DAS Departamento de Automação e Sistemas
CTC Centro Tecnológico
UFSC Universidade Federal de Santa Catarina

Desenvolvimento Solução MES para unidade de Aços Longos

Relatório submetido à Universidade Federal de Santa Catarina como requisito para a aprovação na disciplina

DAS 5511: Projeto de Fim de Curso

Henrique Borba Behr

Florianópolis, Agosto de 2013

Desenvolvimento Solução MES para unidade de Aços Longos

Henrique Borba Behr

Esta monografia foi julgada no contexto da disciplina
DAS5511: Projeto de Fim de Curso
e aprovada na sua forma final pelo
Curso de Engenharia de Controle e Automação

Max Hering de Queiroz

_____
Assinatura do Orientador

Banca Examinadora:

Wislann Alves dos Santos
Orientador na Empresa

Max Hering de Queiroz
Orientador no Curso

Wislann Alves dos Santos
Orientador na Empresa

José Eduardo Ribeiro Cury
Avaliador

Guilherme Bencke Teixeira da Silva
Juliano Norio Marcondes Toda
Debatedores

Agradecimentos

Agradeço:

Ao professor Max Hering de Queiroz pela orientação.

Aos engenheiros de controle e automação João Zaiden e Wislann Alves dos Santos e a Engenheira de Software Turah Xavier de Almeida pela supervisão do meu trabalho.

À equipe de desenvolvimento do MES Aços Longos pelo suporte e auxilio.

À Beatriz Gietner pelo apoio e revisão da monografia.

Resumo

Este PFC foi desenvolvido na empresa Radix Engenharia e Software, sendo esta uma empresa de serviços que se propõe a oferecer ao mercado serviços e soluções de engenharia e software. Sua principal participação no mercado sempre foi o de prestação de serviços para a indústria de petróleo e petroquímicas, entretanto existem também clientes das mais variadas áreas, os quais demandam outras soluções de engenharia, como: siderurgia, mineração, entretenimento, alimentos e transporte.

Esta monografia registra o desenvolvimento de um sistema de execução de manufatura (MES) para uma nova fábrica de Aços Longos em construção. O projeto foi feito em cima de uma plataforma web que será acessado pelo navegador Internet Explorer. Para isso foram utilizadas diferentes ferramentas, as quais se destacam: a linguagem de programação C#, o framework web ASP .Net MVC3 (para construir o servidor onde estão implementadas), o sistema gerenciador de banco de dados relacional SQL Server (realização da persistência dos dados), as linguagens Html, Javascript e jQuery (para exibir os dados e interagir com o usuário), requisições AJAX (realizar comunicação entre o browser e o servidor), WebService e o protocolo PI (realizar a comunicação com o SAP) e um WebService com o protocolo XQI (para comunicação com o nível dois e demais sistemas periféricos).

O trabalho de desenvolvimento do MES envolve um meio termo entre o nível dois (PLC's, Scada, controles de máquinas) e o nível superior de controle de vendas e produção, além de uma série de informações para serem controladas e armazenadas internamente. O MES inclui diversas funcionalidades, entre as principais destacam-se: programação e controle da produção, controle de estoques intermediário e final, contabilização de uso dos equipamentos para a manutenção, controle de paradas da fábrica, inspeções e decisões de qualidade, turno e rastreabilidade de lotes.

Abstract

This project was developed in “*Radix Engenharia e Software”*, which is a service company that intends to offer to the market services and engineering solutions and software. Its main market share has always been to provide services for the oil and petrochemical industry, however there are also clients from many areas, which require other engineering solutions: Steel, Mining, Entertainment, Food and Transportation, for example.

This monograph records the development of a manufacturing execution system (MES) for a new factory in Long Steel construction. The project was done over a web platform that will be accessed by Internet Explorer browser. For this we used different tools C# Programming Language ASP.Net MVC3 (framework) to build the server where they are implemented, SQL Server (management system relational database) to perform data persistence. Using Html, Javascript and jQuery to display data and interact with the user. AJAX requests to communicate with the server browser. WebService and the IP protocol to perform communication with the SAP and a WebService with XQI protocol to communicate with the two levels.

The MES's development work revolves around a relation between the level two (PLC, SCADA, Controls Machines) and the higher level of sales and production control, and a lot of information to be tracked and stored internally. The MES includes several features, among the principal we can highlight: scheduling and production control, inventory control, intermediate and final accounting of use of the equipment for maintenance, control charts of factory inspections and quality decisions, shift and traceability batch.

Sumário:

Simbologia

A seguir:

PFC – Projeto Final de Curso.

MES – Sistema de execução da Manufatura.

SQL – *Structured Query Language*

HTML – *Hyper Text Markup Language*.

PLC – Controlador lógico Programável.

JS - JavaScript

AJAX – *Assynchronous Javascript and XML*.

SCADA – *Supevisory Control and Data Acquisition*.

jQuery - Biblioteca de Javascript, usada para facilitar o desenvolvimento e melhorar a interoperabilidade entre navegadores e versões diferentes.

ERP – *Enterprise resource planning*

SAP – Implementação de um ERP, mundialmente famosa e aceita.

SGBD - Sistema de gerenciamento de banco de dados.

OO – Orientado a Objetos.

IDE – Ambiente de desenvolvimento Integrado.

.NET – *Dot Net Framework.*

CLR – *Common Language Runtime*

MVC – *Model View Controller*

CRUD - Create*, read, update, delete*

SAP - *Systeme, Anwendungen und Produkte in der Datenverarbeitung* (Sistemas, Aplicativos e Produtos para Processamento de Dados).

Capítulo 1: Introdução

A indústria do aço brasileira é uma das mais competitivas do mundo. Vem gerando anualmente mais de R$ 45 bilhões em valor adicionado para o Brasil (sendo responsável por um saldo comercial acima de US$ 4 bilhões) o que significa aproximadamente 18% do saldo comercial total do país. Além da grande movimentação de capital, vale ressaltar que as indústrias de aços empregam atualmente cerca de 110 mil pessoas. O aço está presente em diversos segmentos da indústria, entre os quais: Automotivo, Construção Civil, Embalagens, Linha Branca e OEM.

Existe também uma grande competitividade de mercado, e esta tem exigido que as indústrias aumentem cada vez mais sua eficiência de operação, ao mesmo tempo reduzindo os custos e acelerando o ciclo produtivo sem comprometer a qualidade final do produto.

A fim de atingir tais objetivos, sistemas industriais de controle já são largamente empregados, assim como sistemas corporativos. Apesar de todo o avanço nesses sistemas, a camada intermediária entre o chão de fábrica e gestão de negócios é, por vezes, ineficiente. Se, por um lado, os sistemas corporativos focam nas operações financeiras, contábeis e logísticas, os sistemas supervisórios e de controle atuam na gestão da operação dos equipamentos e processos ao nível do chão de fábrica. O sistema MES surge então como uma importante ferramenta para monitorar o processo produtivo, auxiliando na resolução de problemas de produção e na melhoria contínua do desempenho.

A integração com os sistemas de automação e corporativos permite ao MES coletar todas as informações relevantes ao gerenciamento da produção e registrá-las em um único local e de modo organizado. O registro pode ser feito de modo manual, através de telas de apontamento ou automaticamente a partir de interfaces com sistemas PLC, Supervisórios, PIMS, LIMS e SAP, entre outros. Os dados coletados são processados pelo MES, ficando disponíveis para consultas futuras e para a geração de indicadores para uma gestão da produção mais eficaz. Este é, portanto, um dos pontos diferenciais em termos de benefícios do sistema MES, já

que ele fornece informações para tomada de decisão ou para investimentos em ações que busquem melhorias contínuas. Na prática, tem se mostrado de grande valia para o processo de melhoria contínua e para uma gestão eficiente da produção.

Há funcionalidades típicas comumente encontradas em sistemas MES. A especificação completa de um MES depende, no entanto, dos processos característicos de cada negócio.

De maneira geral, o MES deve auxiliar os gestores da produção no gerenciamento de recursos, metas, custos e qualidade, de forma a maximizar ganhos e produtividade. O MES deve interagir com a programação fina, aumentando a velocidade de reação a mudanças na produção e provendo informações confiáveis e em tempo real que melhorem a qualidade das decisões.

O MES deve criar uma visão geral do fluxo de manufatura e aumentar não só a velocidade do ciclo de produção como também sua eficiência, a partir da identificação e redução de tempos de fila, tempos mortos de processo, estoques e paradas de equipamentos.

Este trabalho visa mostrar como foi realizado o projeto de desenvolvimento do MES para uma nova fábrica de aços longos em construção, sempre focando na parte do projeto que se relaciona ao curso de Engenharia de Controle e Automação com o objetivo de consolidar os conhecimentos adquiridos durante o curso.

O primeiro capítulo desta monografia trata da contextualização do projeto desenvolvido na RADIX dentro do curso de Engenharia de Controle e Automação. Em seguida, no segundo capítulo, é apresentada uma visão geral sobre a empresa em que o trabalho foi desenvolvido, o cliente e a fábrica onde a solução será implantada. Já no capítulo três há o desenvolvimento da explicação acerca de metodologias de desenvolvimento utilizadas, framework e ferramentas. No quarto capítulo está sucintamente exibida a estrutura do projeto e o trabalho das diversas equipes que fizeram o software. Para terminar, no quinto capítulo encontra-se o detalhamento das funcionalidades mais importantes que foram desenvolvidas por mim no projeto.

1.1: Contextualização com o Curso

O projeto desenvolvido se enquadra em diversas áreas do curso de Engenharia e Controle e Automação:

- Boas práticas de programação, modularização de código, boa legibilidade do código e separação de funções (dados, controladores, views) são alguns dos assuntos aprendidos nas disciplinas de base, como em Introdução à Informática para Automação e Fundamentos da Estrutura da Informação;
- Análise de requisitos, aplicação de metodologias de desenvolvimento e programação orientada a objetos foram estudados em Metodologia para Desenvolvimento de Sistemas;
- Conceitos de programação concorrente vistos na disciplina de Informática Industrial II;
- Conceitos da disciplina de Processos de Fabricação Metal-mecânica como fundição e laminação de aço. Também houve um conhecimento mais amplo sobre a área de controle de manufatura, planejamento de produção e qualidade, conceitos estes estudados em Sistemas Integrados de Manufatura;
- Técnicas de interoperabilidade entre sistemas aprendidos na disciplina de Integração de Sistemas Corporativos.

Capítulo 2: Empresa

Neste capítulo serão apresentadas a empresa onde o projeto foi desenvolvido, um pouco sobre o processo de fabricação do cliente e a planta onde o projeto vai ser executado.

2.1: Radix Engenharia e Software

A Radix foi fundada no Rio de Janeiro em 2010, com estimativa de somar 100 funcionários já no primeiro ano de atuação. Ela nasceu como uma empresa especializada em serviços e soluções de TI e engenharia para indústrias de processo (principalmente óleo e gás, petroquímicas e químicas), mas ao longo desses 3 anos de funcionamento, devido a excelência de seus projetos, conseguiu uma gama de clientes dos mais variados setores dos quais alguns principais:

Óleo e gás/Petroquímica: Petrobras, AkerSolutions, OGX e Queiroz Galvão.

Siderurgia e Mineração: CSN e Vale.

Entretenimento: Rede Globo e iMusica.

Química: FosBrasil.

Farmacêutica: Laborvida.

Transporte: Supervia.

Com apenas três anos de mercado, já é uma empresa posicionada fortemente no mercado de engenharia. Suas principais áreas de atuação são:

- Projetos de Engenharia: projetos conceituais, avaliações técnico-econômicas de unidades de processo. Simulação e otimização de processos.
- TI Industrial: serviços para informação e automação de unidades industriais. Auditoria e sintonia de malhas de controle, controle

avançado, gerenciamento de alarme, planejamento da produção, gerenciamento de dados.

- Software: soluções tecnológicas sob demanda para diversos segmentos, como indústrias, portais, mídia e entretenimento. Aplicações Web, aplicativos móveis nativos (iOS, Android) e Web responsivos, além de serviços de especificação de requisitos, testes e avaliação de segurança da informação.

Figura 1 - Áreas de Atuação da Radix

2.2: Siderúrgica

O presente trabalho se situa no segmento de aços longos, numa das maiores empresas do setor, fundada em 1941 pelo então presidente Getúlio Vargas. Sua principal fábrica está situada na cidade de Volta Redonda, RJ.

A siderúrgica é uma empresa de capital aberto, que atua no segmento e em diversos outros setores ligados à siderurgia (mineração, cimento, logística e

energia). Ela teve no ano de 2012 um faturamento de aproximadamente 16 Bilhões de reais.

Ela abrange (na área de siderurgia) toda a cadeia produtiva do aço, desde a extração do minério de ferro até a produção e comercialização de uma diversificada linha de produtos siderúrgicos de alto valor agregado.

No ano de 2012, a usina produziu 4,8 milhões de toneladas de aço bruto, enquanto comparativamente a produção de laminados atingiu 4,7 milhões de toneladas.

A empresa vem trabalhando na diversificação de suas atividades siderúrgicas com a entrada no segmento de aços longos por meio da construção de uma unidade em Volta Redonda, tendo esta capacidade de produção de 500 mil toneladas anuais (abrangendo desde vergalhões a fios-máquina).

A seguir será exposta a estrutura de produção de uma fábrica de aços longos, onde o MES será implantado.

2.2.1: Aciaria:

Aciaria é a unidade de uma usina siderúrgica onde existem máquinas e equipamentos voltados para o processo de fundição do ferro gusa e sucatas.

No processo de aciaria elétrica é produzido o aço fundido que é utilizado na produção de tarugos, os quais são produtos semiacabados que irão posteriormente ser transformados nos produtos finais por outras unidades.

Figura 2 - Interior de uma Aciaria

2.2.1.1: Forno elétrico a Arco

O processo se inicia quando sucata é misturada ao ferro-gusa e então são ambos inseridos no forno elétrico a arco com o objetivo de serem fundidos por meio de um arco elétrico.

No forno elétrico, três eletrodos de grafite formam um arco elétrico cuja energia transferida é capaz de aquecer a carga metálica e promover sua completa fusão. Após a fusão, inicia-se o tratamento do metal líquido e alguns elementos indesejáveis são removidos como, por exemplo, o fósforo.

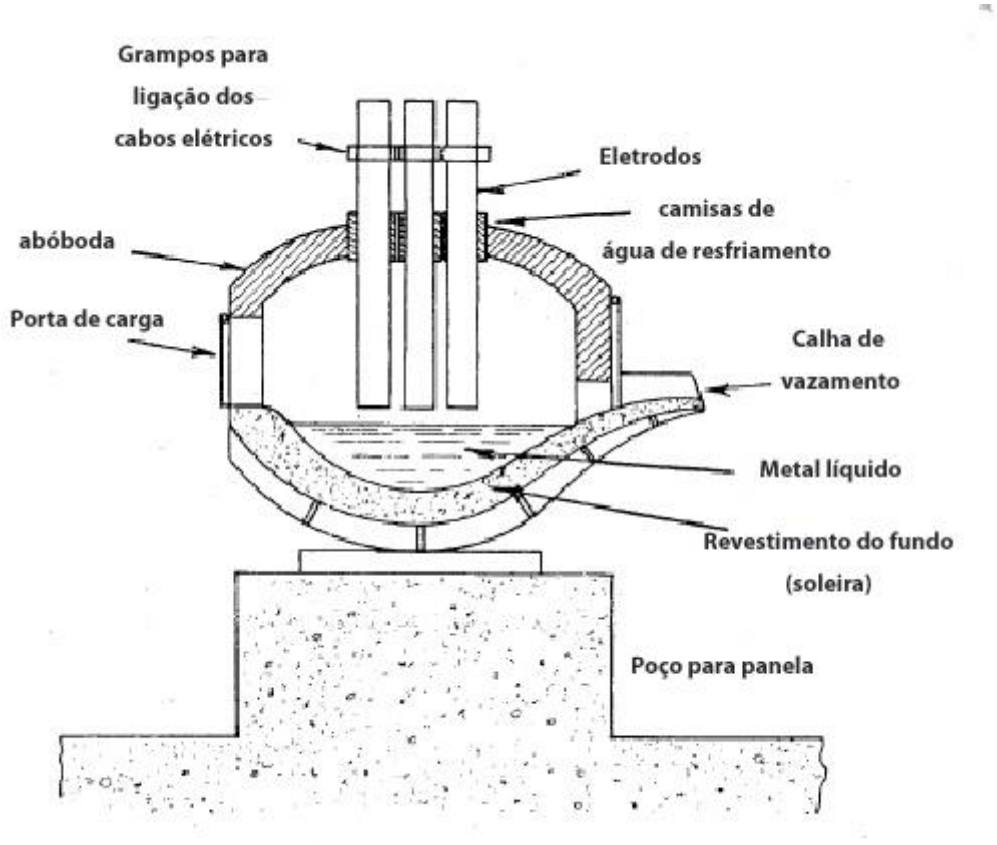

Figura 3 - Forno elétrico a Arco

2.2.1.2: Forno Panela:

Após a fusão e início do processo de refino no forno elétrico a arco, o aço líquido é transferido para uma panela. No forno-panela é possível controlar a temperatura do metal líquido e, mediante injeção de um gás inerte, propiciar

condições controladas de agitação. Assim, ocorre a homogeneização térmica do metal e de todas as ferro-ligas adicionadas para atingir a composição química desejada do aço.

Figura 4 - Forno Panela

2.2.1.3: Lingotamento Continuo:

O aço refinado passa pelo próximo processo de lingotamento contínuo: o aço liquido é forçado por moldes de resfriamento para se solidificar e então atravessa um cortador onde adquire o formato final de tarugo. O tarugo é um produto semiacabado, que já pode ser vendido dependendo das condições do mercado, ou ser usado para abastecer as próximas unidades para produzir um produto final de maior valor agregado.

