Universidade de Aveiro Departamento de Electrónica, Telecomunicações e Informática

2011

Diogo Manuel Albuquerque Diogo Figueira

GEMA – Sistema de Gestão Magra

GEMA - SISTEMA DE GESTÃO MAGRA

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia de Computadores e Telemática, realizada sob a orientação científica do Dr. Joaquim Arnaldo Martins, Professor do Departamento de, Telecomunicações e Informática (DETI) da Universidade de Aveiro

Diogo Manuel Albuquerque Diogo Figueira Nº 18043

Dedico este trabalho aos meus pais pelo apoio que sempre me deram na minha vida.

O júri

Presidente	Joaquim Manuel Henriques de Sousa Pinto Professor Auxiliar da Universidade de Aveiro
Orientador	Joaquim Arnaldo Carvalho Martins Professor Catedrático da Universidade de Aveiro
Arguente	Fernando Joaquim Lopes Moreira Professor Associado da Universidade Portucalense

Palavras - chave Produção, *lean*, *kanban*, *gestão*, *just-in-time*, stock, *kaizen*, logística, simulação

Resumo

As mudanças comportamentais nos negócios industriais foram iniciadas pelos pressupostos de Taylor, exigindo que os novos gestores implementassem acções técnicas, que se traduzissem em ganhos operacionais motivados, principalmente, pela adopção de técnicas e ferramentas já conhecidas e aplicadas em processos produtivos.

Este novo comportamento tem como suporte as filosofias de melhoria contínua (kaizen), introdução de práticas de prevenção de erros, introdução do sistema *pull*, *empowerment*, tornando os sistemas de produção flexíveis, orientados à satisfação do cliente, através da eliminação dos desperdícios.

O software de gestão de produção tem, por isso, um papel muito importante. Armazena os dados de todo o processo de produção, permitindo, com recurso a diversas ferramentas, monitorizar, melhorar e alterar as diferentes etapas.

Pretende-se, então, combinar três aspectos importantes. O primeiro passa por manter o desenvolvimento do software e respectiva manutenção com custos baixos, de forma a ser suportável pelo mercado que se pretende abranger. O segundo aspecto tem como objectivo que, na solução final, estejam presentes as ferramentas essenciais para a gestão de produção. No terceiro e último aspecto pretende-se que a solução seja escalável, com desenvolvimentos à medida e com possibilidade de integração com ferramentas de terceiros.

Keywords production, lean, kanban, management, just-in-time, stock, kaizen, logistic, simulation

Abstract

The behavioural changes in the industrial business were started by the assumptions of Taylor, demanding that the new managers implement technical actions, which resulted in operating profit motivated primarily by the adoption of tools and techniques known and applied in production processes.

This new behaviour is supported by the philosophies of continuous improvement (kaizen), introduction of practices to prevent errors, introduction of pull, empowerment, making productions systems more flexible, oriented by customer satisfaction, by eliminating waste.

The production management software has therefore a very important role. Storing the data of the entire production process, allowing the use of various tools, monitor, improve and change the various steps.

It is intended then to combine three important aspects; The first is to maintain the development of software and its maintenance costs low, to let it to be affordable by the market that is intended to cover. The second aspect is that, in final solution, the essential tools for production management are present. In the third and final aspect, is intended that the solution to be scalable, allowing custom developments and possible integration with third-party tools.

Índice

Índice Imagens

Índice Tabelas

1. Introdução

O *Lean Management* é uma filosofia de gestão que procura optimizar os recursos existentes numa empresa, independentemente da área de actuação (indústria, serviços, etc.), de forma a reduzir desperdícios e aumentar a eficiência, eficácia, mantendo ou mesmo melhorando a qualidade, com o objectivo de majorar os lucros.

Existem no mercado ferramentas (1) que suportam esta filosofia, ajudando a transportar para o terreno as implementações pretendidas, monitorizar a produção alertando para anomalias, prever o comportamento desta no caso de alteração das filosofias, entre outras possibilidades.

Há, no entanto, um grande défice de oferta que se enquadre na realidade do tecido empresarial português. As soluções comerciais são caras ou demasiado complexas e as opções mais acessíveis estão principalmente vocacionadas para a parte financeira, não existindo um suporte informático que apoie as operações.

Isto faz com que as PME's encontrem uma grande barreira para a adopção da filosofia lean. Existe muita dificuldade em fazer a "leitura" do que se passa no terreno para o sistema informático e de forma coerente.

O desafio prende-se com a construção de um sistema de informação para gestão de produção, de baixo custo, que consiga integrar as ferramentas lean de utilização simples e rápida, permitindo uma maior disseminação do Lean Management no tecido empresarial português.

1.1. História

➢ Os primeiros Passos

A Gestão Magra (do Inglês *Lean Management*) começou a dar os primeiros passos pela mão de Eli Whitney que, embora tenha ficado famoso pela invenção da limpeza de algodão, criou um sistema de produção, que viria a ficar conhecido por Sistema Americano de Produção. Em 1799 conseguiu um contrato com o exército dos Estados Unidos a um preço baixo, imbatível, usando peças intercambiáveis. Esta uniformização permitiu que pessoas com baixas qualificações conseguissem um produto final com a mesma qualidade de um especialista, porém em muito menos tempo. **(Mirsky, 2011)**

Até finais do século XIX o sistema de produção conheceu mudanças significativas, não olhando para a cadeia de produção como um todo e a forma como o trabalhador executava a tarefa.

