out. 2007. 848 p.

MUEHLEN, M. Z.; HO, D. T. Service Process Innovation: A Case Study of BPMN in Practice. In: PROCEEDINGS OF THE ANNUAL HAWAII INTERNATIONAL CONFERENCE ON SYSTEM SCIENCES, 41., 2008, Walkoloa, Hawaii, USA: IEEE, 2008. Disponível em: <http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=4439077>. Acesso em: 08 ago. 2012.

MURANKO, B.; DRECHSLER, R. Technical Documentation of Software and Hardware in Embedded Systems. In: IFIP INTERNATIONAL CONFERENCE ON VERY LARGE SCALE INTEGRATION, 2006, Nice, France: IEEE, 2006. Disponível em: <http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=4107640>. Acesso em: 08 ago. 2012.

OMG, Object Management Group. **Business Process Model and Notation (BPMN)**, [S.I.:s.n.], 2011. 538 p. Disponível em: <http://www.omg.org/spec/BPMN/2.0>. Acesso em 17 ago. 2012.

PILLAT, R. M.; OLIVEIRA, T. C.; FONSECA, F. L. Introducing Software Process Tailoring to BPMN: BPMNt. In: INTERNATIONAL CONFERENCE ON SOFTWARE AND SYSTEM PROCESS, 2012, Zurich, Switzerland: IEEE, 2012. Disponível em: <http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=6219601>. Acesso em: 08 ago. 2012.

PMI, Project Management Institute. **A Guide to the Project Management Body of Knowledge (PMBOK Guide)**, [S.I.:s.n.], 2012. Disponível em: ¡http://www.pmi.org/GLOBALS/StandardsUpdate.aspx¿. Acesso em 17 ago. 2012.

RICCOBENE, E. *et al.* Designing a Unified Process for Embedded Systems. In: INTERNATIONAL WORKSHOP ON MODEL-BASED METHODOLOGIES FOR PERVASIVE AND EMBEDDED SOFTWARE, 4., 2007, Braga, Portugal: IEEE, 2007. Disponível em: <http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=4149155>. Acesso em: 08 ago. 2012.

RIESCO, D. *et al.* Formalizing the Management Automation with Workflow of Software Development Process Based on the SPEM Activities View. In: INTERNATIONAL CONFERENCE ON INFORMATION TECHNOLOGY: NEW GENERATIONS, 6., 2009, Las Vegas, Nevada, USA: IEEE, 2009. Disponível em: <http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=5070605>. Acesso em: 08 ago. 2012.

SALO, O.; ABRAHAMSSON, P. Agile methods in European embedded software development organisations: a survey on the actual use and usefulness of Extreme Programming and Scrum. **IET Software**, v. 2, n. 1, p. 58-64, Feb. 2008. Disponível em: <http://ieeexplore.ieee.org/xpl/tocresult.jsp?isnumber=4460888&punumber=4124007>. Acesso em: 16 abr. 2012.

SCHROEDER, W. J.; IBÁÑEZ, L.; MARTIN, K. M. Software Process: The key to developing robust, reusable and maintainable open-source Software. In: IEEE INTERNATIONAL SYMPOSIUM ON BIOMEDICAL IMAGING: NANO

TO MACRO, 2004, McLean, Virginia, USA: IEEE, 2004. Disponível em: <http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=1398621>. Acesso em: 16 abr. 2012.

SCOZZI, B. *et al.* Shared mental models among open source software developers. PROCEEDINGS OF THE ANNUAL HAWAII INTERNATIONAL CONFERENCE ON SYSTEM SCIENCES, 41., 2008, Walkoloa, Hawaii, USA: IEEE, 2008. Disponível em: <http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=4439011>. Acesso em: 16 abr. 2012.

SEGAL, J.; MORRIS, C. Developing Scientific Software. **IEEE Software**, v. 25, n. 4, p. 18-20, Jul/Aug. 2008. Disponível em: <http://ieeexplore.ieee.org/xpl/tocresult.jsp?isnumber=4548393&punumber=52>. Acesso em: 08 ago. 2012.

SRINIVASAN, J.; DOBRIN, R.; LUNDQVIST, K. 'State of the Art' in Using Agile Methods for Embedded Systems Development. In: ANNUAL IEEE INTERNATIONAL COMPUTER SOFTWARE AND APPLICATIONS CONFERENCE, 33., 2009, Seattle, Washington, USA: IEEE, 2009. Disponível em: <http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=5254080>. Acesso em: 08 ago. 2012.

TACKETT, B. D.; DOREN, B. V. Process Control for Error-Free Software: A Software Success Story. **IEEE Software**, v. 16, n. 3, p. 24-29, May/Jun. 1999. Disponível em: <http://ieeexplore.ieee.org/xpl/tocresult.jsp?isnumber=16577&punumber=52>. Acesso em: 16 abr. 2012.

WONGWATKIT, C. A Development of Order Processing System: BPMNModel. In: INTERNATIONAL CONFERENCE ON ADVANCED COMMUNICATION TECHNOLOGY, 14., 2012, PyeongChang, South Korea: IEEE, 2012. Disponível em: <http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6174752>. Acesso em: 08 ago. 2012.

WOODWARD, M. V. Challenges for embedded software development. In: IEEE INTERNATIONAL MIDWEST SYMPOSIUM ON CIRCUITS AND SYSTEMS, 50., 2007, Montreal, Canada: IEEE, 2007. Disponível em:

<http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=4488660>. Acesso em: 08 ago. 2012.

WU, J.; PENG, Y.; ZHANG, W. Study on a Flexible Workflow Technology and Its application. In: ASIA-PACIFIC CONFERENCE ON COMPUTATIONAL INTELLIGENCE AND INDUSTRIAL APPLICATIONS, 2009, Wuhan, China: IEEE, 2009. Disponível em: <http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=5406466>. Acesso em: 08 ago. 2012.