Figura 5 - Lingotamento Continuo

2.2.1.4: Pátio de Tarugos:

Como processo intermediário, os tarugos são armazenados no pátio até a hora de serem abastecidos na unidade de laminação.

2.2.2: Laminação:

Os tarugos que vêm do pátio são abastecidos na unidade de laminação. O processo de laminação inicia com o reaquecimento do tarugo em um forno que eleva sua temperatura até uma faixa entre 1000 e 1200 °C.

Para a elaboração dos produtos longos o aço passa por um trem de laminação. Lá ele é espremido entre canais cada vez mais finos de cilindros alocados horizontalmente e verticalmente. No fim do processo são produzidos diferentes tipos de aço vergalhões/feixes com formas de acordo com a especificação, ou o aço é enrolado para produzir bobinas.

Figura 6 - Processo de Laminação

Figura 7 - Bobinas produzidas na fase de laminação

2.2.3: Acabamento:

Por último, os feixes e as bobinas produzidos podem ainda passar pelo processo final de acabamento onde então aqueles podem vir a passar por um processo de dobragem e estes a passar pelas máquinas de endireitadeira já no setor final de acabamento.

2.3: Lugar do MES na fabricação:

O MES é um sistema que visa coletar e agrupar informações de sistemas de produção (nível 2). Ele é capaz de organizar e armazenar dados para então

disponibilizá-los aos supervisores da produção que, por sua vez, irão analisar o desempenho e tomar uma decisão mais precisa. Em seguida agrupam-se os dados da produção para envia-los para um nível acima, denominado sistema de planejamento e vendas “ERP”.

O MES também realiza o caminho inverso: ele recebe informações genéricas de produção do sistema de planejamento e possibilita aos operadores montar a programação que será realizada a cada instante e envia essa informação ao nível 2, (que irá produzir de acordo com a programação).

Além destas, o sistema tem outra gama de funcionalidades próprias que são necessárias para o bom funcionamento da fábrica:

- Cadastros Gerais: são as informações gerais usadas pelos demais módulos da aplicação e também informações que o nível 2 precisa porém não tem como cadastrar, como por exemplo: locais, equipamentos de laboratório e justificativas de paradas.
- Segurança: contempla lógicas de segurança da aplicação, assim como controle de permissões e acesso com autenticação via rede.
- Histórico de banco de dados: todos os dados da aplicação tem que ficarem salvos no banco de produção, os quais de tempos em tempos serão migrados do banco operacional para um banco histórico com o objetivo de garantir o bom desempenho.
- Auditoria: onde todas as mudanças realizadas no banco pelo usuário do sistema deverão ser registradas.
- Módulo de produção: contempla as funcionalidades de programação e reprogramação da produção das áreas de aciaria, laminação e acabamento, além de ser possível observar também o acompanhamento da produção.
- Rastreabilidade da produção: possibilita ao usuário consultar os dados de um lote antigo. Com essa funcionalidade é possível rastrear e mostrar cada insumo utilizado para se produzir um lote e os seus dados de qualidade apontados no MES.

- Apontamento de produção: é possível apontar as entradas (insumos, matérias-primas) e as saídas (produtos) para cada fase da produção.
- Comparativo realizado x programado: poderá ser comparado o que está de fato sendo produzido com o que está sendo programado.
- Tratamento de ocorrências: essa funcionalidade contempla o registro e consulta das ocorrências de anormalidades nas operações de produção, as quais também serão usadas para fins de rastreabilidade da produção.
- Reclassificação de aço: o sistema permite a reclassificação de lotes da aciaria para materiais diferentes daqueles no qual estavam primeiramente programados.
- Controle de sucata: nesta funcionalidade é possível o apontamento de sucatas geradas em cada operação da produção para todas as áreas, como também o envio destes ser enviados ao SAP para consolidação dos custos de produção.
- Apontamento/Inspeção de qualidade dos produtos: análises de produtos intermediários e finais. Também é possível o posterior lançamento de uma inspeção com o objetivo de averiguar se o bloqueio de material deve ser mantido.
- Gestão de análise laboratorial: apontamento manual e a importação automática (sistemas de automação e SAP) de análises laboratoriais executadas para posterior verificação de seus resultados
- Módulo de paradas: é responsável pela gestão das paradas de equipamentos ocorridas no ambiente de produção, existe tanto na versão manual quanto automática.
- Módulo de manutenção (Campanha): neste módulo estão as funções de gerenciamento da manutenção de equipamentos de campanha, cilindros e outros consumíveis.
- Módulo de inventários (estoques e expedição): funções de gerenciamento de estoques e expedição do sistema.

- Emissão / Reimpressão de etiquetas: etiquetas de produtos intermediários e finais numa impressora Zebra.
- Relatórios: dentre os principais destacam-se o relatório de produção e o relatório de rendimento.

Capítulo 3: Fundamentação teórica

No escopo do projeto de desenvolvimento de MES para siderúrgica foram usados diversos conteúdos das disciplinas na área de software do curso de Engenharia de Controle e Automação. Neste capítulo serão apresentados tópicos referentes à área de programação e TI industriais que auxiliarão no entendimento deste projeto.

3.1: Ambiente de Desenvolvimento Integrado (IDE):

Preocupados com o aumento da competitividade no mercado de software, empresas competitivas aderiram aos IDEs como ferramentas indispensáveis de produção. O IDE é um programa de computador utilizado para aumentar a produtividade dos desenvolvedores de software bem como a qualidade desses produtos. Alguns dos melhores IDEs possuem diversas ferramentas embutidas, as quais ajudam o desenvolvedor, por exemplo:

- Editor: edita o código fonte e usa uma gama de cores para diferenciar: classes, funções, variáveis, melhorando a visualização e o entendimento. Capaz de indicar erros no programa antes mesmo de ele ser compilado.
- Compilador: tem como principal função fazer a tradução do código fonte em um formato que o compilador pode entender.
- *Linker*: liga o código objeto às bibliotecas e transforma tudo em um único executável.
- Carregador: carrega o programa na memória e executa com apenas um clique para facilitar o trabalho.
- Depurador: ajuda o programador na verificação e correção de erros.
- Distribuição: auxilia o processo de criação do instalador.

- Refatoração: combina poderosas ferramentas de *refatoração* que permite substituir nomes em diferentes arquivos de diferentes projetos com um clique.

3.1.1: Ambiente do Projeto

O ambiente de desenvolvimento utilizado na codificação do software foi o Microsoft Visual Studio 2010, sendo este um ambiente integrado de desenvolvimento construído pela Microsoft. O Visual Studio é flexível, possibilitando desenvolver desde aplicações de console, aplicações com interface gráfica e até websites (aplicações e serviços web). Ele suporta diferentes linguagens, como as básicas C e C++, até linguagens mais complexas orientadas a objeto, como Visual Basic, C# e F#. Uma das vantagens de se utilizar o ambiente de desenvolvimento do Visual Studio é a facilidade de integração com os outros serviços da Microsoft: banco de dados, usuários da rede da MS e também a exportação de dados para o Excel.

3.2: Linguagem Programação

O advento do computador pessoal deu o impulso definitivo à área de computação. Hoje em dia é indiscutível a presença e a importância dos sistemas computadorizados no mundo todo, tanto no setor produtivo quanto para uso pessoal.

Para atender as necessidades do mercado, sistemas complexos com milhares de linhas de código são produzidos todos os anos. Tornou-se imperativo lidar com a crescente complexidade do desenvolvimento de software. O paradigma da Orientação a Objetos (OO) é um dos maiores avanços na área de software, sendo também uma evolução natural da programação estruturada. Com seu advento, foi possível visualizar um programa como sendo uma rede de

relacionamento entre objetos, dividindo a complexidade do sistema em pequenas unidades lógicas, que, por sua vez, tornou-as mais facilmente gerenciáveis.

A programação orientada a objetos é uma solução mais natural e intuitiva porque no mundo físico cada objeto tem um papel definido (por exemplo um garfo ou uma caneta). Podemos dizer que todo objeto tem uma ou mais finalidades e características físicas representadas na programação orientada a objetos como métodos e atributos, respectivamente.

3.2.1: Reutilização

A reutilização está baseada na padronização, sendo esta adotada há longa data em toda indústria moderna, indo desde o projeto de carros até televisores, computadores, etc. A padronização traz inúmeras vantagens, entre elas citam-se:

- redução de custo;
- aumento da confiabilidade;
- facilidade de reparo e manutenção.

3.2.2: Vantagens da Orientação a Objetos

- Há maior facilidade para reutilização de código (esta é a principal vantagem da programação orientada a objetos).
- Possibilita a agregação de módulos prontos porque estes podem ser agregados ao longo do projeto (através da importação de suas funcionalidades) mesmo tendo seu desenvolvimento separado.
- Possibilita ao desenvolvedor trabalhar em um nível mais elevado de abstração.
- Exige um menor custo de desenvolvimento (pode-se desenvolver mais por menos). Os módulos já prontos podem ser agregados a novos projetos.

3.2.3: Linguagem Utilizada - C#

A linguagem C# faz parte do conjunto de ferramentas oferecidas na plataforma .NET (explicada posteriormente em 3.5.2:) e surge como uma linguagem simples, robusta, orientada a objetos, fortemente tipada e altamente escalável (a fim de permitir que uma mesma aplicação possa ser executada em diversos dispositivos de hardware, independentemente destes serem PCs ou um dispositivo móvel).

O C# tem raízes em C, C++ e Java, adaptando os melhores recursos de cada linguagem e acrescentando novas capacidades próprias. Ele fornece os recursos que são mais importantes para os programadores, como programação orientada a objetos, *strings*, elementos gráficos, componentes de interface com o usuário (GUI), tratamento de exceções, múltiplas linhas de execução, multimídia (áudio, imagens, animação e vídeo), processamento de arquivos, estruturas de dados pré-empacotadas, processamento de banco de dados, redes cliente/servidor com base na internet e computação distribuída.

3.3: JavaScript

JavaScript (JS) é uma linguagem de programação interpretada (não precisa ser compilada) que roda diretamente nos navegadores *web*, permitindo que scripts possam ser executados do lado do cliente e interajam com o usuário sem a necessidade deste script passar pelo servidor. O JS tem a capacidade de controlar o navegador, enviar requisições assíncronas para o servidor e alterar o conteúdo do documento exibido.

É atualmente a principal linguagem para programação *cliente-side* em navegadores web. Foi concebida para ser uma linguagem script com orientação a objetos baseada em protótipos e dinâmica. Criada em 1995 com o nome de *LiveScript,* foi primeiramente lançada para o navegador *Netscape*. Este já não se encontra mais ativo, porém o *JavaScript* adquiriu ampla aceitação como linguagem de navegador.

As principais características desta linguagem são:

- Tipagem dinâmica: tipos são associados como valor e não como variáveis. Por exemplo, a variável “x” pode ser associada a um número e mais tarde associada a uma cadeia de caracteres.
- Baseada em objetos: objetos JS são *arrays* associativos, aumentados com protótipos. Os objetos também são dinâmicos, suas propriedades e seus valores podem ser adicionados, alterados ou destruídos em tempo de execução.

O uso primário do JS é o de escrever funções que serão incluídas nas páginas HTML, e essas funções irão interagir com o usuário de diversas formas como abrindo uma nova janela (*pop-up)*, validando valores de um formulário para garantir que são aceitáveis antes de enviar ao servidor, mudando imagens e melhorando a interface gráfica. O fato do JS rodar localmente no navegador do usuário, e não em um servidor remoto, possui dois benefícios: o usuário tem uma resposta muito mais ágil do que se tivesse que esperar a resposta do servidor, e o servidor fica com uma carga menor, garantindo maior velocidade.

3.3.1: jQuery

Antigamente escrever código em *JavaScript* era uma tarefa tediosa e custosa, resultando em altos custos para se desenvolver uma pequena funcionalidade. Com o passar do tempo, diversos programadores desenvolveram bibliotecas de auxilio à programação. A jQuery é uma das mais famosas bibliotecas que se tem no mercado. Ela foi criada por *John Resig* em 2006, e ele definiu sua criação assim: “o foco principal da biblioteca jQuery é a simplicidade. Por que submeter os desenvolvedores ao martírio de escrever longos e complexos códigos para criar simples efeitos?”.

O jQuery é uma maneira simples e fácil de escrever aplicações *JavaScipt*. As principais vantagens do uso de jQuery sobre o *JavaScipt* tradicional são:

- Código “*open source*” (gratuito).
- Mundialmente utilizado em diversos sites e soluções web.

- Acesso direto a qualquer componente da página, ou seja, não há necessidade de várias linhas de código para acessar determinados objetos. Duas comparações são feitas na Tabela 1 - Comparação Sintaxe Javascript vs jQuerya primeira mostra como selecionar todos os elementos da página com o mesmo nome (um elemento pode ser um botão, uma linha de uma tabela, entre outros) e a segunda mostra como é fácil alterar um atributo de um elemento.
- Manipulação de conteúdos com algumas poucas linhas de código.
- Suporte para toda a gama de eventos de interação com o usuário sem limitações impostas pelos navegadores.
- Possiblidade de inserir uma grande variedade de efeitos de animação com uma simples linha de código.
- Simplificação na criação de scripts.
- Emprego *cross-browser*. A mais poderosa ferramenta do jQuery é abstrair o desenvolvedor do navegador para o qual se está programando. Suas funções tem o mesmo resultado, embora por baixo a biblioteca adeque seu funcionamento de acordo com o navegador em que foi aberto e versão em que se encontra.

Sintaxe JavaScript	**Sintaxe jQuery**
document.getElementsByTagName("p")	$("p")
document.getElementById(“um”).setAttribute(“class”, “cor”)	$("#um").attr("class", "cor")

Tabela 1 - Comparação Sintaxe Javascript vs jQuery

3.4: Modelo de dados

Em muitos sistemas informatizados é necessário armazenar informações em bancos de dados, e pode-se constatar isso ao observar que nas últimas décadas o banco de dados se tornou o coração de muitos sistemas. A informação é muitas

vezes a coisa mais valiosa das empresas e mantê-las e poder acessá-las sempre que necessário é primordial para a tomada de decisões importantes.

Além disso, é importantíssimo controlar o acesso a essas informações e não deixar que elas vazem. Evitar a perda de informações, fazendo backups periódicos é uma forma de garantir que as informações relevantes não sejam perdidas.

O gerenciamento de um banco de dados é uma peça fundamental para ajudar a empresa a ter sucesso, mas também pode levá-la ao fracasso. Para garantir a consistência dos dados é preciso controlar o acesso e manter os dados seguros. Com essa finalidade foram criados os Sistemas de Gerenciamento de Bancos de Dados (SGBD).

Figura 8 - Modelo de funcionamento de Banco de dados

3.4.1: Tecnologia Utilizada

Devido aos requisitos, por parte do cliente, usou-se o SGBD (desenvolvido pela Microsoft) SQL Server.

3.4.2: Documentação e Construção do Modelo

Devido à alta utilização de banco de dados relacionais na imensa maioria das soluções de TI e Automação pelo mundo todo, encontram-se no mercado de software diversos programas de auxilio à construção e documentação de um banco de dados.

Usa-se de interface gráfica para montar as tabelas, construir as relações 1x1, 1xN “Foreign Keys” e declarar “Unique Keys”, e através disso o programa constrói automaticamente o código na linguagem SQL. Esses softwares reduzem o custo de desenvolvimento se tornando essencial utilizar um desses para qualquer projeto de média a alta complexidade.

No projeto foi utilizado o software “Enterprise Architect”, desenvolvido pela empresa Sparx Systems, sendo esta uma empresa australiana especializada em desenvolver software de auxílio a projetos.

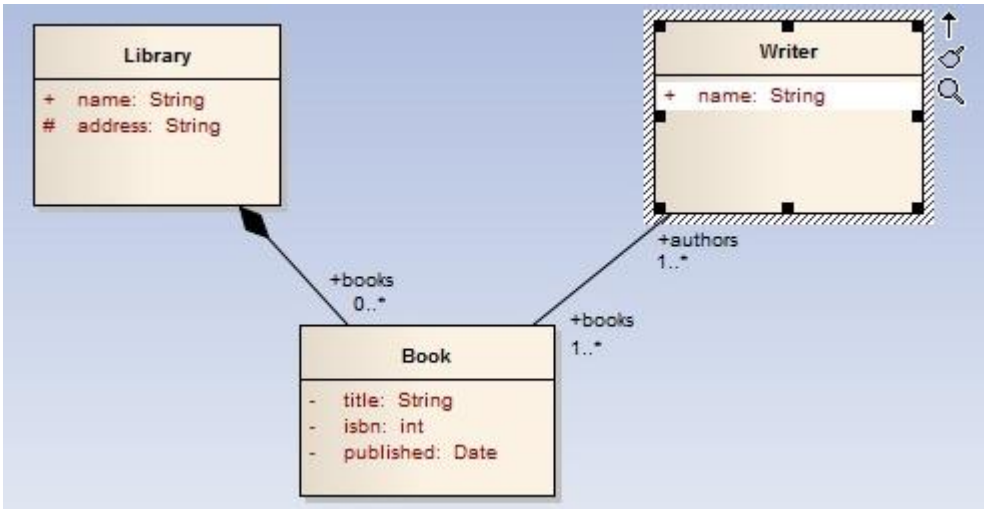

Figura 9 - Interface Gráfica do EA para criação das tabelas de Banco de Dados

3.5: Framework

Um dos principais objetivos da Engenharia de Software é o reuso. Através da reutilização de software obtém-se o aumento da qualidade e redução do esforço de desenvolvimento. A orientação a objetos fornece funcionalidades para que classes possam ser reutilizadas, bem como métodos, por meio de seus mecanismos de herança e polimorfismo. Componentes de software definem unidades reutilizáveis que oferecem serviços através de interfaces bem definidas. “Padrões de projeto” é outra abordagem mais abstrata que objetiva a reutilização de projetos de soluções para problemas recorrentes.