Foi, então, que Frederick W. Taylor começou a olhar para o trabalhador e métodos de trabalho, ignorando as ciências comportamentais. Aplicou métodos científicos cartesianos, com a atenção virada para a eficiência e eficácia na gestão, a que chamou gestão científica. Surgiram, assim, os estudos de tempo e trabalho normalizado. **(Papesh, 2011)**

(1) Ver capítulo 6

Nesta altura apareceram também Frank e Lillian Gilbreth. Frank ficou conhecido por elaborar análises de tempos e movimentos e pela invenção dos Gráficos de Processo. Estes focavam a atenção em todos os elementos, incluindo os que não acrescentavam valor e que, geralmente, ocorriam entre os processos normais. Já Lillian adicionou a psicologia, estudando a motivação dos trabalhadores, o modo como as atitudes afectam os processos. **(womak, 2007)**

Frederick W. Taylor, Frank e Lillian Taylor são considerados os fundadores do Taylorismo e originaram a ideia de eliminar o desperdício.

➢ Produção em Massa

Henry Ford (1863-1947) iniciou a sua carreira profissional como mecânico, chegando posteriormente a engenheiro-chefe. Em 1903 fundou a Ford Motor Co.

No início, a produção de veículos processava-se da mesma forma que nas outras fábricas, um veiculo de cada vez.
Em 1913, ele e o seu braço direito Charles E. Sorensen, idealizaram e concretizaram o sistema de produção em massa, reordenando todos os elementos presentes na produção, de maneira a formar uma linha contínua de produção. A concretização da sua primeira linha de montagem deu-se na primeira fábrica Ford em Highland Park, Michigan, tornando-se um marco de referência para os métodos de produção em série, no mundo.

O sucesso deste sistema inspirou muitos outros a copiar, mas muitos não percebiam os fundamentos, aplicando a mesma técnica em produtos que não eram adequados.

O sistema inicial pecava pela "rigidez". Todos os automóveis eram exactamente iguais, não dando ao consumidor possibilidade de escolha.

Em meados da década de 1920 a GM, uma concorrente da Ford, elaborou um sistema de produção em massa bastante flexível, usando máquinas de rápida adaptação. Optou por construir as peças dos seus automóveis em diversas fábricas e não todas na mesma, como acontecia na Ford. No caso de alteração de algum dos seus componentes, este sistema permitia testar e melhorar a mesma em pequena escala e, só depois, alterar a produção nas fábricas principais, levando, assim, pouco tempo na implementação.

Em contraste, quando o mercado ficou saturado do modelo T e houve a necessidade de seguir em frente com um novo modelo, o Modelo A, a Ford teve de parar a sua fábrica durante 6 meses, mergulhando a companhia num caos. Depois desta fase, a Ford acabou por descentralizar a sua produção.

Durante a 2ª Guerra Mundial o sistema de produção em massa foi um factor muito importante, já que permitiu construir equipamentos militares em grande escala.

É no pós guerra que o sistema atinge o seu auge, entre os anos de 1955-1960, período conhecido pela época dourada do capitalismo, em que se atingiu o fabuloso número de veículos vendidos por ano. **(womak, 2007)**

Contudo, há diversos aspectos a ter em conta, que penalizavam este sistema:

- Exigia um grande investimento em fábricas e máquinas, que necessitavam de muito espaço.
- Era necessário elevado nível de stock, quer de matéria-prima, quer de produto final.
- Não era aplicável a mercados pouco consumistas, como o do Japão, por exemplo
- Ainda gerava muito desperdício.

➢ Just In Time (JIT) e Toyota Production System (TPS)

Os dois pilares do sistema são a automação inteligente ou "Jidoka" e o *Just in Time Manufacturing* (JIT).

A automação inteligente surgiu ainda antes de ser criada a Toyota, pela mão do seu fundador Sakichi Toyoda, na produção de teares mecânicos. O exemplo que mais impacto teve foi o modelo G, um tear mecânico de alta velocidade e autónomo. Entre outros aspectos a implementação de sensores mecânicos (*um grande avanço para a época*) permitiram eliminar o desperdício (*quando uma linha partia a máquina parava, o que impedia que a peça ficasse com defeito*), o erro humano e a necessidade de haver um operador por máquina, podendo apenas um operador estar a acompanhar mais de 30 teares.

O JIT surgiu pela mão do filho de Sakichi, Kiichiro. Ignorando os avisos, Kiichiro apostou na produção de automóveis, tendo obtido o seu primeiro protótipo em 1935, o modelo A1. Em 1937 estabeleceu a primeira fábrica de do grupo Toyota, aperfeiçoando os métodos de produção em massa americanos e dando início ao conceito JIT. A ideia era eliminar o desperdício produzindo apenas o necessário, quando era necessário e na quantidade necessária. Melhoraram o conceito JIT, aplicando os seus princípios, sistematicamente, eliminando dessa forma o desperdício e "esticando" os recursos limitados.

Já depois da 2ª Guerra Mundial, os Estados Unidos tinham 8 vezes maior capacidade de produção de automóveis e a Toyota não possuía os recursos para poder superar esta diferença. Eiji Toyoda, primo de Kiichiro, designou Ohno Taiichi para desenvolver um sistema de produção mais eficiente. Aplicaram o conceito "Jidoka" em todas as operações, de forma a aumentar o valor da produtividade de cada operário. Ohno adaptou uma prática dos supermercados americanos, tornando cada processo "cliente" do anterior, ou seja, quando um processo "consumia" uma peça, usando o Kanban (*palavra japonesa que significa registo ou placa visível*) era feito um pedido para o processo anterior produzir nova peça.

Estes foram os conceitos iniciais do TPS, os quais estão sempre em evolução, adaptando-se e procurando aumentar o valor do produto e eliminar os desperdícios.

O Sistema de Produção da Toyota ganhou a atenção do resto do mundo, depois da crise do petróleo em 1973, quando a Toyota recuperou muito mais rapidamente que as outras empresas**. (womak, 2007)**

Em 1990 é editado o livro "*The Machine That Changed de World*", que percorre a história da produção de automóveis, combinado com um estudo do sistema implementado nas fábricas da América, Europa e do Japão. Este livro, para além de mostrar ao mundo o porquê do sucesso do TPS, sintetizou os conceitos, formando o que é hoje conhecido como "Lean Manufacturing" ou Produção Magra.