Além destas formas de reutilização, a tecnologia de frameworks possibilita que uma família de produtos seja gerada a partir de uma única estrutura que captura os conceitos mais gerais da família de aplicações.

Um framework é um projeto de software abstrato que fornece a outro projeto diversas funcionalidades, reduzindo o esforço computacional e facilitando o trabalho do programador que irá usar o framework, assim ele não precisa entender a implementação, além de ter a sua disponibilidade uma gama de funcionalidades amplamente testadas.

3.5.1: Framework Radix

As empresas produtoras de softwares enfrentam como principais desafios a necessidade de garantir uma alta qualidade do software gerado, aumentar a produtividade dos desenvolvedores, diminuir os custos e prazos de desenvolvimento, sempre com o objetivo de garantir a satisfação e preservar a sua relação com seus clientes.

A solução encontrada pela Radix para que as necessidades sejam atendidas foi o desenvolvimento de um framework orientado a objetos. Trata-se de uma ferramenta que agrupa funcionalidades comuns a diversas aplicações possibilitando

a solução de uma família de problemas recorrentes em desenvolvimento de software. Os pilares do framework consistem em:

- Reutilização de código: possibilita o aumento da produtividade e, de maneira geral, a redução de prazos e cistos, visto que evita o gasto de tempo de trabalho implementando funcionalidades previamente desenvolvidas. Há também uma confiabilidade maior do código já que este foi usado, testado e aprimorado em diversos projetos passados (diminuição de bugs).
- *Design Patterns*: são técnicas consagradas de solução de problemas. Empregar um *design pattern* na confecção de um código implica adicionar um nível de abstração a ele de modo que detalhes particulares de implementação sejam isolados do resto do código. Assim, caso seja necessário, pode-se propagá-lo para todos os blocos que reutilizam este código. Verifica-se, portanto, que o emprego de *design patterns* possibilita uma melhor manutenção do código. Vale a pena ressaltar que muitas dessas funções vieram de projetos antigos, que nada tinham a ver com uma fábrica de Aços Longo. Estas melhorias feitas no Framework serão propagadas para projetos futuros.
- Arquitetura de Três Camadas: organização lógica do software em módulos independentes, mas que são conectados entre si. Tipicamente se usa um modelo de camadas onde são envolvidas: camada de apresentação (faz a interface com o usuário); camada de negócios (controle) onde são armazenadas as regras de negócios do processo produtivo do cliente; camada de dados (onde são armazenados os dados envolvidos nos processo). O objetivo de usar a arquitetura em três camadas é desacoplar a lógica de negócios dos dados e da apresentação, de maneira que alterações em cada uma destas duas últimas camadas se processem de maneira transparente às outras camadas, facilitando a manutenção do sistema.

Figura 10 - Estrutura de Camadas

3.5.2: Framework .NET

O .Net, ou *dotnet,* é uma plataforma para desenvolvimento de aplicações que foi criada pela Microsoft em 1999. Essa plataforma disponibiliza uma vasta biblioteca de funcionalidades e componentes, fornecendo interoperabilidade com uma gama de linguagens de alto nível.

A interoperabilidade do ambiente é semelhante à plataforma Java pois usa de uma máquina virtual independente da linguagem. A linguagem Java possui a JVM (*Java Virtual Machine*), e para .NET temos a máquina virtual chamada *Common Language Runtime* (CRL). Isso permite ao desenvolvedor não se preocupar com o dispositivo ao qual está destinada sua aplicação, o que, por sua vez, resulta em um aumento da eficiência. Atualmente a CLR permite rodar a mesma aplicação em vários dispositivos como smartphones, tabletes, Xbox, computadores desktops e servidores.

O framework .Net abstrai do desenvolvedor a necessidade de usar apenas uma única linguagem, já que hoje em dia têm-se mais de 30 opções destas, sendo algumas: C#, C++, Boo, F#, COBOL, Python, Pascal e Visual Basic.

As aplicações programadas em cima do framework .NET são duplamente compiladas. A primeira faz para uma linguagem intermediária, *Common Intermediate Language*, o que permite a integração entre projetos de diferentes linguagens. No final são compiladas para a linguagem máquina, conforme apresentado na Figura 1.

Dentre as vantagens do framework .Net destacam-se:

- Interoperabilidade: possibilidade de executar funcionalidades a partir de qualquer linguagem disponibilizada;
- Segurança: trata diversas vulnerabilidades comuns e fornece um modelo de segurança integrado para as aplicações.
- Biblioteca de funcionalidades comuns: existem várias funcionalidades disponíveis independentes da linguagem que esteja sendo utilizada, desde manipulação de arquivos, acesso a base de dados, criptografia, fórmulas matemáticas, processamento gráfico até gerenciamento paralelo de aplicações.

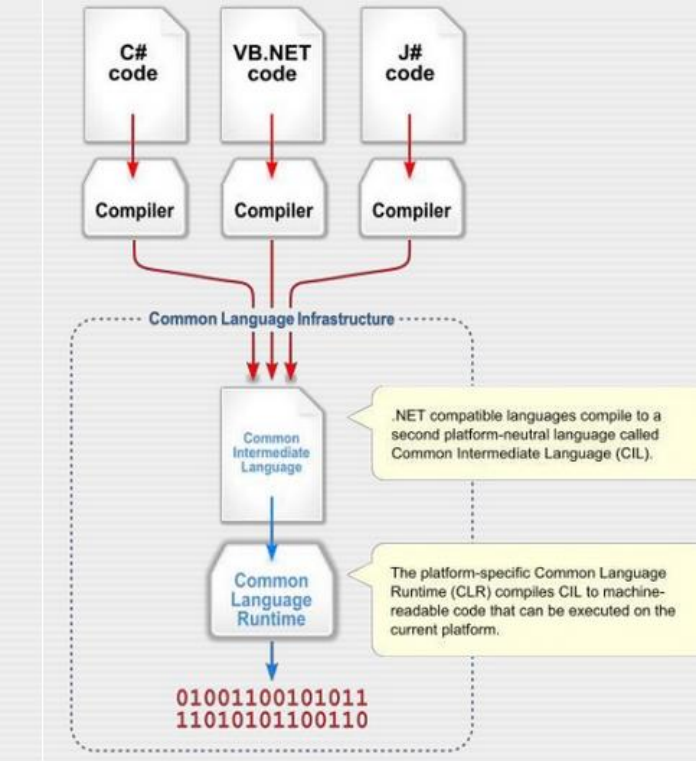

Figura 11 - Compilação framework .NET

3.5.3: Arquitetura MVC

A arquitetura padrão MVC fornece uma maneira de dividir a funcionalidade envolvida na manutenção e apresentação dos dados de uma aplicação. Na arquitetura MVC o modelo representa os dados da aplicação e as regras de negócio

que governam o acesso e a modificação dos dados. O modelo mantém o estado persistente do negócio e fornece ao controlador a capacidade de acessar as funcionalidades da aplicação encapsuladas pelo próprio modelo.

Um controlador define o comportamento da aplicação, é ele que interpreta as ações do usuário e as mapeia para chamadas do modelo. Em um cliente de aplicações web essas ações de usuário poderiam ser cliques de botões ou seleções de menus. As ações realizadas incluem ativar processos de negócio ou alterar o estado do modelo.

Com base na ação do usuário e no resultado do processamento do modelo, o controlador seleciona uma visualização a ser exibida como parte da resposta à solicitação do usuário. Há normalmente um controlador para cada conjunto de funcionalidades, e no sistema MES existe um controlador para cada tela da aplicação.

3.5.4: ASP .NET MVC

O ASP.NET MVC é uma implementação da arquitetura MVC para o framework .NET com o objetivo de criar aplicações Web no padrão MVC. O framework ASP.NET MVC fornece um ambiente de qualidade e leve que está integrado com os recursos do .NET. É uma evolução do framework que agrega novas funcionalidades para programar um projeto na arquitetura Web.

As vantagens de se usar este framework são:

- Pode-se fazer a divisão da aplicação em camadas, o que facilita o entendimento e a manutenção das aplicações.
- Um desenvolvimento *front-end* (interface) mais simples, fácil de integrar no projeto com os designers gráfico. As páginas (*View*) possuem apenas o HTML e a marcação dos valores dinâmicos (mas não tem lógicas de aplicação embutidas).
- Possuir um controle sobre a interface, pois isso garante controle total sobre o HTML gerado e os arquivos de CSS.

- Fácil integração com bibliotecas *JavaScript* e *jQuery*.
- Escalabilidade.

3.5.4.1: Fluxo Execução da Aplicação ASP.NET MVC

As requisições que vem do usuário, como por exemplo acessar uma página pelo browser (*URL*), são enviadas para o servidor. No servidor o framework do ASP .NET MVC faz o roteamento da requisição para o controlador correto. O controlador invoca a camada de modelo e no final *renderiza* o html da tela, então retorna para o browser e atualiza a tela do usuário.

Simbologia CSS - URL

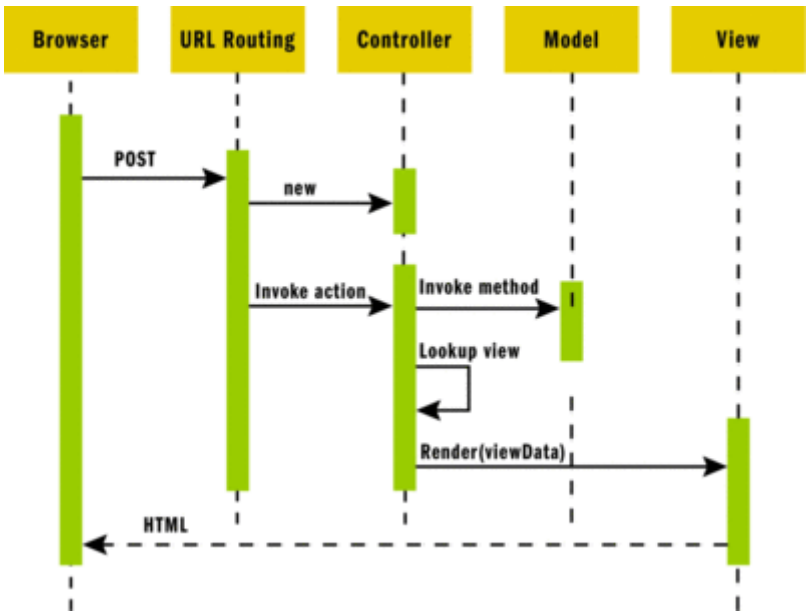

Figura 12 - Funcionamento Arquitetura ASP .NET MVC

3.5.4.2: Razor

ASP.NET *Razor* é uma *view engine*, ou seja, é uma ferramenta para escrever visualizações (páginas) em aplicações web. Com ela é possível integrar os dados da aplicação com as páginas em *Html* e montar no servidor antes de enviar ao usuário.

A característica funcional mais importante do *Razor* é a de se usar um código limpo e legível, onde é possível construir páginas extremamente elaboradas (ricas em conteúdo com código organizado e limpo) gerando assim facilidade no processo de manutenção.

As imagens a seguir demonstram um pouco o poder da ferramenta ao criar uma tabela com uma lista de entidades.

```
@for (var i = 0; i < listaSistemaOperacional.Count(); i++)
{
    <tr class="@(i % 2 == 1 ? "odd" : "even")">
        <td>@listaSistemaOperacional[i].Nome</td>
        <td>@listaSistemaOperacional[i].Cadeia.Nome</td>
        <td>@listaSistemaOperacional[i].CodigoN2</td>
        <td>@listaSistemaOperacional[i].CodigoSAP</td>
        <td align="center" width="18px"><span style="cursor:po
        <td align="center" width="18px"><span style="cursor:po
    </tr>
}
```

Figura 13 - Parte de um arquivo Razor para preencher as linhas de uma tabela

```
▼<tr class="even">
   <td>Aciaria</td>
   <td>Aciaria</td>
   <td></td>
   <td>ACIA000001</td>
  ▶<td align="center" width="18px">…</td>
  ▶<td align="center" width="18px">…</td>
  </tr>
▼<tr class="odd">
   <td>Dobradeira</td>
   <td>Acabamento</td>
   <td></td>
   <td></td>
  ▶<td align="center" width="18px">…</td>
  ▶<td align="center" width="18px">…</td>
  </tr>
▼<tr class="even">
   <td>Endireitadeira</td>
   <td>Acabamento</td>
   <td></td>
   <td></td>
  ▶<td align="center" width="18px">…</td>
  ▶<td align="center" width="18px">…</td>
  </tr>
```

Figura 14 - Figura 14 arquivo html gerado pelo Razor enviado ao browser

3.6: Filosofias de Desenvolvimento:

Conforme descrito por *[3]* V.B. Mazzola e J-M Farines, “Metodologias de Concepção de Software e de Sistemas”, nos anos 40, quando se iniciou a evolução dos sistemas computadorizados, grande parte dos esforços, e consequentes custos, era concentrada no desenvolvimento do hardware, em razão, principalmente das limitações e dificuldades encontradas na época. À medida que a tecnologia de hardware foi sendo dominada, as preocupações se voltaram, no início dos anos 50, para o desenvolvimento dos sistemas operacionais, onde surgiram então as primeiras realizações destes sistemas, assim como das chamadas linguagens de programação de alto nível (como FORTRAN, COBOL, e seus respectivos compiladores). A tendência da época foi de poupar cada vez mais o usuário de um computador de conhecer profundamente as questões relacionadas ao funcionamento interno da máquina, permitindo que aquele pudesse concentrar seus esforços na resolução dos problemas computacionais em lugar de preocupar-se com os problemas relacionados ao funcionamento do hardware. Uma consequência deste crescimento foi a necessidade, cada vez maior, de desenvolver grandes sistemas de software em substituição aos pequenos programas aplicativos utilizados até então. Desta necessidade surgiu um problema nada trivial devido à falta de experiência e a não adequação dos métodos de desenvolvimento existentes para pequenos programas, o que foi caracterizado, ainda na década de 60, como a "crise do software", mas que, por outro lado, permitiu o nascimento do termo "Engenharia de Software".

Atualmente, apesar da constante queda dos preços dos equipamentos, o custo de desenvolvimento de software não obedece a esta mesma tendência. Pelo contrário, corresponde a uma percentagem cada vez maior no custo global de um sistema informatizado. A principal razão para isto é que a tecnologia de desenvolvimento de software implica, ainda, em grande carga de trabalho, como também os projetos de grandes sistemas de software envolvendo, em regra geral, um grande número de pessoas num prazo relativamente longo de desenvolvimento. O desenvolvimento destes sistemas é realizado, na maior parte das vezes, de forma

"ad-hoc", conduzindo a frequentes desrespeitos de cronogramas e acréscimos de custos de desenvolvimento.

Na busca por reduzir os custos e prazos de construção dos softwares, as empresas de desenvolvimento tem se utilizado de metodologias pré-definidas para sua construção. A seguir são explicados os modelos mais comumente utilizados e também como funciona o modelo utilizado pela Radix. Logo após é feita uma pequena comparação entre os modelos tradicionais de desenvolvimento.

3.6.1: Modelo Cascata

Este modelo foi o primeiro a ser criado e é também o mais simples de desenvolvimento de software, estabelecendo uma ordenação linear na realização das tarefas. Como mostra a Figura 15, o ponto de partida se dá na Engenharia de Sistemas, onde o objetivo é ter uma visão global do sistema como um todo (hardware, software e usuários) como forma de definir o papel do software. Em seguida vem a etapa de Análise de Requisitos que vai permitir uma clara definição dos requisitos de software, sendo que o resultado de um passo é usado como referência para as etapas posteriores (Projeto, Codificação, Teste e Manutenção).

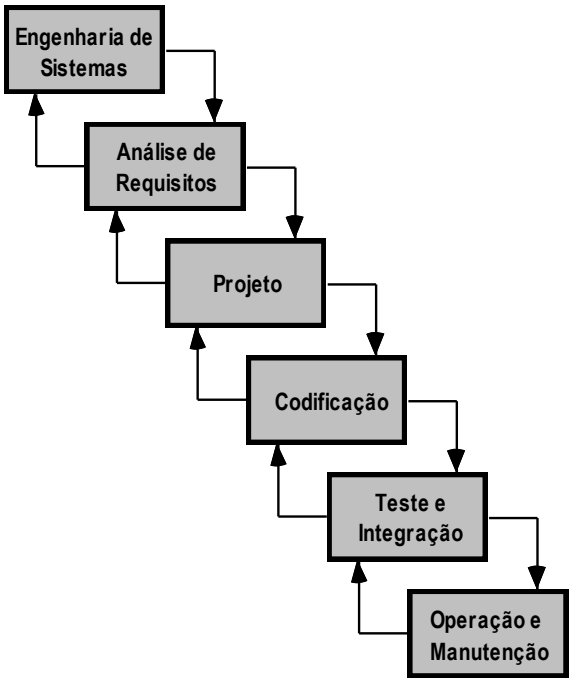

Figura 15 - Modelo Cascata

O modelo cascata define em uma ordem linear as etapas de desenvolvimento. No fim de cada etapa, um mecanismo de certificação é implementado, validando se o que está saindo de cada etapa é relacionado com a saída da anterior.

3.6.2: Modelo Prototipação

A grande inovação deste modelo é eliminar a política de “gongelamento” dos requisitos antes do projeto do sistema. Ele foi feito através da confecção de um protótipo com base no conhecimento dos requisitos iniciais para o sistema. O desenvolvimento do protótipo é feito obedecendo as diferentes etapas mencionadas anteriormente só que de maneira mais rápida e sem muita formalidade.

O protótipo pode ser oferecido ao cliente em duas diferentes formas: protótipo em papel ou modelo executável em PC retratando a interface gráfica, capacitando ao cliente compreender como ele irá interagir com o software. Colocado à disposição do cliente, o protótipo vai ajudá-lo a melhor compreender o que será o sistema de desenvolvimento. Através da manipulação do protótipo é possível validar ou reformular os requisitos para as etapas seguintes.

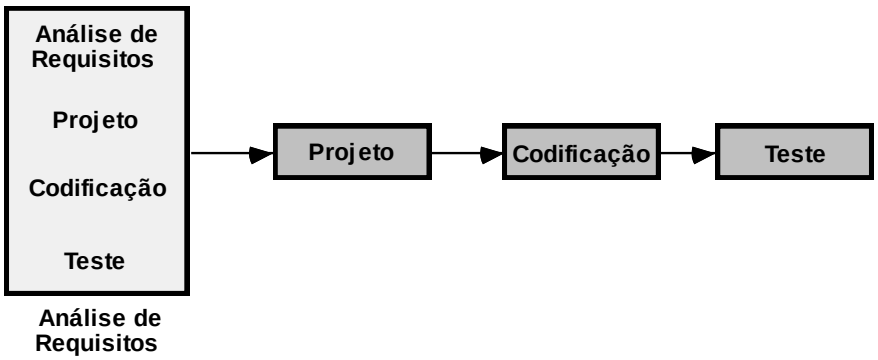

Figura 16 - Modelo Protótipo

Os protótipos não são sistemas completos e deixam normalmente a desejar. A experiência adquirida durante a construção do protótipo ajuda nas etapas

posteriores do desenvolvimento real, permitindo manter o que foi bem concebido e repensar o que ficou trabalhoso ou mal projetado.

3.6.3: Modelo Iterativo

Este terceiro modelo foi concebido para combinar as vantagens dos modelos anteriormente citados. O funcionamento do modelo iterativo, ilustrado na **Error! Reference source not found.**, é o de que um sistema deve ser desenvolvido de forma incremental sendo que cada incremento vai adicionando ao sistema novas funcionalidades até a obtenção do sistema final, sendo que a cada passo modificações podem ser introduzidas.

Uma vantagem é a facilidade de realizar os testes do sistema, uma vez que fazer isso em cada nível é mais fácil do que realizar todos os testes de um sistema inteiro depois de um grande tempo de desenvolvimento.

Cada iteração do modelo de Desenvolvimento Iterativo consiste em avançar um passo na direção do projeto completo, através da realização das três etapas: projeto, codificação e análise. O ciclo se repete até convergir para o software final.

Pode-se alterar componentes anteriores em razão de erros ou problemas detectados em alguma análise.

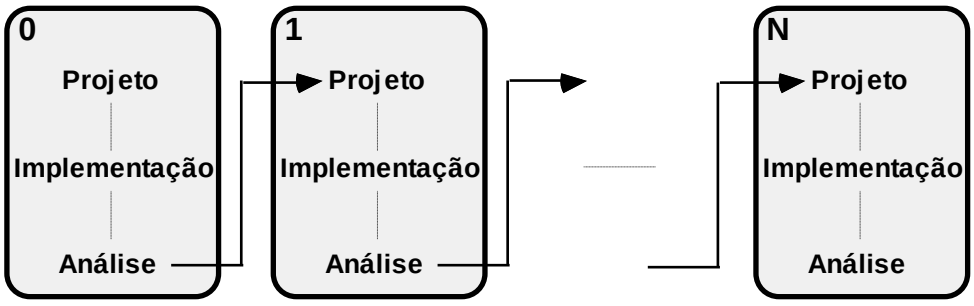

Figura 17 - Modelo Desenvolvimento Iterativo

3.6.4: Desenvolvimento Ágil de Software – Radix

Desenvolvimento ágil (também conhecido como método ágil) é um conjunto de metodologias de desenvolvimento de software. Tal como as outras metodologias, providencia uma estrutura conceitual para reger projetos de engenharia de software.

A maioria dos métodos ágeis tenta minimizar o risco pelo desenvolvimento do software em curtos períodos chamados de “iteração”. Cada iteração é um projeto de software dentro de si mesmo e inclui todas as tarefas: análise de requisitos, projeto, codificação, teste e documentação. Enquanto em um processo convencional cada iteração não necessariamente adiciona um novo conjunto significativo de funcionalidades, um projeto de software ágil busca a capacidade de implantar uma nova versão do software ao fim de cada iteração.

Métodos ágeis tendem a enfatizar mais a comunicação em tempo real, preferencialmente face-a-face, no lugar de documentos escritos. A maioria dos componentes de um grupo ágil deve estar agrupada em uma sala. Isso inclui todas as pessoas necessárias para o desenvolvimento de software: programadores, clientes (que definem os requisitos), testadores e gerentes.

Como descrito em [4] B. Boehm. Balancing “Agility and Discipline: A Guide for the Perplexed”. os princípios que o desenvolvimento ágil valoriza são:

- Garantir a satisfação do cliente ao entregar rapidamente e continuamente softwares funcionais;
- Softwares funcionais entregues frequentemente (semanas);
- Softwares funcionais são a principal medida de progresso;
- Cooperação constante entre pessoas que entendem do ‘negócio’ e desenvolvedores;
- Design do software que zela pela excelência técnica;
- Simplicidade;
- Rápida adaptação às mudanças;
- Preferência: pelo software funcional à documentação extensa, pela colaboração com clientes à negociação de contratos e por geração de mudanças ao seguir um plano.

Os métodos ágeis surgiram como uma reação aos métodos antigos “pesados” (que tinham regulamentações rígidas). O processo originou-se da visão que o modelo em cascata era burocrático, lento e não condizia com a forma usual que os programadores trabalham.

Hoje em dia, empresas que prestam serviço de tecnologia quase que em sua totalidade adotam a maioria dos princípios de desenvolvimento ágeis. Na Radix não é diferente. Apesar de nesse projeto não seguir nenhum método específico de desenvolvimento ágil (como o Scrum, programação extrema, etc) seguem-se seus princípios, fazendo uso de entregas frequentes de softwares funcionais, constante conversa e validação do que se está sendo feito, pouca documentação, maior foco no software e às reações entre diversas trocas de código.

3.6.4.1: Comparação com o Desenvolvimento Iterativo

Os métodos ágeis compartilham a ênfase no desenvolvimento iterativo e incremental para construções de versões funcionais do software em curtos períodos de tempo. Os métodos ágeis diferem do método iterativo porque seus períodos de tempo são medidos em semanas, ao invés de meses, e a realização é efetuada de uma maneira altamente colaborativa e com menos documentos formais escritos.

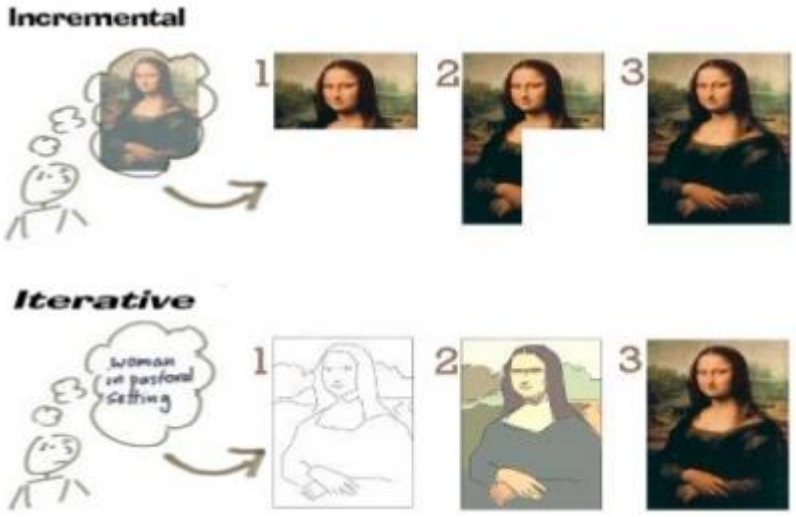

Figura 18 Método Incremental x Iterativo

3.6.5: *Clean Code*, Código Limpo:

Nesta época de desenvolvimentos ágeis de software, o foco está em colocar o produto rapidamente no mercado, ou, no caso da Radix, entregar o produto mais

rápido possível para o cliente. Deseja-se que a indústria funcione em velocidade máxima na produção de software, e, por causa disso, muitas metodologias de desenvolvimento ágil tentam transformar o desenvolvimento em uma linha de montagem.

De acordo com a filosofia de código Limpo de Robert C. Martin *[5]* R. C. Martin – “Código Limpo, Habilidades Práticas do Agile Software”., assim como na indústria automobilística, a maior parte do trabalho não está na fabricação, mas na manutenção e prevenção. Em software, 80% ou mais do que se faz é chamado de “manutenção”: o ato de reparar. Em vez de focar-se na produção, dever-se-ia pensar mais como um pedreiro ou mecânico, o qual irá reparar o produto e mantê-lo funcionando. Na área de manufatura foi criada uma abordagem japonesa chamada Manutenção Produtiva Total utilizando dos chamados cinco princípios (5S). A filosofia de Código Limpo tenta trazer esses princípios de organização e disciplina da manufatura para a engenharia de software.

3.6.5.1: Princípios do Código Limpo:

Os princípios do código limpo são os mesmos que da filosofia 5S adaptados para a engenharia de software. Eles são listados a seguir:

- *Seiri*, ou “Organização”: saber onde estão as coisas e usar abordagens com nomes adequados é crucial.
- *Seiso,* ou “Arrumação”: há um antigo ditado americano que diz: “um lugar para tudo e tudo em seu lugar”. Um pedaço de código deve estar onde você o espera encontrar.
- *Seiso,* ou “Limpeza”: manter o local de trabalho limpo. Não encher o código de comentários que informam o passado ou os desejos para o futuro.
- *Seiketsu,* ou “Padronização”: a equipe toda tem que se esforçar para manter o código limpo.
- *Shutsuke,* ou “Disciplina”: para manter o código limpo é necessário manter a disciplina, seguir as práticas e repetir frequentemente isso no trabalho.

3.6.5.2: Por que usar a técnica de Código Limpo:

Na área de programação um problema muito comum enfrentado por todos é o de ao ter que mexer no código confuso de outra pessoa, ou no seu próprio código antigo. Conforme o projeto evolui, as equipes que trabalhavam rapidamente no início começam a perceber que o mesmo está indo a passos de tartaruga. Cada pequena alteração feita no código causa uma falha em duas ou três partes do mesmo código. Mudança não é trivial, cada adição ou modificação exige que restaurações e remendos sejam entendidos para poder incluir outra. Com o tempo a bagunça se acumula e a produtividade da equipe diminui assintoticamente aproximando-se de zero.

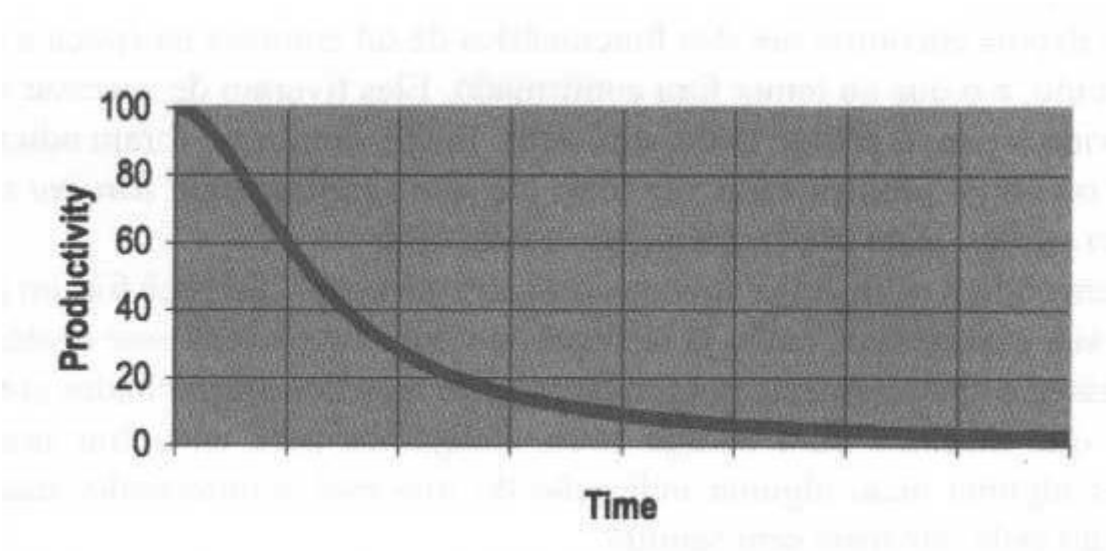

Figura 19 - Produtividade vs Tempo

3.6.5.3: O que é um Código Limpo:

Ao programar, passa-se muito mais tempo lendo código o já existente com o objetivo de saber se uma mudança feita não irá afetar nenhuma outra parte do projeto, ou se não está replicando as linhas do código em si. Então, para melhorar a habilidade como programadores, deve-se deixar o código de fácil leitura.

A principal técnica usada para melhorar a qualidade do código é a escolha de bons nomes para as classes, funções e variáveis, nomes representativos que eliminem a necessidade do uso de comentários.

Outras boas práticas de programação incluem: deixar as funções as menores possíveis e não deixar uma função que o leitor tenha que *scrollar* a tela para poder lê-la inteira.

O uso adequado de comentários serve para compensar o fracasso em expressar-se no código, portanto evita-se ao máximo usá-los. Deve-se sempre tentar manifestar-se pelo código, nomeando variáveis e funções.

Na figura a seguir tem-se um pedaço de código do projeto. Como se pode observar, não foi necessário nenhum comentário no código para demonstrar o que ocorre em sua execução. Esta é uma função pequena, não tendo mais que 15 linhas. Todos os nomes de função e variáveis têm um significado. Apesar de sempre ser possível realizar uma melhoria na legibilidade do código, a função atende todos os requisitos de um código limpo e pode ser entendida sem muito trabalho.

```
public virtual void InserePerdaDeProdutoFeixeBobina(int codigoFeixeBobina, double PesoPerda, MotivoPerda MotivoPerda)
{
    MotivoPerda = BusinessFactory.GetInstance().Get<MotivoPerdaBusinessFacade>().GetById(MotivoPerda.Codigo);
    var MotivoPerdaEquipamentoProducao = BuscarMotivoPerdaDaLaminacao(MotivoPerda);

    TratamentoConcorrenciaErro(MotivoPerda, MotivoPerdaEquipamentoProducao, EnumTipoPerda.PerdaProduto);

    Perda perda = ((IPerdaDataAccess)crudDataAccess).BuscaPerdaFeixeBobinaLaminacaoPorCodigoFeixeBobinaLaminacao(codigoFeixeBobina);

    if (ExistePerdaProdutoNoFeixeBobina(perda))
    {
        perda.PesoPerda = PesoPerda;
        perda.MotivoPerdaEquipamentoProducao = MotivoPerdaEquipamentoProducao;
        Edit(perda);
    }
    else
    {
        InsereNovaPerdaFeixeBobinaLaminacao(codigoFeixeBobina, PesoPerda, MotivoPerdaEquipamentoProducao);
    }
}
```

Figura 20 - Exemplo de um código limpo do projeto

Capítulo 4: Estrutura do Projeto

Nesse capítulo será descrito como o projeto surgiu, sua estrutura e suas fases, dando ênfase às partes onde tive intensa participação. O MES aços longos é um projeto consideravelmente extenso que foi produzido por diversas pessoas, então se fez necessária uma distinção entre as partes que mais interessava a este trabalho.

4.1: Sistemas de Informação necessários:

Para controlar a produção da fábrica de aços longos, a arquitetura de TI foi dividida em três subsistemas, os quais recolhem informações do processo de produção e as comunicam entre si (cada um no seu nível de abstração).

A camada de nível mais elevado, conhecida como *Enterprise resource planning* (ERP), é responsável por controlar as vendas e a quantidade de cada material que vai ser produzida num determinado período de tempo, como também os custos gerais de produção. Já a camada de nível dois faz o papel de controlar os atuadores do processo e medir as variáveis de processo. A camada de nível intermediário é a mais importante porque realiza muitas funções: desempenha o meio campo da comunicação entre as duas outras camadas, é responsável por registrar os consumos do processo e da produção, controla o estoque e o fluxo da qualidade, estima a “vida” dos equipamentos da produção, programa a produção da fábrica para que atinja os objetivos gerados pelo ERP, contabiliza as paradas do processo, e disponibiliza diversas informações e histórico do processo para a tomada de decisão.