➢ Princípios do sistema de produção da Toyota

O sistema de produção da Toyota possuí 14 princípios e que podem ser agrupados em 4 grupos.
Estes princípios são a base do Sistema de Produção Magra (Lean Management)

I - Filosofia como fundação

Princípio 1 - Basear as decisões de gestão numa filosofia de longo prazo, mesmo à custa de resultados financeiros de curto prazo.

Por mais tentadora que seja a ideia de reduzir custos, o primeiro passo é o de definir a filosofia da empresa, criar as bases de negócio e olhar para o longo prazo, diferenciando-se da concorrência.

A Toyota tem sempre como objectivo inicial criar valor para os clientes para a sociedade e para a economia. Conhecer o mercado, os gostos e as necessidades dos clientes é de extrema importância. Este deverá ser sempre o ponto de partida não só para os produtos ou serviços, mas para a companhia no seu todo.

II - O Processo adequado produzirá os resultados adequados

Princípio 2 – Criar um Fluxo Contínuo do Processo para trazer os problemas para a superfície.

Redesenhar o processo de trabalho de forma a alcançar o fluxo tipicamente resulta no gasto na ordem de um décimo do tempo necessário do que anteriormente gasto.

Este fluxo é mais evidente no sistema de produção, mas também está presente a nível organizacional, o qual é focado numa cultura de acréscimo de valor.

A razão para criar o fluxo não é só ter material ou informação em movimento rápido, é ligar os processos e as pessoas para que os problemas surjam de forma quase imediata.

O fluxo é a verdadeira chave para o melhoramento contínuo.

Princípio 3 – Usar Sistemas "Puxar" para evitar excesso de produção

Devido à mudança de procura e diferentes níveis de procura por parte dos clientes, a Toyota adoptou um método baseado no sistema americano de supermercados, tendo apenas pequenas quantidades de cada produto e com reabastecimentos frequentes.

Para que o sistema funcione, foi implementado um sistema de cartões, conhecido por Kanbans, para sinalizar a necessidade ou não se reabastecimento dos produtos no supermercado.

Princípio 4 – Nivelar a carga de trabalho (trabalhar como a tartaruga, não como a lebre)

A única maneira realista de criar um fluxo contínuo é conseguir alguma estabilidade. Se a procura numa organização tem muitos altos e baixos, força a que esta trabalhe de forma reactiva e, naturalmente, os desperdícios irão começar a aparecer.

Princípio 5 – Construir uma cultura de parar para resolver problemas para obter qualidade logo à primeira.

A qualidade e o desenvolvimento de um produto de qualidade tem valido à Toyota a obtenção de prémios de prestígio.
As ferramentas usadas são as mesmas das outras companhias, porém a filosofia introduzida pelo fundador da Toyota, Sakishi Toyoda, faz toda a diferença:

> Quanto há um problema não deixes as coisas continuarem com a intenção de as solucionar mais tarde. Para e arranja-o agora. A produtividade pode sofrer no imediato, mas a longo termo a produtividade será melhorada á medida que os problemas forem sendo solucionados.

Princípio 6 – Normalização de tarefas e processos são a base para a melhoria contínua e o *fortalecimento* dos trabalhadores.

Não se pode prever o timing e saída dos processos a menos que estes sejam estáveis e repetitivos. O princípio base para o fluxo de processos e do sistema de "puxar" é ser previsível e repetitivo.

Contudo a standarização e muitas vezes confundida com rigidez, mas na cultura da Toyota é exactamente o oposto, ao standarizar as melhores práticas de hoje, eles capturam o conhecimento até este ponto. A tarefa de melhoria contínua é então melhorar esse padrão, sendo estas posteriormente também elas.

Princípio 7 – Usar controlos visuais para que nenhum problema permaneça escondido.

Nestes dias em que tudo é informatizado, poderia pensar-se que o uso de papel é uma forma de desperdício, puro engano. O Homem é uma criatura visual. Por exemplo o uso de sistemas visuais no local de trabalho permite ao trabalhador ver a situação se encontra dentro da standarização ou não de forma rápida e eficaz ou então um gráfico bem desenhado fixo numa parede permite que realizar uma discussão muito frutuosa.

Princípio 8 – Use apenas tecnologia confiável e exaustivamente testada e que sirva os trabalhadores e os processos.

O sistema de produção da Toyota centra-se fortemente na estabilidade, fiabilidade e previsibilidade. A introdução de uma nova a organização ou no produto deve ser feita apenas se for provada a sua necessidade e depois de testada intensivamente.

Secção III - Acrescenta valor à organização formando o pessoal e Parceiros

Princípio 9 – Formar líderes que conheçam minuciosamente o trabalho, vivam a filosofia e a ensinem aos outros.

Na Toyota os líderes são criados dentro da própria empresa. Estes têm de ter bem interiorizado os conceitos da empresa, exemplificando a filosofia no que fazem e ensinando a maneira de trabalhar da Toyota aos outros. Desta maneira as metodologias vão permanecendo sempre presentes dentro da empresa e ao longo do tempo.

Princípio 10 – Formar pessoal e equipas excepcionais que sigam a filosofia da Empresa.

A Toyota tem uma forte cultura interna é muito consciente da importância a manter em todos os seus trabalhadores, estando permanentemente a reforça-la.
A essência é ter indivíduos equipas e excepcionais que trabalhem dentro da filosofia do Sistema Toyota de Produção para alcançar excelentes resultados. As ferramentas são apenas ferramentas, as pessoas que utilizam as ferramentas e como eles os usam é que marca a diferença.

Princípio 11 – Respeitar toda a rede de parceiros e fornecedores desafiando-os e ajudando-os a melhorar.

A Toyota olha para os seus fornecedores como uma extensão da companhia. Estes são desafiados a melhorar as equipas da Toyota ajudam os fornecedores a descobrir e resolver problemas para que se tornem melhores e mais fortes.

Secção IV - A continua resolução de problemas conduzem à aprendizagem da organização

Princípio 12 – Vai e vê com os teus olhos para que possas perceber a situação com pormenor.