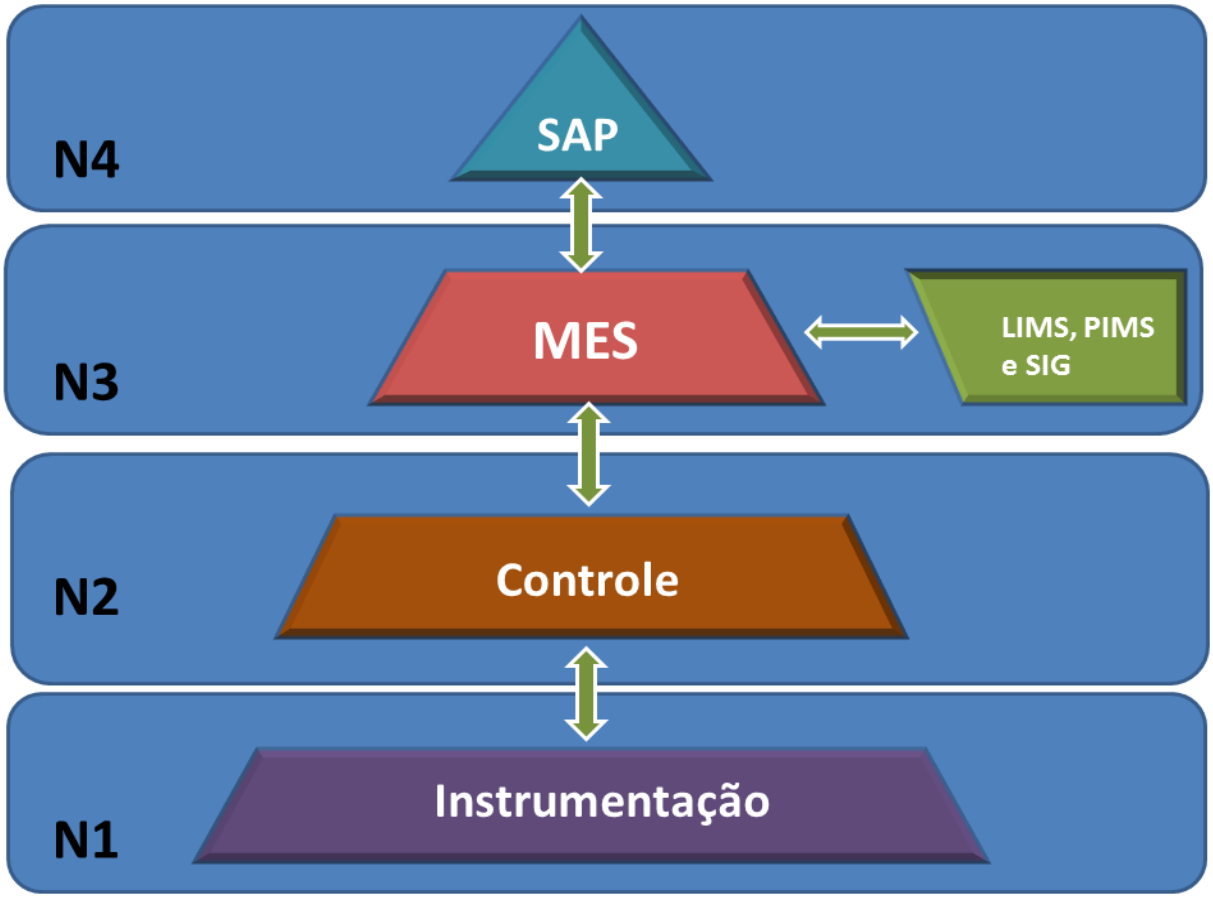

Figura 21 Arquitetura de níveis do sistema

4.2: Projeto MES

O projeto do MES como um todo começou em meados de outubro de 2012 com a parte de negociação, da área comercial e a parte de planejamento do projeto. Como vimos nos métodos de desenvolvimento ágil, a construção do software e a especificação dos requisitos é feita em paralelo com o desenvolvimento conforme pode-se observar o no cronograma do projeto.

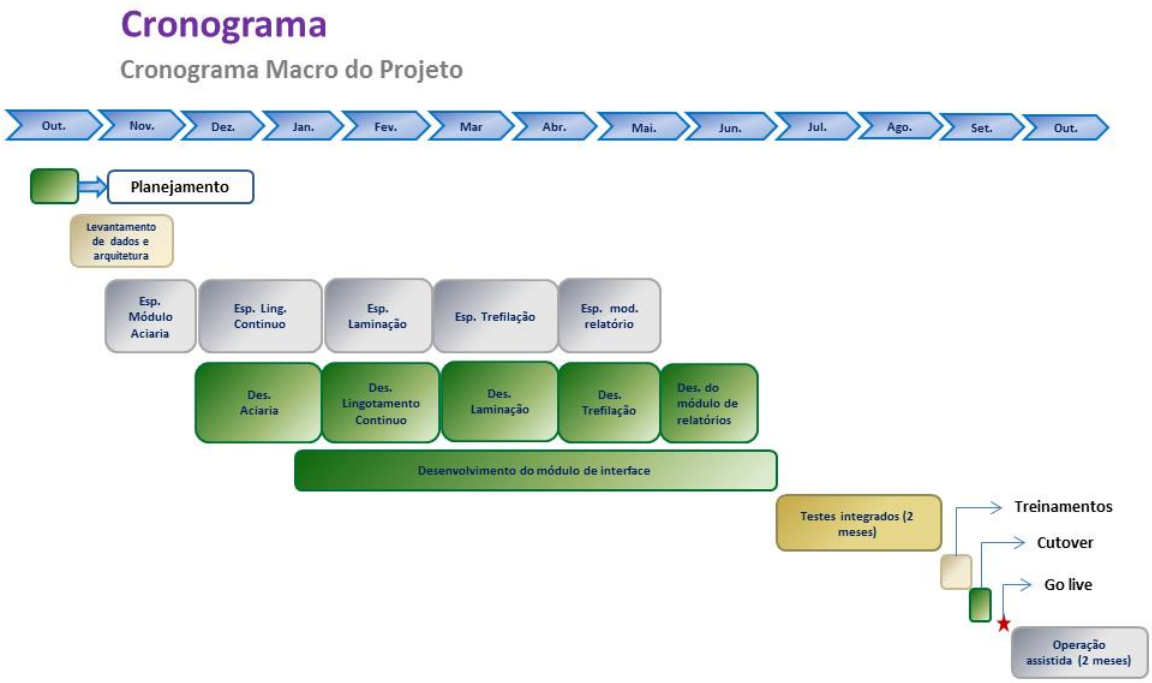

Figura 22 - Cronograma geral do Projeto

4.2.1: Análise de Requisitos

A análise de requisitos é um aspecto importante no projeto porque é responsável por coletar dados indispensáveis e necessários, já que são exigências visadas pelo usuário para solucionar um problema e alcançar seus objetivos (assim como determinar as expectativas de um usuário para determinado produto).

A análise de requisitos é vital para o desenvolvimento do sistema, pois o documento de especificação será a referência para validar o produto final, e isso é importante porque estabelece uma ponte entre os pensamentos abstratos de como o cliente imagina o software e como o software realmente fará.

A análise consiste nos 5 princípios, conforme descrito por *[1]* FILGUEIRAS, Dra. Lúcia V. L.; MELNIKOFF, Dra. Selma Shin Shimizu, “Engenharia de Software”:

- Reconhecer o problema: entender as necessidades do cliente e o que ele deseja.
- Avaliar o problema e a síntese da solução: compreender o problema, identificar as informações que serão necessárias ao usuário e propor a melhor solução possível.

- Modelar um recurso para o suporte da síntese da solução.
- Especificar: consolida funções, interfaces, desempenho e as regras de negócio do sistema.
- Revisar: juntos, cliente e Radix, avaliarão o objetivo do projeto visando eliminar inconsistências e omissões do sistema.

Como o projeto é feito usando os métodos ágeis descritos na seção 3.6.4:, é produzido um documento de requisito para cada iteração do projeto. Na abordagem usada pela empresa, nossa equipe de requisitos tem um contato diário com um cliente, estando alocado no escritório deles. Isso permite uma fácil validação das funcionalidades do software e readequação dos requisitos, caso alguma demanda seja alterada, com o menor retrabalho possível.

4.2.1.1: Tipos de Requisitos:

Dentro da especificação do software existem diferentes tipos de requisitos, os funcionais e os não funcionais:

- Funcionais: estabelece como o sistema vai agir e o que deve fazer as funcionalidades e serviços.
- Não funcionais: definem as propriedades do sistema e suas restrições, confiabilidade, tempo de resposta, espaço em disco, segurança.

4.2.2: Desenvolvimento

A equipe de desenvolvimento é responsável por criar o código da aplicação, especificando os requisitos detalhadamente de modo que uma máquina possa executá-las. Tendo em mãos o documento de especificação, é trabalho da equipe de desenvolvimento montar a estrutura do banco de dados e a arquitetura do

projeto, programarem as funcionalidades do software e a montar a interface com os usuários (seja ele um operador humano ou outro sistema).

O trabalho de programação foi dividido entre os desenvolvedores de modo que cada um ficou responsável por programar diferentes funcionalidades do sistema. No caso do MES, cada programador ficou responsável por desenvolver um conjunto de telas separadamente. Também era papel deles programar todas as funcionalidades para a tela funcionar corretamente, desde o *front-end* (página web, JavaScript) até o controlador que processaria as diferentes requisições do usuário, a camada de negócios e a persistência. Neste trabalho será dada uma ênfase no módulo de produção, onde se encontram diferentes telas de produção.

Como não estamos produzindo um software próprio para a empresa e sim para um cliente, ficamos amarrados. Todas as tecnologias utilizadas são especificadas antes de começar o trabalho de programação. Os itens listados abaixo se referem às tecnologias e requisitos necessários para suportar as camadas de apresentação, negócios e persistência:

- ***Internet Information Service* (IIS) 7.5:** IIS é o servidor de aplicativos da Web e Microsoft e deve ser instalado e configurado no servidor do sistema;
- ***Banco de Dados SQL Server 2012***: Sistema Gerenciador de Banco de Dados;
- ***Internet Explorer* (IE):** navegador web para acessar as funcionalidades do sistema. O sistema será desenvolvido para funcionar na versão do IE 8.0;
- ***Framework* .NET 4.0:** Framework Microsoft para suportar a aplicação;
- **ASP.NET**: Linguagem web que as páginas da Web são desenvolvidas;
- **C#**: linguagem em que o código é escrito no servidor;

Bibliotecas utilizadas pela aplicação:

- **ASP.NET MVC 3.0**: Framework. NET para desenvolvimento web que implementa o padrão MVC;
- **MSEL - Unity Application Block**: Biblioteca utilizada para realização de injeção de dependência e interceptação de tipos.

- **MSEL - DataAccessApplication Block:** Incorpora funcionalidades padrões de banco de dados nas aplicações (como acesso a dados) e retornam dados em uma variedade de formatos, o que simplifica a codificação e conexão com banco de dados.
- **Log4Net:** biblioteca para geração de logs de informação e erros do sistema;
- **NPOI**: Biblioteca para geração de relatórios no formato Excel.
- **NUnit**: Framework para realização de testes unitários.

Mecanismos de apresentação:

- **Razor**: Otimizador de sintaxe para HTML que utiliza uma abordagem focada na codificação de templates. Faz parte do Microsoft MVC 3.0.
- **JQuery**: Coleção de ferramentas Javascript para manipulação e exibição de dados do cliente.
- **Knockout JS**: Biblioteca Javascript que trabalha em conjunto com JQuery para implementação de padrão MVVM, ajudando a criar uma visualização rica de dados com um modelo limpo e eficiente de codificação.

4.2.2.1: Sistema de controle de versão

Um sistema de controle de versão (SVN), ou versionamento, é um software com a finalidade de gerenciar diferentes versões no desenvolvimento de um software, para controlar as diferentes versões e histórico do desenvolvimento de código fonte e da documentação. Presente em várias empresas que trabalham em desenvolvimento, é também utilizado para o desenvolvimento de software livre.

A eficácia do controle de versão do software é comprovada por ser uma exigências do desenvolvimento de certificações como a CMMI (*Capability Maturity Model Integration*). As principais vantagens de se utilizar um sistema de controle são:

- Controle do histórico: facilidade em desfazer alterações e analisar o histórico do desenvolvimento, como também facilidade no resgate de versões antigas e estáveis.

- Trabalho em equipe: um sistema de controle de versão permite que diversas pessoas trabalhem sobre o mesmo conjunto de documentos e desenvolvam a mesma aplicação.
- Ramificação do projeto em linhas de desenvolvimento.
- Marcação e resgaste de versões estáveis.

O software usado para fazer o controle de versão do projeto é o *TortoiseSVN*. De tempos em tempos, conforme dita a filosofia de métodos ágeis, sobe-se uma versão para o cliente. O *TortoiseSVN* salva uma cópia da versão estável, como uma fotografia do código no dia, chamado de *Tag*.

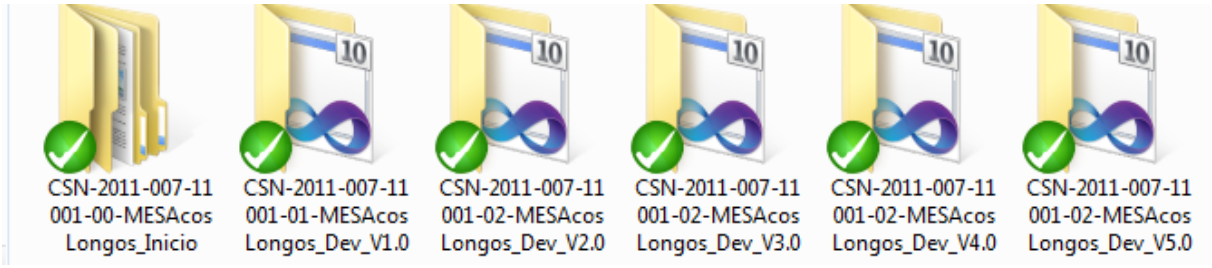

Figura 23 - Versões Entregues

4.2.2.2: Arquitetura Utilizada

O MES do projeto possuirá uma arquitetura Web Componetizada (.NET) divida em três camadas: persistência, negócios e apresentação. Esta arquitetura segue o padrão MVC e facilita a separação entre as funcionalidades da tela e dos serviços, resultando no maior desacoplamento entre as camadas de negócio e apresentação, o que torna o software organizado, fácil de manter e atualizar.

A Figura 24 mostra como o padrão MVC do *framework* .NET se encaixa na arquitetura de 3 camadas do *framework* da Radix.

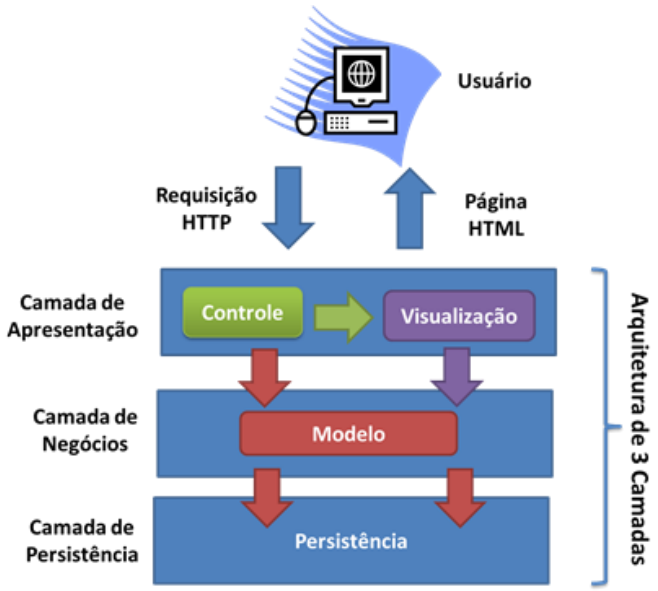

Figura 24 - MVC e arquitetura 3 camadas

4.2.2.3: Camada de Apresentação

A camada de apresentação é responsável por: fazer a lógica de construção das páginas para serem exibidas pelos usuários, tratar os eventos do browser (como cliques) e gerenciar o fluxo de execução do MES. Para isso, a camada de apresentação conta com as partes de Controle e Visualização do padrão MVC.

A parte de lógica da camada de apresentação será divida entre o cliente (browser do usuário) e o servidor. No cliente será utilizado JavaScript para validação de dados e Razor para construção dinâmica da página, assim como AJAX para realizar requisições de partes da página ao servidor sem que seja preciso recarregar a página inteira. Esta funcionalidade permitirá um melhor desempenho e usabilidade da aplicação.

4.2.2.4: Camada de Negócios

A camada de negócios é responsável por: implementar a lógica do domínio da aplicação, expor esta lógica para a camada de apresentação por meio de uma

interface bem definida e obter as informações necessárias acessando as diferentes fontes de dados disponibilizadas através das camadas de Persistência e Serviços.

Todos os cálculos necessários para controlar e garantir a qualidade do fluxo de produção serão desenvolvidos na camada de Negócios.

4.2.2.5: Camada de Persistência

A camada de Persistência é responsável pela lógica de acesso ao banco de dados e pelo mapeamento do modelo relacional. Essa camada recebe as requisições da camada de negócios, retornando informações ou persistindo informações no banco de dados.

4.2.2.6: Camada de Dados

A camada de dados é responsável pelo armazenamento físico dos dados representativos das entidades do sistema. É representada pelo banco de dados escolhido: *SQLServer*.

4.2.2.7: Camada de Serviços

A Camada de Serviços é composta por serviços que acessam ou fornecem informações para os sistemas de terceiros, como o SAP e XQI (Sistemas Nível 2, LIMS, SIG e WINOE). Essa camada interage com a Camada de Negócios para buscar ou armazenar informação no MES. Cada serviço irá fazer a conversão de dados necessária de/para a estrutura MES ao enviar ou recuperar dados de sistemas terceiros. Todos os serviços serão implementados com o uso de WebServices para realizar consultas a sistemas de terceiros através de SOA, ou

prover interfaces que podem ser usados por outros aplicativos para acessar dados do MES.

4.2.2.8: Modularização

O Desenvolvimento de sistemas sem uma metodologia geralmente resulta em um software com vários erros e com alto custo de desenvolvimento que, consequentemente, exige um custo elevado para sua manutenção futura. A modularização de programas, juntamente com outras técnicas de programação, integra o ferramental para a elaboração de softwares altamente complexo, sem perder aspectos fundamentais como confiabilidade, legibilidade, fácil manutenção e flexibilidade.