Não se consegue resolver problemas e melhorar sem se conhecer pormenorizadamente a situação actual. Isto significa ir ao local do problema, observar e analisar o que é que se passa.
É impossível conseguir resolver problemas de forma remota, apenas observando números e depois teorizar sobre as possíveis soluções.

Princípio 13 – Tomar decisões lentamente e de forma consciente, considerando exaustivamente todas as opções. Implementar as decisões rapidamente.

É do conhecimento comum que a gestão Japonesa se move de forma lenta quando se trata de tomar decisões com o intuito de criar consenso. Isto em parte é verdade, mas a chave não é criar consenso, é explorar potenciais problemas e soluções para assim obter a melhor resposta.

Princípio 14 – Torne-se numa organização de aprendizagem através da reflexão e da melhoria contínua.

A melhoria contínua vem depois de se conseguir a estabilidade dos processos. Quando esta já foi alcançada e se tem os desperdícios e ineficiências bem visíveis então temos a oportunidade de aprender de forma continua.

A aprendizagem é feita através das pessoas e também aqui é necessário ter estabilidade de pessoal, promoções lentas e sucessões pensadas de forma a proteger o conhecimento da organização.

1.2. Objectivos

Com este trabalho pretende-se ter uma solução que, por um lado, possua as funções e ferramentas de gestão de produção essenciais, gerindo os elementos que fazem parte de todo o processo produtivo, com base nos fundamentos e metodologias de produção magra e que, por outro lado, permita que o desenvolvimento seja simples, com vista a um custo suportável pelos clientes alvo.

De forma detalhada, os objectivos que pretendemos atingir agrupam-se em quatro ideias principais:

- ➔ Criar uma base de conhecimento sobre a Produção Magra: Princípios, métodos e ferramentas.

  Ter bom conhecimento sobre o tema sobre o qual vamos trabalhar, embora seja vasto, é importante saber que dados importantes serem guardados ou que ferramentas são imprescindíveis de implementar.

- ➔ Verificar como é que são aplicados, na prática, os conceitos de gestão e o modo como são usados os conhecimentos obtidos sobre a produção.

  Saber como é que se passa da teoria à prática, ou seja neste caso perceber como é que a informação recolhida deve ser tratada e apresentada aos utilizadores do sistema.

- ➔ Criar uma base de conhecimento sobre ferramentas já presentes no mercado e que são as possíveis concorrentes.

  Como em todos os tipos de negócio, sempre que se pretende lançar um novo produto ou serviço, saber o que já existe no mercado e quais os principais concorrentes é importante.

- ➔ Implementação de uma solução adequada à realidade de uma pequena empresa, com possibilidade de expansão e integração com ferramentas externas.

  A implementação, tal como já referido, tem de ter em conta o mercado alvo que se pretende atingir. Para isso é necessário garantir que cumpra algumas características:

  - O seu desenvolvimento tenha baixo custo.
  - Seja suficientemente genérico para que possa satisfazer o maior número de possíveis clientes.
  - Possibilidade de integração com outras ferramentas
  - Seja uma base escalável de forma a permitir desenvolvimentos à medida, caso seja pretendido.

1.3. Desenvolvimento do Trabalho

Este trabalho tem, como objectivo, a criação de uma base para um software de gestão de produção magra, possibilitando o armazenamento de dados necessários à aplicação das várias ferramentas que a mesma (produção magra) possui.

Perguntar-se-á então quais os dados a guardar e como os guardar? Como são recolhidos? Como são "lidos"? Que tarefas tem a solução de executar?

Para conseguir responder a estas questões o desenvolvimento vai ter três fases distintas:

- **Pesquisa bibliográfica sobre o Produção Magra:**

  Numa primeira fase tem de se compreender os fundamentos, as metodologias e as ferramentas da produção magra, fazendo o registo da mesma de forma condensada e simples.

  Numa segunda fase é importante conseguir perceber como é que a teoria é posta em prática fazendo dessa forma um levantamento dos elementos que teremos de guardar para depois permitir fazer uso das ferramentas Lean.

- **Fazer levantamento de soluções existentes no mercado**

  Para se puder desenvolver uma solução que, por um lado cumpra os objectivos propostos e por outro se diferencie do que neste momento existe no mercado é importante saber o que existe no mercado e quais as suas principais características.

- **Modelar a solução:**

  Por fim é necessário analisar todos os elementos conseguidos e criar uma lista de requisitos essenciais que permitam cumprir os objectivos propostos e a partir daqui proceder ao processo de modelação, elaborando os casos de utilização, diagrama de classes e a estrutura de dados necessários.

  Definição da implementação do sistema, tendo em conta o que foi modelado e os requisitos especificados.

  Por fim, elaboração de uma proposta de interface de gestão dos diversos elementos do sistema, visualização de dados e uso das ferramentas implementadas.

2. Produção Magra (Lean Manufacturing)

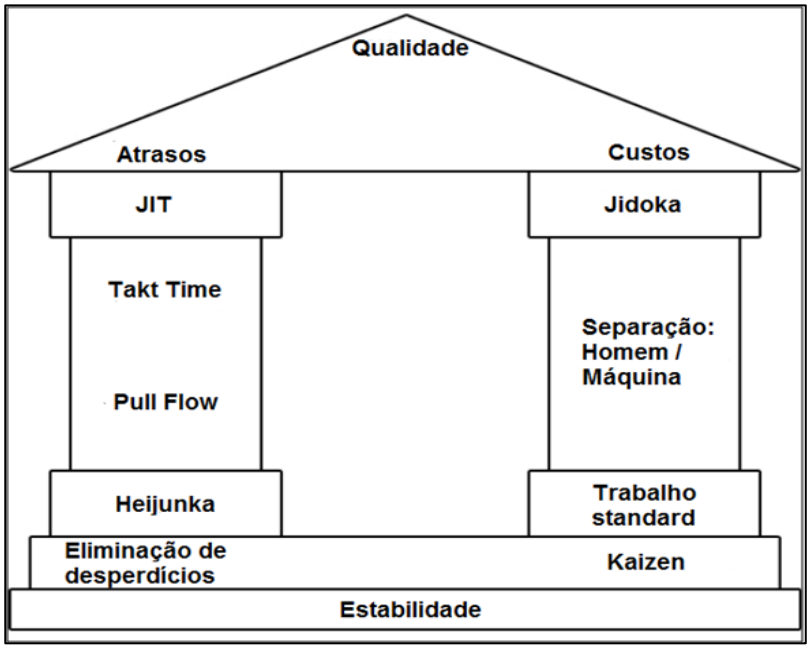


Imagem 1 – Casa do Sistema de Produção Magra

"O monumento lean é o símbolo utilizado pelos seus fundadores para explicar a coerência e a harmonia do sistema.
Na sua fundação está a estabilidade. Esta consegue-se, por exemplo, através de estabilidade de equipas, normalização de métodos, etc.
O edifício possui dois pilares que estão suportados pela dinâmica kaizen (melhoria continua) e pela eliminação de desperdícios.
Os 2 pilares do monumento Lean JIT e JIDOKA estão assentes em:

Heijunka: Alisamento – sequenciamento da produção

Trabalho standard: uma variabilidade reduzida do ritmo e dos processos de trabalho: um sistema destinado a absorver tanto quanto possível a irregularidade da procura.

As ferramentas utilizadas nas paredes para sustentar o são:

Para o pilar JIT: Pull Flow, Takt time e fluxo contínuo.

Para o pilar Jidoka: Separação homem – máquina (um operador gere várias máquinas ou Autonomação

O tecto, ou objectivo do método, é resumido por CQD, baixa de custos de produção, melhoria do nível de qualidade, adaptação dos prazos dos Processos às necessidades do cliente. " **(Marques, Lean Manufacturing)**

A utilização das metodologias lean provoca uma redução dos desperdícios e a implementação de comportamentos Kaizen, a melhoria contínua.

"O pensamento Lean consiste num conjunto de conceitos e de princípios, que têm como objectivo simplificar o modo como uma organização produz valor para os seus clientes, enquanto todos os desperdícios são eliminados.

- **Valor -** A organização deverá fornecer o valor que o cliente realmente deseja.
- **Cadeia de Valor -** Identifica as etapas que não acrescentam valor.
- **Fluxo -** Criação de um fluxo contínuo
- **Pull -** Produzir apenas o necessário, quando for necessário
- **Perfeição -** Completa eliminação do desperdício. Só as actividades que acrescentam valor estão presentes nos processos." **(Sousa, 2008)**

A Produção Magra traz diversos benefícios, tais como crescimento do negócio, aumento da produtividade, redução de custos, aumento do nível de serviço (ex: cumprimento de requisitos e de pedidos, entregas nas datas acordadas), redução do espaço, aumento da capacidade de resposta, redução do *Lead Time*.

2.1. Estratégias de Base

O combate e as estratégias estão intimamente ligados. A estratégia permite poupar forças durante a batalha para, dessa forma, assegurar a vitória. No caso de empresa com problemas com os seus oponentes internos e externos, a estratégia é definida como um plano de acção, que transforma um problema num factor de sucesso. A estratégia descreve a linha de conduta, que permite achar a solução para uma tarefa essencial a toda a empresa.

As seis estratégias do regime de poupança representam modelos de soluções para as tarefas «internas» essenciais à empresa, designadamente, **produção** que esteja de acordo com as condições do mercado, com a qualidade pretendida pelos clientes e com duração adaptada à concorrência, **ajustamento** ao mercado e às necessidades dos clientes, **remuneração** dos capitais e, por fim, **uma certa concepção** das relações com o pessoal e parceiros de mercado. Essas seis estratégias são:

- ✓ **Marketing de Prospecção**

O cliente leal é o melhor cliente. A sua satisfação deve ser medida e planificada, já que faz com que volte de novo. É necessário aconselhar os clientes com competência e honestidade.

Não é uma função de distribuição, mas sim uma missão da empresa. O contacto entre clientes e empresa deverá ser directo, pois é necessário o seu envolvimento.

É essencial o espírito do cliente e não o espírito do produto ou da produção. Todas as normas da empresa são fixadas de acordo com o ponto de vista do cliente e o desenvolvimento é feito em conformidade com as suas expectativas.

- ✓ **Fluxo limitado das matérias/ (na hora, Kanban)**

Estes fluxos não exigem stocks. Assim, o investimento é menor, porque é preciso menos espaço, menos dinheiro em matéria-prima e menos gastos em despesas administrativas.

Permite reagir rapidamente às necessidades dos clientes e às evoluções do mercado, proporcionando melhores preços e, em consequência, fidelidade do cliente. Deixa de existir liquidação de stocks ou saldos. A detecção dos defeitos é feita durante a fase de fabrico e, por isso, vão existir menos devoluções e retoques. Apenas se produz sob encomenda e os lotes apresentam uma dimensão, a mais reduzida possível.

- ✓ **Gestão da qualidade total/ TQM (*Total Quality Management*)**

É um factor estratégico essencial do êxito, logo uma tarefa essencial da empresa. É dever de cada um e não só de especialistas. Cada responsável deverá praticar e responder pela qualidade total do seu trabalho.

O ponto de vista do cliente também é tido em conta. É ele que determina os critérios de qualidade, que podem ser compreendidos como factores objectivos e subjectivos.

Tenta prevenir cada vez mais a eliminação dos defeitos, pois a automatização dos procedimentos substitui a escolha do controlo final e os estudos detectam e eliminam, à partida, as eventuais causas dos defeitos.

- ✓ **Desenvolvimento dos Produtos/ SE (*Simultaneous Engineering*)**

A redução do atraso no desenvolvimento é um factor decisivo de competição, ou seja, melhores preços, melhores oportunidades para a liderança sobre o mercado e economia de tempo, através da execução de tarefas num sistema paralelo e não em etapas sucessivas.