A solução MES deste projeto contempla o desenvolvimento/manutenção dos módulos apresentados nas próximas seções:

- Módulo Central: é responsável por organizar a solução MES, além de concentrar as funcionalidades gerais da aplicação.
- Módulo de Produção: este módulo foca na gestão de capacidades de produção; também mantém o registro da posição das bobinas na planta de produção.
- Módulo de Programação: este módulo cobre as funcionalidades de programação da produção.
- Módulo de Qualidade: neste módulo tem as funcionalidades relativas ao controle de qualidade.
- Módulo de Campanha: este módulo é responsável pelas funções de controle de manutenção.

4.2.3: Homologação

A atividade de desenvolvimento de software torna-se cada vez mais complexa à medida que usuários sentem a necessidade de interagir com sistemas para realização de diversas tarefas, da forma mais automatizada, confortável e funcional possível. A fim de atender essas necessidades dos usuários, tornou-se essencial orientar o desenvolvimento do software com foco na qualidade do produto.

Porém, para garantir a qualidade de um produto de software, é necessário identificar as exigências implícitas e explícitas para, posteriormente, avalia-lo. Infelizmente, o processo de identificação de exigências ainda é bastante complexo em um software em fase de concepção.

Enquanto desenvolve-se um software, realizam-se diversos testes, para saber se o que foi feito está correto. Alguns testes unitários (como se subentende, uma função apenas) e outros testes integrados (onde se liga a aplicação e roda-se passo a passo como se fosse um usuário testando tudo, aplicação, interface gráfica e banco de dados). Entretanto, os testes de desenvolvedor geralmente estão “viciados”, além de apenas testar pequenas funcionalidades.

Para garantir a qualidade do software, enquanto desenvolve-o, tem-se uma equipe especializada em homologação responsável por fazer um meio campo entre o desenvolvimento e a especificação. Se houve garantia de que o que foi especificado é o que está sendo desenvolvido (ao mesmo tempo em que se procura os “bugs” nos softwares e as inconsistências no que foi especificado), então o retorno é positivo.

4.2.3.1: Trac Integraded SCM & Project Management:

Usando da filosofia de desenvolvimentos ágeis, tem-se a equipe de homologação que trabalha concomitantemente à equipe de desenvolvimento. As duas equipes também tem conversas face-a-face todos os dias, com o objetivo de sanar dúvidas e trocar ideias. Porém, faz-se necessário utilizar algum método de

controle de erros e correções. No projeto usa-se do software *trac*, uma abordagem web que qualquer pessoa pode acessar pelo navegador porque é programa integrado com interface para o controle de *Subversion*.

No *trac* o homologador abre *tickets* para a equipe de desenvolvimento como se fosse um pequeno *post-it* para avisar e ficar gravado o que os desenvolvedores precisam fazer. O software tem uma gama de possibilidades para categorizar, separando os tickets por versão, prioridades, tipos (defeito, sugestão de melhoria, alteração de requisito).

Accepted (1 match)

Ticket	Summary	Component	Version	Milestone	Type	Priority	Created	Status
#357	Tela Visualização de Vidas - Apotamento de produção de lotes antigos afeta na contabilização de vidas de equipamentos recém montados.	Generico-Business	2.0	Desenvolvimento - Construção	defeito_interno	major	11/03/2013	accepted

Owned (18 matches)

Ticket	Summary	Component	Version	Milestone	Type	Priority	Created	Status
#166	Sistema não está fazendo nenhuma validação para a exclusão e edição de uma ocorrência.	Generico-Business	1.0	Desenvolvimento - Construção	defeito_interno	critical	06/02/2013	in_QA
#228	Programa com status Não Executado, apesar de ter uma corrida já Reclassificada.	Generico-Business	1.0	Desenvolvimento - Construção	defeito_interno	critical	18/02/2013	needs_work
#593	As abas de parada das modais de detalhes de produção (Aciaria/Laminação) não estão carregando as paradas de forma correta.	Generico-Business	3.0	Desenvolvimento - Construção	defeito_interno	critical	03/05/2013	needs_work
#476	Localização das telas nos menus não seguem a especificação / Padronizar títulos das telas	Generico-Business	3.0	Desenvolvimento - Construção	defeito_interno	major	22/04/2013	needs_work
#502	Tela Visualização de Vidas Laminação - Layout da tela quebra quando tem se 10 posições de cadeira em um laminador.	Generico-Business	3.0	Desenvolvimento - Construção	defeito_interno	major	25/04/2013	needs_work
#858	Tela de Abastecimento Laminação - Sistema está permitindo abastecer OP sem que a anterior tenha todos seus itens de emprego abastecidos.	Generico-Business	4.0	Desenvolvimento - Construção	defeito_interno	major	07/06/2013	assigned
#928	[Concorrência] Produção Endireitadeira - Sistema permite cortar um mesmo lote mãe mais de uma vez.	Generico-Business	5.0	Desenvolvimento - Construção	defeito_interno	major	26/06/2013	assigned
#948	Gráfico de Gantt - Data de término da produção real está definida pela data de confirmação da produção.	Generico-Business	5.0	Desenvolvimento - Construção	defeito_interno	major	27/06/2013	assigned
#1004	Programação\Produção Aciaria - PréFiltro Pendentes está com alguns problemas.	Generico-Business	5.0	Desenvolvimento - Construção	defeito_interno	major	03/07/2013	needs_work
#1039	Produção Aciaria - Programa permanece com status Em execução apesar de corridas finalizadas, canceladas e diluidas	Generico-Business	5.0	Desenvolvimento - Construção	defeito_interno	major	09/07/2013	needs_work

Figura 25 - Página do Trac Tickets abertos, visão de um desenvolvedor.

O software do track ainda permite também aos gerentes e coordenadores terem uma ideia melhor de como está o desenvolvimento, quem está fazendo o que e quantos bugs em média estão sendo encontrados.

Capítulo 5: Descrição da Implementação, Módulo de Produção:

Como já foram descritas a infraestrutura do software, as ferramentas usadas e os modelos de desenvolvimento adotados para que se pudesse desenvolver a aplicação do MÊS, pode-se neste capítulo pontuar as partes do software desenvolvidas por mim, os quais são os resultados obtidos na confecção deste PFC. Primeiramente será apresentada uma breve introdução aos cadastros do sistema, em seguida os as telas de produção da aciaria, e posteriormente são apresentados a parte de produção da laminação. No fim se apresenta a parte do Acabamento.

5.1: CRUD

Para o MES poder funcionar de forma correta, são necessários diversos tipos de informações: materiais produzidos, grau de aço, análises de qualidade necessárias, insumos e matéria-prima utilizados na produção, informações referentes a depósitos e baias onde serão guardados a produção, entre muitos outros.

Essas informações poderiam ser levantadas e colocadas diretamente no código (prática conhecida como “*hard coding*”). Apesar de esta ser a maneira mais eficiente em tempo de processamento, ela “engessa” o código de uma forma que se precisasse alterar alguma informação (como inserir um novo material para ser produzido ou alterar algum limite de qualidade de inspeção) seria necessária abrir o código, fazer as alterações e homologar novamente em um ambiente especifico. Isso se traduz em um alto custo para a empresa porque só depois de tudo isso feito é levado o novo programa para o ambiente de produção.

Visando a flexibilidade do sistema e a redução dos custos de manutenção, todas essas informações do processo são salvas no banco de dados e carregadas quando necessárias. Para facilitar o trabalho, para cada tabela de informação no

banco de dados foi criada uma tela de *“create, read, update, delete”* (*CRUD*, o que significa criar, ler, atualizar e apagar), ou seja, apenas as funções básicas de persistência dos dados. Sendo assim, não é necessário escrever diretamente no banco de dados essas informações, mas se criar via o sistema. O que possibilita validar se os dados estão coerentes, por exemplo, limite superior maior que o inferior, nomes não repetidos entre outras inconsistências de dados. Que viram causar inconsistências e “bugs” no sistema.

O MES utilizado no projeto pela Radix é um sistema relativamente grande que inclui mais de 40 diferentes *CRUD*s. Na Figura 26 pode-se ver um exemplo de uma tela de CRUD do sistema. Neste caso foi a tela de depósito, mas todas as outras telas seguem o mesmo padrão que é: uma aba superior para filtrar os resultados, uma tabela com as características e os dados que estão no banco, uma linha cada linha é um registro do banco de dados, um botão para abrir a modal de edição e outro para apagar o registro e um botão embaixo da tabela para adicionar novas entidades. Para o sistema funcionar todos os dados tem que estar cadastrados no sistema.

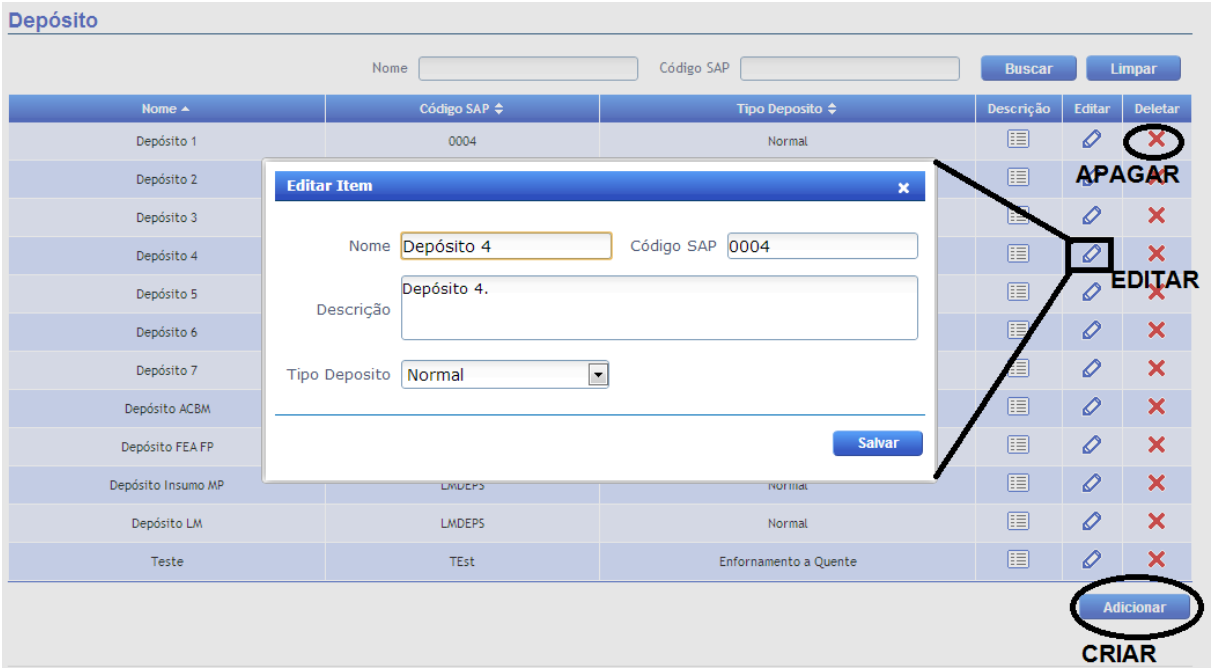

Figura 26 - Exemplo de Tela de CRUD - Depósito

5.2: Modulo de Programação da Produção

Apesar dessa parte não ser o foco principal do trabalho é necessário apresentar uma breve explicação para se entender o fluxo de produção no sistema.

A decisão de quais e em que quantidades os materiais serão produzidos na siderúrgica não é uma decisão no nível do sistema MES, ela é resolvida a nível gerencial e comercial onde se avaliam diversos fatores da economia, pedidos de clientes e peças em estoques. Em posse dos dados do que se devem produzir, estes são inseridos no sistema SAP (*Systeme, Anwendungen und Produkte in der Datenverarbeitung,* em português: Sistemas, Aplicativos e Produtos para Processamento de Dados). O SAP no final de cada mês envia os dados de quais materiais deverão ser produzidos no mês seguinte para a aciaria. Esses dados no sistema são chamados de ordem de produção. A produção da laminação e do acabamento é enviada no meio durante o mês corrente.

De posse dessas informações o operador da programação do MES, por meio da interface web, organiza e ordena a produção dos próximos dias com tempos teóricos de produção. Essas informações serão enviadas para o nível dois e ficarão visíveis para os operadores de produção para que estes possam por fim realizar a produção.

5.2.1: Programação Aciaria

Na aciaria recebe-se a informação de quais materiais deverão ser produzidos no mês inteiro. A partir disso o programador pode montar a programação repetindo os materiais que foram pedidos até o fim do mês. Conforme ele cria uma programação de *n* toneladas, o sistema MES cria um número de corridas suficiente para produzir essas n toneladas (já que o peso produzido por corrida é um parâmetro ajustável no sistema).

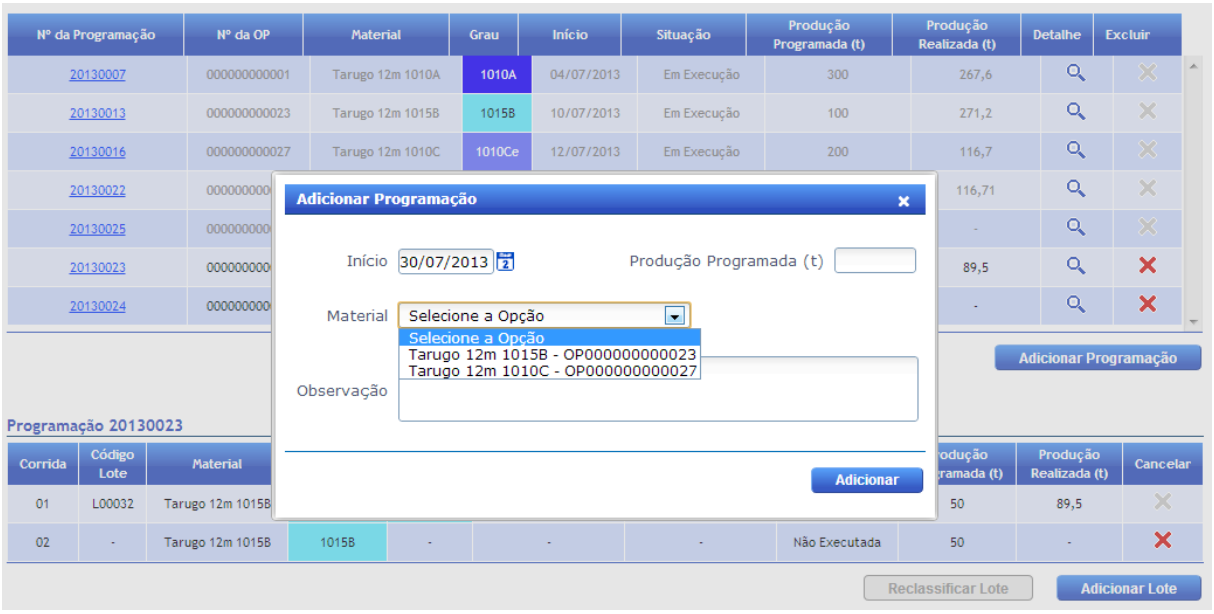

Figura 27 - Programador da Aciaria

5.2.2: Programação da Laminação/Acabamento

Diferente da aciaria (que recebe as ordens dos materiais por mês), a laminação e o acabamento recebem as ordens no decorrer do mês. Cada ordem de produção fica disponível na tela de programação para ser ordenada pelo programador, e para isso o usuário conta com uma funcionalidade de *drag and drop,* toda desenvolvida via *JavaScript*.

Com essas informações se tem as ordens em que os materiais serão produzidos, no chão de fábrica.

Figura 28 - Programador da Laminação

5.3: Acompanhamento Produção da Aciaria:

O ciclo de produção de uma aciaria para a indústria siderúrgica de aços longos foi previamente explicado no capitulo 2.2.1:. O ferro-gusa e a sucata são jogados numa panela onde são fundidos no Forno Elétrico a Arco. Depois o aço líquido é levado para o forno panela e fica um tempo até atingir propriedades químicas ideais. Por fim o aço líquido é despejado no lingotador (onde solidifica e é cortado tomando a forma do tarugo), e é posteriormente enviado ao pátio ou diretamente para a próxima etapa do processo.

5.3.1: Casos de Uso

Casos de uso são narrativas das funcionalidades de um sistema ou parte dele do ponto de vista do usuário. Nesse diagrama não são aprofundados os detalhes técnicos de como o sistema irá realizar as funções, ele apenas descreve qual deve ser o comportamento do sistema no caso do usuário realizar uma ação.

Para a produção da Aciaria temos apenas um ator, que é o apontador de produção, como exemplificado na Figura 29.