Trabalho preciso com dados imprecisos, ou seja, substituir os dados de entrada imprecisos ou em falta, por séries de medidas fixadas de comum acordo, conforme vão sendo mais necessárias.

- ✓ **Aplicação Estratégica dos Capitais**

As operações correntes devem ser económicas para libertar os capitais destinados aos projectos estratégicos, que não devem ser obrigatoriamente amortizados a curto prazo, de modo a permitir constituir uma plataforma sólida para o desenvolvimento da empresa. O investimento dirige-se igualmente aos bens incorpóreos, tais como a perícia do pessoal e a imagem de marca.

Permite aplicar uma economia de necessidades em capitais na fase crescente, diminuindo o activo circulante, através de um fabrico com stock zero e reduzindo os investimentos pelo crescimento da produtividade.

Confiança em vez de desconfiança entre o subscritor de capitais e o utilizador de capitais. A confiança coloca os capitais em condições vantajosas.

- ✓ **A empresa é uma família**

Os conflitos são caros e custosos, pelo que devem evitar-se, através de uma verdadeira colaboração, a fim de se obter uma sociedade de confiança e não de desconfiança.

Integrar com dinamismo a empresa no ambiente social e industrial com a plena utilização dos recursos dos fornecedores, clientes, colaboradores e subscritores de capitais. Os clientes fiéis são os mais preciosos. Integrando-os na empresa, aumentará a sua fidelidade.

(Bösenberg, 2009)

2.2. Eliminação de desperdícios (mudas)

A eliminação de desperdícios ou *mudas* é o aspecto mais importante e que está na base do Lean.

Contudo a eliminação de desperdícios deve ser um compromisso a longo termo. Isto porque, a eliminação em si até pode ser uma tarefa que se concretize numa semana, mas a implementação dos mecanismos que permitem que a mesma situação não se volte a repetir demoram muito mais tempo.

A Toyota identificou sete tipos mais visíveis de acções que não acrescentam valor no negócio ou no processo de manufactura e que são descritos a seguir:
(Liker, 2005)

2.2.1. Muda Processamento

A utilização de pequenos contentores em abastecimento frontal permite a redução do tamanho da linha, fonte de economia de despesas gerais e de redução dos custos e dos tempos de escoamento. **(Marques, Lean Manufacturing)**

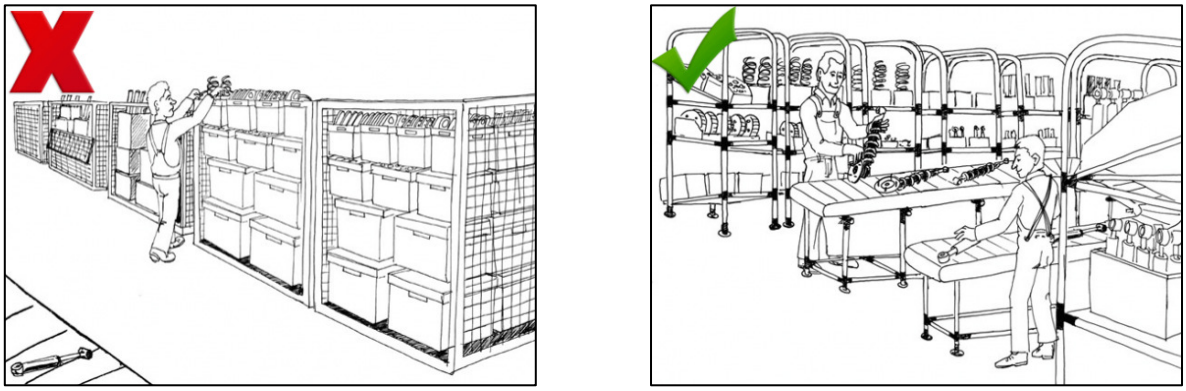


Imagem 2 – Exemplo Muda Processamento **(Marques, Lean Manufacturing)**

2.2.2. Muda Defeitos

A melhor maneira de eliminar os defeitos é, exactamente, não os criar nenhum. Estes geram custos e perdas de tempo para a empresa, já que é necessário um sistema para poder refazer o trabalho. A eliminação dos defeitos é, por vezes, ainda mais dispendiosa.

Para o efeito é necessário criar um ambiente de trabalho adequado e uma ergonomia adaptada: peças e ferramentas nos seus lugares, ao alcance imediato para as operações. Assim, reduzem-se os riscos de choques, de quedas e de defeitos. **(Marques, Lean Manufacturing)**

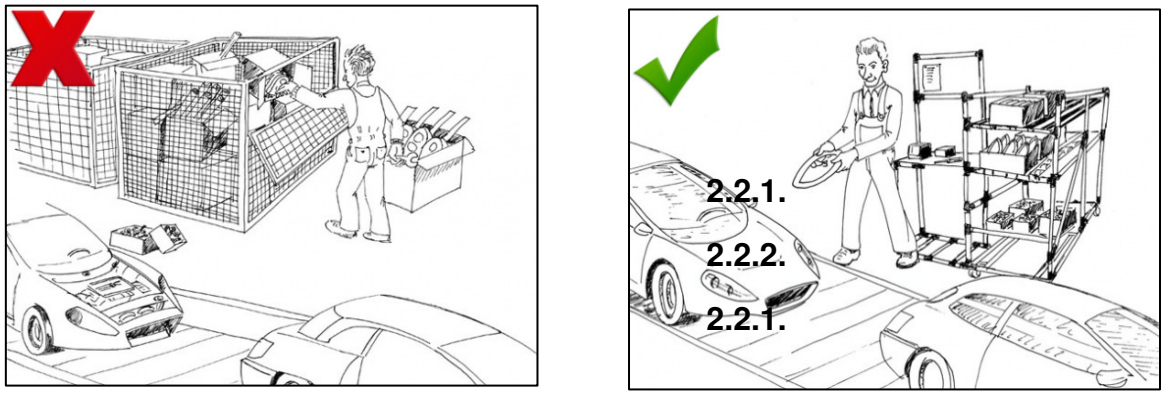


Imagem 3 – Exemplo Muda Defeitos **(Marques, Lean Manufacturing)**

2.2.3. Muda Deslocamento

Os deslocamentos e movimentos inúteis no posto de trabalho não criam nenhum valor acrescentado. Pelo contrário, aumentam a dificuldade do trabalho e consomem espaço. Um sistema Lean de arquitectura modular permite a configuração dos postos de trabalho, de maneira a permitir a colocação de peças o mais próximo possível da mão do operador. Isso contribui para a redução do valor não acrescentado, gerado pelos deslocamentos inúteis. Assim, a produtividade do operador é aumentada e as dificuldades de trabalho reduzidas: a actividade do operador é concentrada nas tarefas produtivas. **(Marques, Lean Manufacturing)**

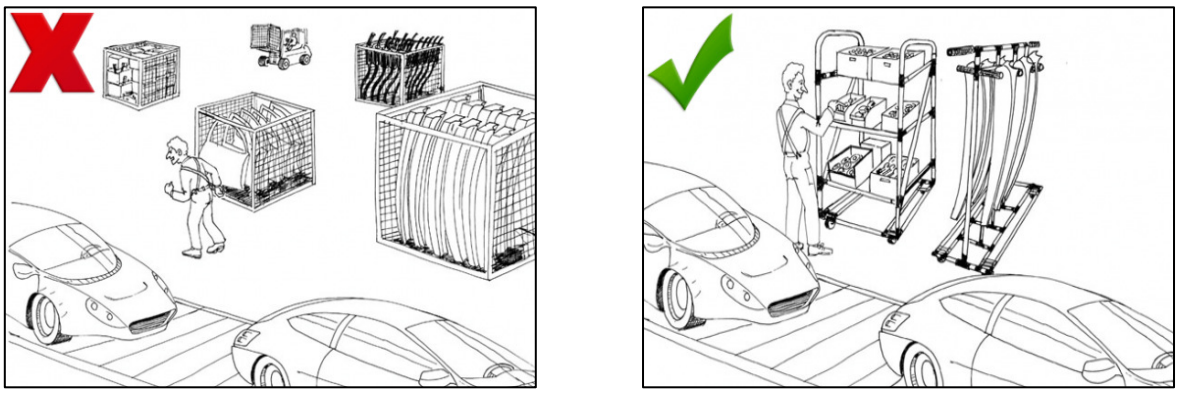


Imagem 4 – Exemplo de Muda de Deslocamento **(Marques, Lean Manufacturing)**

2.2.4. Muda de Stocks

Os produtos acabados, semi-acabados e matérias-primas não criam nenhum valor agregado. Ao contrário, os stocks excessivos aumentam os custos devido à disponibilidade de espaço e aos investimentos necessários para sua movimentação.
Este *Muda* está ligado ao *Muda* de superprodução. A utilização de um sistema Lean associado a pequenos acondicionamentos dos produtos e ao aumento da frequência das entregas, permite a redução dos stocks. Esta é possível com a instalação de estantes dinâmicas, idênticas às usadas nos supermercados, localizadas o mais próximo possível da linha de produção. **(Marques, Lean Manufacturing)**

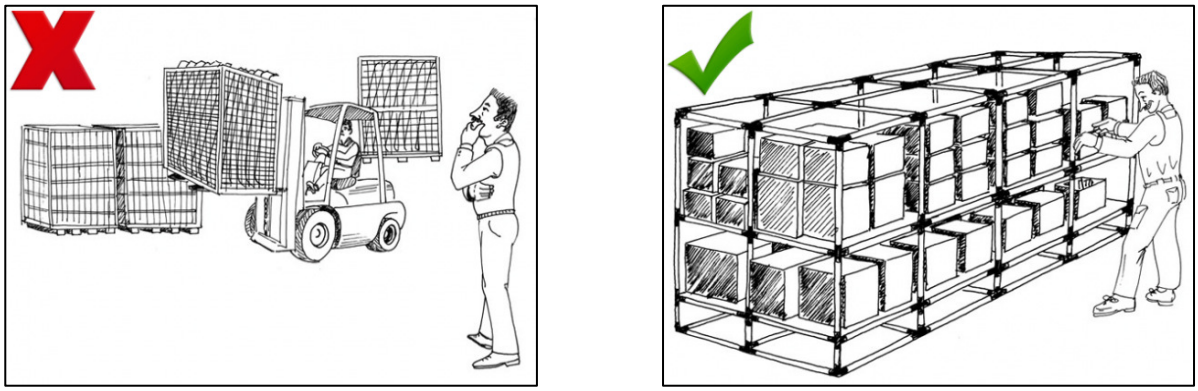


Imagem 5 – Exemplo Muda de Stock **(Marques, Lean Manufacturing)**

2.2.5. Muda de Espera

Este *Muda* acontece quando o operador deixa de ter à sua disposição as peças necessárias para a execução do trabalho: as mãos ficam desocupadas.