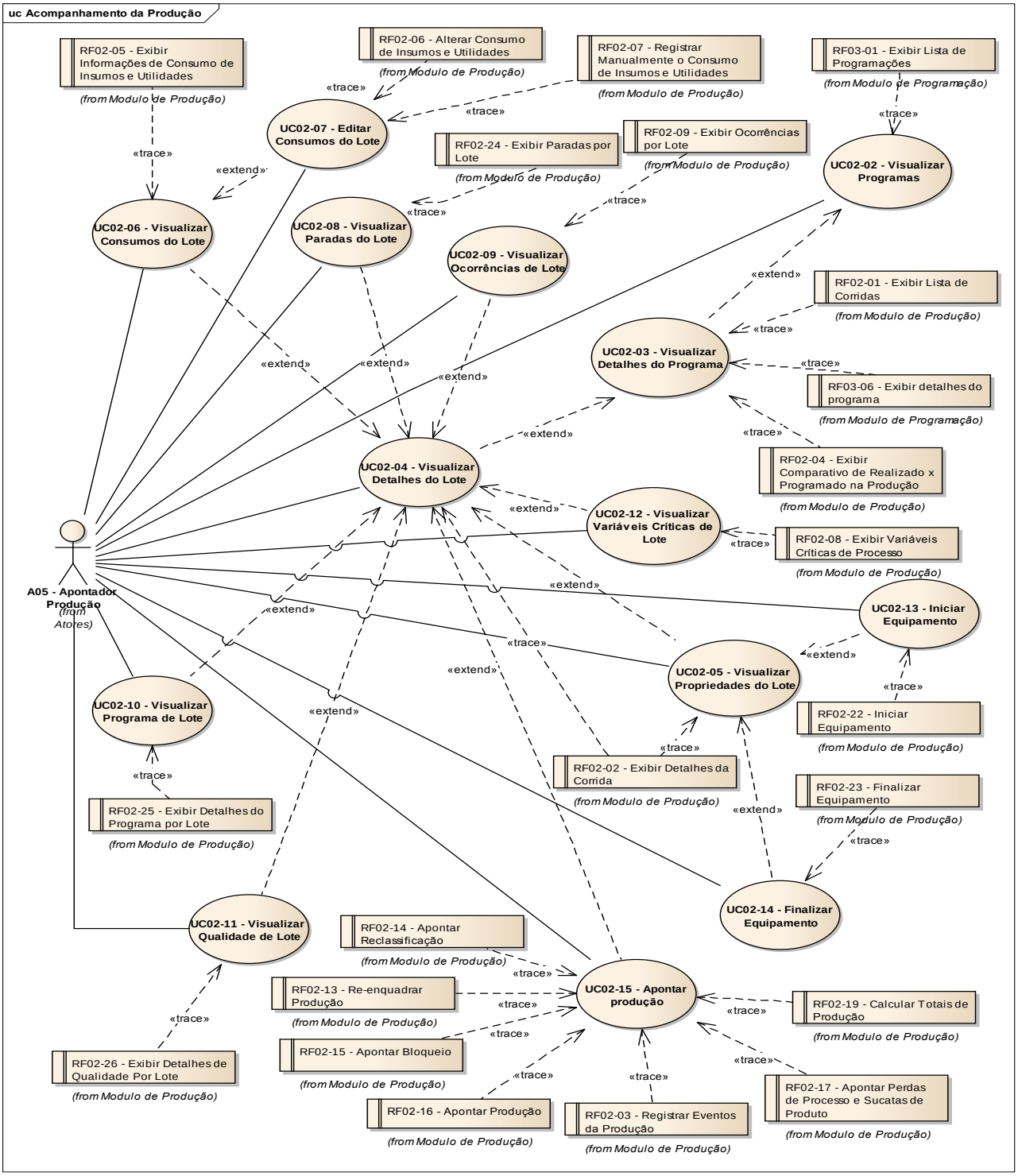

Figura 29 - Casos de uso do Acompanhamento de produção da Aciaria Diagrama UML

5.3.2: Descrição da Implementação

Uma vez descrita a infraestrutura criada para que se pudessem desenvolver o MES de maneira simples e objetiva, neste capítulo serão apresentadas um resumo das funcionalidades desenvolvidas para atender aos requisitos e casos de uso especificados com o cliente. Os mesmos são os resultados obtidos na confecção deste PFC.

5.3.2.1: Visualizar Programação

O objetivo de dessa funcionalidade é o de permitir a visualização dos programas de produção e suas informações, possibilitando a filtragem dos mesmos por:

- Período (Data Início e Data Fim),
- Código do programa,
- Material programado,
- Grau do aço programado,
- Situação do programa.
- Lote.

Como essa é uma das funcionalidades mais simples do acompanhamento da produção, será descrito um passo-a-passo do fluxo de informação do programa real.

O objeto do filtro é criado dinamicamente via *JavaScript* (JS) no navegador do usuário. O usuário escolhe o alcance da sua busca preenchendo os campos da interface gráfica (Figura 30). Quando o usuário acabar o preenchimento dos dados de sua busca, confirma a operação clicando no botão buscar. Esse evento de clique

do botão ativa uma função do JS que monta um objeto de filtro, criando uma requisição AJAX e a envia para o servidor da aplicação.

Figura 30 - Interface gráfica para filtro de Programações

A requisição é direcionada via URL na seguinte rota: servidor, aplicação e controlador. O objeto criado é uma *string* serializada que pode ser interpretada por qualquer linguagem de programação. Na Figura 31 é possível ver como foi construída a requisição lançada pelo navegador. Para criar a imagem foi utilizado o navegador *Google Chrome,* o que mostra a flexibilidade e robustez do sistema ao ser utilizado em plataformas diferentes da especificada (*Internet Explorer 8*).

Figura 31 - Requisição HTTP Post feita pelo navegador

A requisição chega ao servidor IIS (onde podem estar rodando diversas aplicações) que redireciona a requisição para a aplicação chamada, neste caso, de MESACOSLONGOSHML (ambiente de homologação do sistema). O framework

ASP .NET inicializa o controlador chamado “*ProduçãoAciaria*”, deserializa os parâmetros passados e realiza a chamada da função “*Filter*” com seus respectivos parâmetros. Na Figura 32 podemos ver o resultado da deserialização com a função do controlador correspondente sendo invocada.

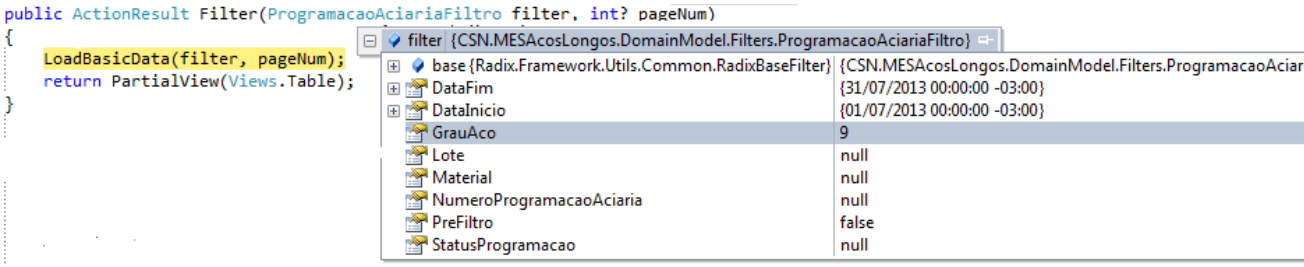

Figura 32 - Requisição AJAX chegando no controlador

O controlador por sua vez chama a aplicação de negócios, requisitando a lista de programações. Como a função de listar não tem nenhuma regra de negócio associada à camada de negócios, esta simplesmente redireciona a requisição para a camada de persistência, sendo que a camada de persistência, auxiliada pelos *frameworks* de acesso ao banco (Radix e .NET), injeta a *query* no banco de dados. A camada de persistência recupera os registros do banco os mapeia para as classes da nossa aplicação, no caso uma lista de programações (Entidade da Figura 33). A lista de programações da aciaria volta os níveis em que entrou: aplicação -> negócios -> controlador; o controlador, de posse da lista, retorna uma nova tabela em HTML *renderizada* via *razor*.

Na Figura 34 temos a resposta da requisição, uma com arquivo em HTML, que via *jQuery* carregamos no lugar da tabela que estava anteriormente. Na Figura 42 podemos ver a interface gráfica para o usuário final.

```
public class ProgramacaoAciaria : RadixBaseModel
{
    public int NumeroProgramacaoAciaria { get; set; }

    public int SequenciaProgramacao { get; set; }

    public OrdemProducao OrdemProducao { get; set; }

    public DateTimeOffset? DataCriacao { get; set; }

    public string UsuarioCriacao { get; set; }

    public DateTimeOffset? DataInicioPlanejada { get; set; }

    public DateTimeOffset? DataInicioRealizada { get; set; }

    public DateTimeOffset? DataTerminoRealizada { get; set; }

    public double QuantidadeProducaoProgramada { get; set; }

    public double? QuantidadeProducaoRealizada { get; set; }

    public int? TotalDePecasProduzidas { get; set; }

    public Status StatusProgramacaoAciaria { get; set; }

    public string Observacao { get; set; }
}
```

Figura 33 - Entidade Programação Aciaria

```
<table width="100%" border="0" cellspacing="0" cellpadding="0" class="grid" id="grid">
    <thead>
        <tr>
            <th style="text-align: center;">N&#186; da Programa&#231;&#227;o</th>
            <th style="text-align: center;">N&#186; da OP</th>
            <th style="text-align: center;">Material</th>
            <th style="text-align: center;">Grau</th>
            <th style="text-align: center;">In&#237;cio</th>
            <th style="text-align: center;">Situa&#231;&#227;o</th>
            <th style="text-align: center;">Produ&#231;&#227;o Programada (t)</th>
            <th style="text-align: center;">Produ&#231;&#227;o Realizada (t)</th>
            <th style="text-align: center;">Detalhe</th>
        </tr>
    </thead>
    <tbody>
        <tr class="even">
            <td align="center"><a style="cursor:hand" href="#" onclick="javascript:TabelaProducaoAciaria.CarregarTabelaDetalhe
            <td align="center">000000000027</td>
            <td align="center">Tarugo 12m 1010C</td>
            <td id="StyleProducao0" align="center">
                <input class="color {styleElement:'StyleProducao0'}" value="#6973FF" type="hidden"/>
                1010Ce
            </td>
            <td align="center">12/07/2013</td>
            <td align="center">Executada</td>
            <td align="center">100</td>
            <td align="center">11,67</td>
```

Figura 34 - Resposta da Requisição AJAX

Nº da Programação	Nº da OP	Material	Grau	Início	Situação	Produção Programada (t)	Produção Realizada (t)	Detalhe
20130019	000000000027	Tarugo 12m 1010C	1010Ce	12/07/2013	Executada	100	11,67	
20130016	000000000027	Tarugo 12m 1010C	1010Ce	12/07/2013	Executada	200	116,704	
20130017	000000000027	Tarugo 12m 1010C	1010Ce	12/07/2013	Executada	300	116,705	
20130022	000000000027	Tarugo 12m 1010C	1010Ce	17/07/2013	Em Execução	150	116,705	
20130024	000000000027	Tarugo 12m 1010C	1010Ce	17/07/2013	Não Executada	100	-	

Figura 35 - Resultado da Busca do Operador

5.3.2.2: Visualizar Lotes

Permite a visualização dos detalhes de um determinado lote. Cada programação é composta por “n” corridas da aciaria (Figura 36). O usuário apontador de produção seleciona a programação que ele quer abrir os detalhes e uma lista das corridas programadas é aberta para visualização (o esquema de chamada é o mesmo).

```
[Serializable]
public class CorridaAciaria : RadixBaseModel
{
    public ProgramacaoAciaria ProgramacaoAciaria { get; set; }
    public int NumeroSequencia { get; set; }
    public LoteAciaria LoteAciaria { get; set; }
    public EquipamentoCampanhaAciaria Panela { get; set; }
    public GrauAco GrauAcoRealizado { get; set; }
    public double QuantidadeProducaoPlanejada { get; set; }
    public double? QuantidadeProducaoRealizada { get; set; }
    public DateTimeOffset? DataInicioRealizada { get; set; }
    public DateTimeOffset? DataTerminoRealizada { get; set; }
    public string UsuarioCriacao { get; set; }
    public string ObsColetor { get; set; }
    public int? QuantidadePeca { get; set; }
    public StatusCorridaAciaria UltimoStatusCorridaAciaria { get; set; }
    public DateTimeOffset? DataConfirmacaoProducao { get; set; }
    public CorridaAciaria CorridaAciariaOrigem { get; set; }
    public string Justificativa { get; set; }
    public EnumTipoEnfornamento TipoEnfornamento { get; set; }
}
```

Figura 36 - Entidade Corrida

Nessa tabela o operador, ou o supervisor da área, pode ver a quantidade de corridas programadas, qual o lote que elas geraram o material que vai ser produzido, e então fazer uma comparação gráfica de qual grau aço foi produzido, qual grau foi programado, qual foi à produção programada e quanto foi efetivamente produzido. Além disso, ele pode ver a duração de cada corrida. No botão à direita está o coração do apontamento de produção (ele abre a janela de acompanhamento de produção).

Programação 20130022

Corrida	Código Lote	Material	Grau Programado	Grau Realizado	Início	Término	Situação	Produção Programada (t)	Produção Realizada (t)	Tipo Enfornamento	Enviado ao SAP	
01	L00030	Tarugo 12m 1010C	1010Ce	1010Ce	16/07/2013 21:02	16/07/2013 23:15	Reclassificado	50	116,7			
02	L00032	Tarugo 12m 1010C	1010Ce	1015B	16/07/2013 23:15	17/07/2013 01:15	Reclassificado	50	0			
03	L54321	Tarugo 12m 1010C	1010Ce	-	19/07/2013 15:50	19/07/2013 18:00	Executada	50	-			

Figura 37 - Tabela de detalhes dos lotes

5.3.2.3: Modal Acompanhamento da Corrida

5.3.2.3.1: Aba Propriedades

Cada corrida da aciaria passa pelos três equipamentos de produção desta: forno elétrico a arco, forno panela e lingotamento contínuo. É fundamental para o acompanhamento da produção saber os tempos de funcionamento de cada equipamento e onde a corrida se encontra. Esses dados devem ser adicionados manualmente nessa aba, ou podem ser recebidos do sistema de nível dois. Eles são salvos no banco de dados e podem ser acompanhados enquanto a corrida está em execução, ou até meses depois que ela terminou. Além disso, dispõe ao operador diversas informações como a numeração do lote e a panela utilizada no processo (ambas são recebidas do N2 ou inseridas manualmente) .

Para se salvar o status da corrida da aciaria criou-se uma tabela independente do banco de dados (Figura 38), onde é possível saber qual o estado

da corrida em determinado data e em qual equipamento de produção ela se encontra. O fluxo de inicialização e finalização de equipamentos é importante, pois está relacionado com o módulo de qualidade. Para cada equipamento que é iniciado são criadas as amostras necessárias para a equipe de qualidade.

```
[Serializable]
public class StatusCorridaAciaria : RadixBaseModel
{
    public CorridaAciaria CorridaAciaria { get; set; }

    public EquipamentoProducao EquipamentoProducao { get; set; }

    public DateTimeOffset DataInicio { get; set; }

    public DateTimeOffset? DataInicioOperador { get; set;}

    public DateTimeOffset? DataInicioN2Producao { get; set; }

    public DateTimeOffset? DataInicioN2Evento { get; set; }

    public DateTimeOffset? DataTermino { get; set; }

    public string UsuarioCriador { get; set; }

    public EnumStatusCorridaAciaria Situacao { get; set; }

    public string DataInicioString { get; set; }
```

Figura 38 - Entidade utilizada para representar o estado da corrida

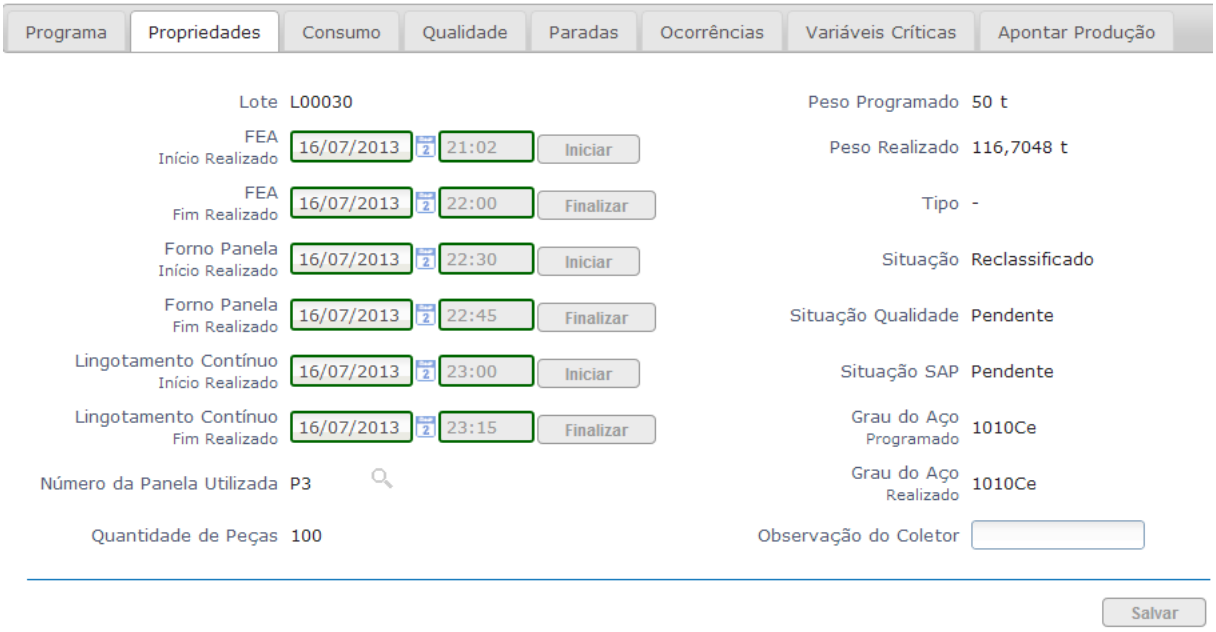

Figura 39 - Aba de Propriedades

5.3.2.3.1.1: Integração com Módulo de Qualidade

Após o operador iniciar a corrida no forno elétrico a arco (sua requisição é enviada até a classe responsável pelas regras de negócio), primeiramente é feito um tratamento de erro onde verifica-se a consistência dos dados em relação ao estado atual do sistema. Caso encontre um erro (por exemplo o programador cancelou essa corrida), retorna uma mensagem de erro ao usuário. Caso não tenha nada de errado nessa mensagem, são duas regras de negócios que tem que ser cumpridas: atualizar o novo estado do sistema e acessar a classe do módulo de paradas responsável por inserir as amostras. A consistência dos dados é garantida através de transações no banco de dados. O *Framework* da Radix trata as transações de forma praticamente automática, e com ao auxilio da biblioteca: "*Unity*.*InterceptionExtension"* conseguimos capturar todas as invocações aos métodos da camada de negócios.

Para isso basta atribuir um atributo ao método chamado para garantir que todas as escritas no banco se encontrem dentro de uma mesma transação, conforme Figura 40.

```
[RequiresTransaction]
public virtual RetornoSalvarStatusCorridaAciaria AdicionarStatusEquipamento(StatusCorridaAciaria novoStatus,
{
```

Figura 40 - Avisando ao Framework que é necessário abrir uma transação para essa função.

O uso de interceptadores (*Figura 41*) também é usado para centralizar a escrita do *log* de erros na aplicação, que é toda exceção não tratada (*try*/*cat*ch). Seu uso está de acordo com as filosofias de desenvolvimento ágeis adotada pela Radix, porque diminui o tempo necessário para desenvolver uma funcionalidade, reduz a duplicação do código, acrescenta uma grande melhoria para a qualidade da aplicação, garantindo que todas as funções rodarão aquela parte do código de escrita no *log* e de abertura de transação, incrementa consideravelmente a qualidade do código tornando-o mais limpo (vão ter apenas as regras de negócio), não precisando se preocupar com abertura de transações e tratamento de erros no meio do código.

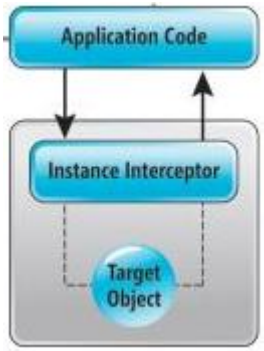

Figura 41 - Interceptadores

5.3.2.3.1.2: Integração com o Nível Dois

Diversos dados da aba de propriedades são informações que são processadas no nível dois. Para ser possível a comunicação entre dois sistemas diferentes é necessário escolher um padrão, e o padrão adotado pelo cliente para

fazer a comunicação entre os seus sistemas é o padrão *XQI* (*XML Query Interface)*, já que o objetivo de adotar um padrão é de minimizar o trabalho dos desenvolvedores. O meio de campo é feito por um serviço da própria empresa, que garante a entrega da mensagem.

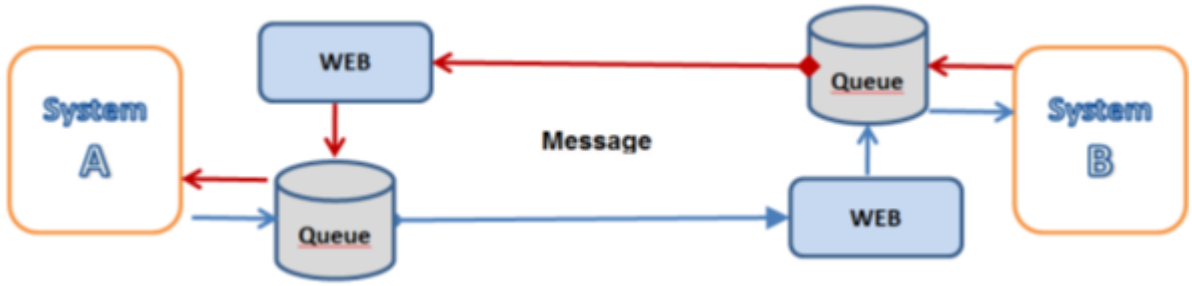

Figura 42 - Visão Geral Integração XQI

O padrão XQI consiste em dois grupos de tabela: uma para mensagens que o sistema ira receber (*IN*) e outro que o sistema irá enviar (*OUT*). Cada grupo é composto por 3 tabelas, sendo a primeira de cabeçalho chamado (XQI_(IN/OUT)_HEADER, a segunda contendo as informações da mensagem (XQI_(IN/OUT)_BODY e uma terceira usada para fazer o log dos erros (XQI_(IN/OUT)_ERRORLOG). Os dois grupos de tabela anteriores estão disponíveis em ambos sistemas, então quando o MES deseja enviar uma mensagem ao nível dois ele escreve na tabela XQI_OUT_HEADER e XQI_OUT_BODY, informando no cabeçalho que o destinatário é o sistema de nível dois. Consequentemente, o serviço XQI entrega a mensagem para as tabelas XQI_IN_HEADER e XQI_IN_BODY do banco do nível 2.

São estes os campos da tabela:

- XQI_*_HEADER (campos de endereçamento e informações de gerais):
    - *Source* foi quem enviou a mensagem, por exemplo: "PROCOM_ALVR_EAF” (forno elétrico).
    - *Message_Id* significa identificador único.

- *Target* é quem irá receber a mensagem MES_LONGOS_VR.

- *Message_Type:* diz qual mensagem é, por exemplo "EAF_HETD" significa "término da produção do forno elétrico".

- *Expiration_Time*: data de validade da mensagem.

- *Msg_Status Flag*: para sinalizar erros no processamento.

- *Date_Time_In*: data/hora de recebimento da mensagem.

- *Date_Time_Proc*: data/hora que o sistema realmente processou a mensagem.

- *Retry_Count*: número de tentativas de processamento. O padrão XQI tenta processar a mensagem automaticamente 5 vezes, se não há sucesso em nenhuma dessas cinco tentativas ele marca a mensagem como erro.

- XQI_*_BODY (corpo da mensagem): onde diz que os dados realmente são enviados, e cada mensagem pode ter uma ou mais linhas de corpo:

  - *Source* e *Message_Id* são ambos iguais ao *Header*, ele associaa o cabeçalho ao corpo.

  - *Field_Seq*: identificador único.

  - *Feature*: o campo, por exemplo, "BEGIN_TIME" da mensagem "EAF_HETD" é o tempo de inicio do forno elétrico.

  - *Value*: valor do campo (*feature)* passado. Cada campo de cada mensagem é salvo como uma cadeia de caracteres, permitindo passar qualquer tipo de dado. É combinado a priori com os desenvolvedores de ambas aplicações o formato que campo vai ser escrito. No exemplo acima

o campo *begin time* é passado como: “YYYYMMDD hhmmss” (ano, mês, dia, espaço em branco, hora minuto, segundo).

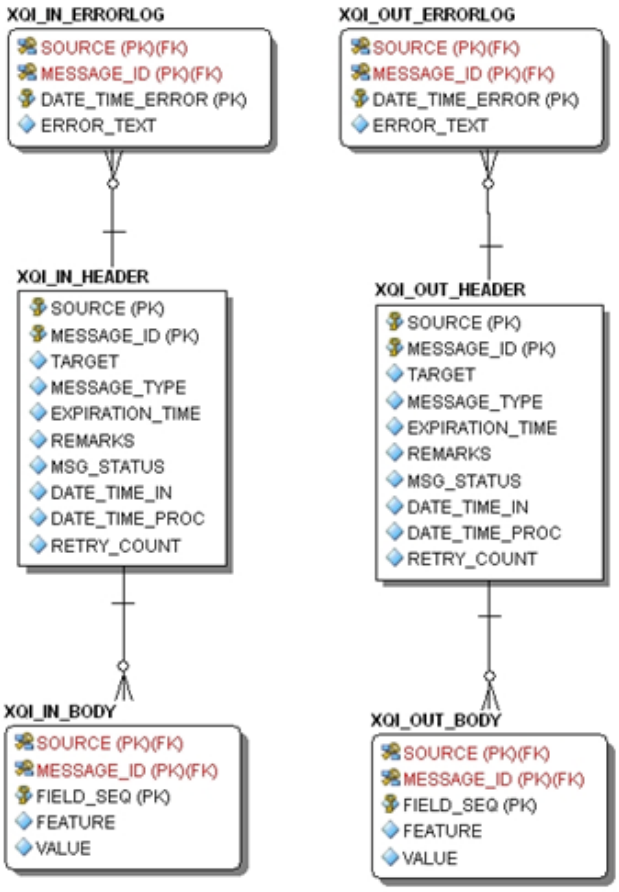

Figura 43 - Tabelas de Entrada e Saída do padrão XQI

O serviço XQI do cliente é um serviço passivo que copia o que está na tabela de saída de um sistema para a tabela de entrada do outro sistema. Para o MES poder processar os dados, foi necessário o desenvolvimento de um serviço Windows chamado de XQIMapperService. O trabalho do serviço é o mais simples, ele lê a tabela de entrada do sistema, vê se tem algo novo, lê a mensagem novamente e se a mensagem estiver mapeada, chama a função da camada de negócios especifica para processá-la. Caso ocorra algum erro no processo, escreve-se no log (arquivo *.txt* e na tabela XQI_*_LOGGER). Se o número de reprocessamento da mensagem bateu no limite, então a mensagem é marcada como *erro* (no campo MSG_STATUS). Se a mensagem for processada sem

problemas, marca-se que já foi processada (para não precisar processar novamente).

5.3.2.3.2: Aba Qualidade, Ocorrências e Paradas:

Essas abas fazem interface com outros módulos do sistema. Elas fornecem informações para o operador da planta (são apenas para leitura).

O módulo de paradas é um modulo onde são registradas todas as paradas do sistema, tanto as programadas (para manutenção) quanto as não programadas (falhas de equipamento).

O módulo de qualidade é onde ocorre toda a inspeção do aço a fim de garantir que ele tenha as propriedades químicas necessárias (como “composição de carbono”).

Já o modulo de ocorrências registra ocorrências genéricas do sistema (essas informações também estão disponíveis apenas para leitura).

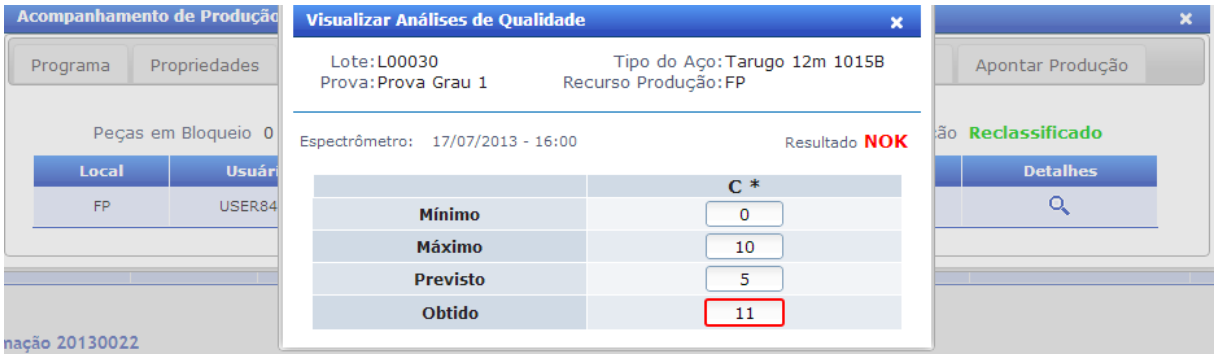

Figura 44 - Interface com o resultado das análises de qualidade da corrida

5.3.2.3.3: Aba Apontamento de Produção

Depois de a corrida terminar (ou seja, ela ter passado por todos os equipamentos), as amostras da qualidade são confirmadas pelo responsável. O operador da aciaria passa pelo processo final de confirmar a produção. Neste módulo ele é responsável por informar ao MES quantas peças foram produzidas e de quanto foram as perdas de produto e/ou processo que ocorreu. Porém, a mais importante responsabilidade é decidir qual o destino das peças. O operador tem três escolhas: ele pode colocar as peças em livre utilização, ele pode bloquear elas ou então reclassificar a localização física (depósito/baia) das mesmas.

Se as peças estiverem liberadas, será possível utilizá-la no processo posterior, a laminação. Se alguma delas estiver bloqueada, esta será enviada para a equipe de qualidade que terá o trabalho de decidir o destino: ou a peça vira sucata ou o lote de peças é liberado.

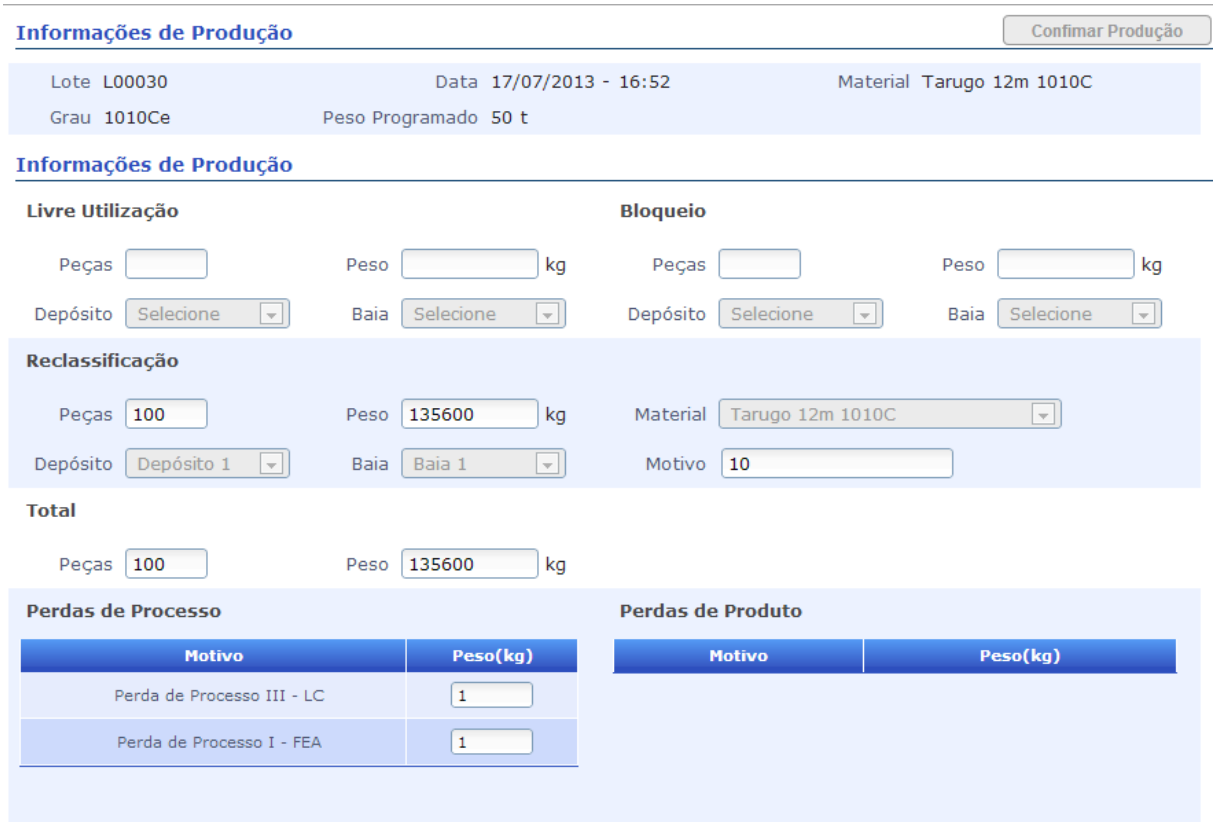

Figura 45 - Aba de Apontamento de Produção

Capítulo 6: Conclusões e Perspectivas

O trabalho desenvolvido durante o estágio na Radix foi uma ótima oportunidade de crescimento profissional porque permitiu o contato com o mercado de trabalho, como também foi possível poder sair como engenheiro com uma excelente experiência em um projeto expressivo na área de Engenharia.

A indústria siderúrgica é uma área muito interessante para a engenharia. Com processos bastante complexos, é necessária a utilização das mais recentes tecnologias para a área de TI visando poder competir de frente com um mercado cada vez mais seletivo.

O interesse pela área de software e programação despertado em mim durante a graduação tornou-se ainda mais acentuado durante o período de estágio. A experiência de participar tão expressivamente de um projeto de grande porte foi muito gratificante. Foi enorme o conhecimento adquirido, não só na área técnica, mas também estar junto e presenciar como se planeja e se organiza equipes para construção de softwares relativamente complexos.

Ao fim dessa parte do projeto, considero que tanto o meu trabalho em particular como o trabalho da equipe de desenvolvimento que participei foram um sucesso.

Falando individualmente, entrei no projeto sem saber nada sobre as linguagens utilizadas e com um conhecimento muito raso sobre a estrutura de programação utilizada. Comecei apenas programando telas de Cadastro, as mais simples do sistema. No entanto, tive uma curva de aprendizado alta e já no meio do projeto fiquei responsável por telas de alta complexidade e importância para o sistema (as telas de produção).

Em nível de equipe foi feito um ótimo projeto, cumprimos os prazos de entrega corretamente. Tanto nosso modelo de dados como o software foi sempre validado sem muito estresse, e mesmo tendo algumas alterações no requisito em tempo de projeto conseguimos executa-las sem estragar ou atrasar o que estava sendo desenvolvido.

Mas o projeto ainda não está terminado, ainda temos alguns meses de testes reais de integração com os outros sistemas, e alguns meses de operação assistida dos quais espero poder participar.

Para a Radix o sucesso em projetos desse tipo tem grande importância. A realização de projetos com qualidade, dentro do prazo e do orçamento aumenta a visibilidade e o reconhecimento da empresa junto aos seus clientes. O sucesso em projetos nessa área permitiu à Radix expandir seu portfolio de clientes para muito além da área de petróleo e gás (no qual muitas empresas de engenharia ainda são extremamente dependentes).

Novos projetos de desenvolvimento dessa área estão por vir nos quais irei participar ativamente com engenheiro.

Bibliografia:

[1] *FILGUEIRAS, Dra. Lúcia V. L.; MELNIKOFF, Dra. Selma Shin Shimizu, "Engenharia de Software"*

[2] *Site RI*

http://www.mzweb.com.br/csn/web/default_pt.asp?idioma=0&conta=28

[3] *V.B. Mazzola e J-M Farines, "Metodologias de Concepção de Software e de Sistemas"*

[4] *B. Boehm. Balancing "Agility and Discipline: A Guide for the Perplexed".*

[5] *R. C. Martin – "Código Limpo, Habilidades Práticas do Agile Software".*