A utilização de um *rack* próprio, na margem da linha, com pequenas embalagens, diminui o risco de interrupção do abastecimento. Isto apenas é possível através da implementação de uma logística fundada num fluxo contínuo e abastecimentos regulares. Os operadores podem concentrar-se nas operações de valor acrescentado, enquanto a logística abastece as peças em pequenos comboios. **(Marques, Lean Manufacturing)**

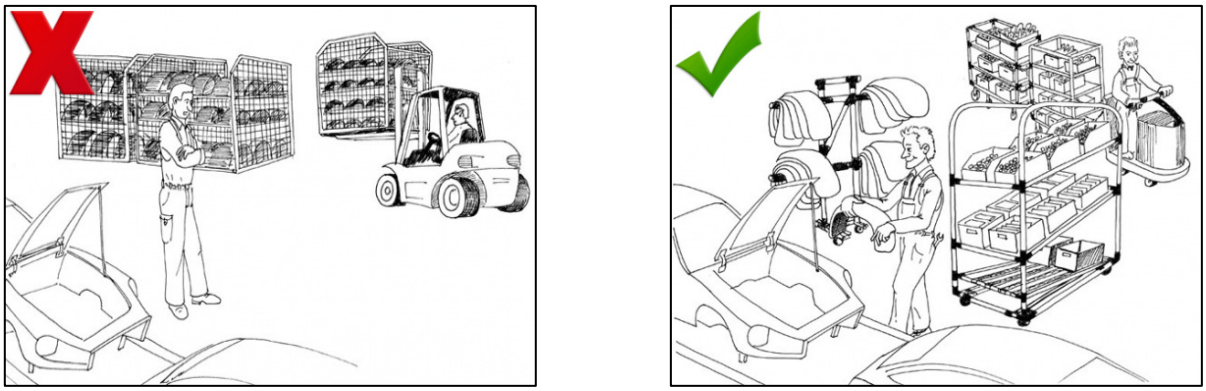


Imagem 6 – Exemplo Muda de Espera **(Marques, Lean Manufacturing)**

2.2.6. Muda de Transporte

O deslocamento de produtos não gera valor acrescentado, pelo contrário, os transportes consomem espaço e capitais.

Numa configuração *lean* os circuitos logísticos devem ser o mais curto possível, entre a plataforma e as prateleiras e também entre as prateleiras e a margem da linha. Isto só é possível usando uma logística baseada em comboios de abastecimento flexíveis, que permitem distribuir diversas vezes e numa só passagem, os componentes necessários à produção. **(Marques, Lean Manufacturing)**

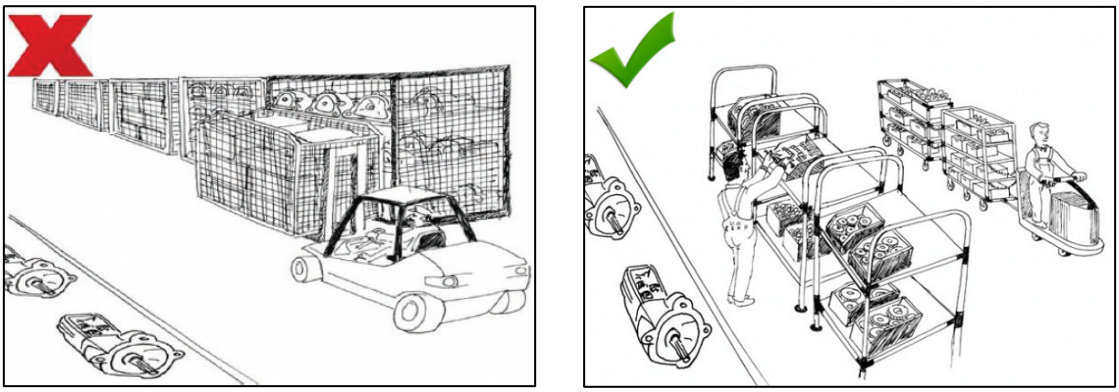


Imagem 7 – Exemplo Muda de Transporte **(Marques, Lean Manufacturing)**

2.2.7. Muda de Superprodução

A implementação de um sistema Kanban permite lutar contra os desperdícios ligados à superprodução, que acontece quando se continua a produzir mesmo depois de satisfeita a ordem de fabrico. **(Marques, Lean Manufacturing)**

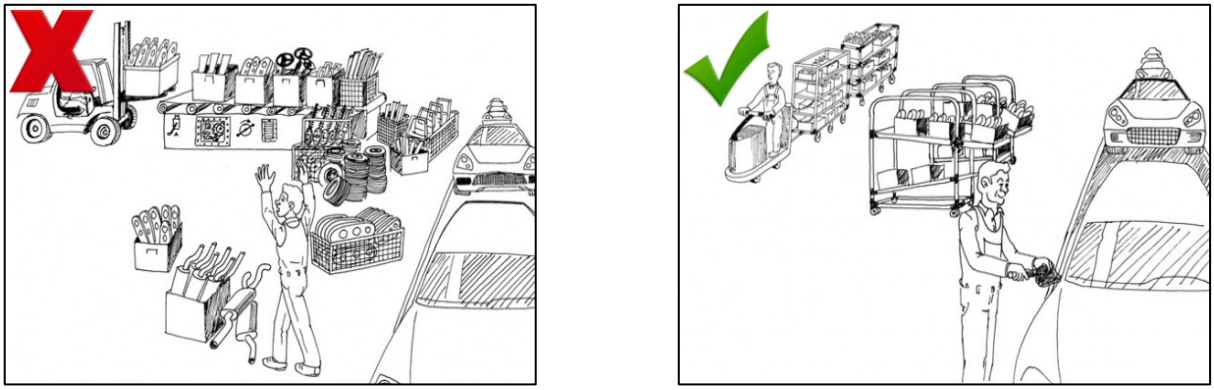


Imagem 8 – Exemplo Muda de Superprodução **(Marques, Lean Manufacturing)**

2.3. Criar estabilidade

Desenvolver estabilidade é uma forma de criar uma fundação para se poderem aplicar outras ferramentas lean. Através de observação directa, um processo instável é indicado pelas condições:

→ Um grau elevado de diferença na medição do desempenho – quer pelo número de peças ou peças por hora de trabalho.

→ Mudança do plano de produção, frequentemente, sempre que surja um problema.

→Não é possível observar um padrão consistente ou um método de trabalho.

→ Volume de trabalho em processo (*Work In Process* - WIP) muito variável.

→ Operações sequenciais que operam independentemente (processo ilha)

→ Fluxo inconsistente ou inexistente, que também pode ser indicado pela variação do volume de trabalho.

→ Na descrição de uma operação o uso frequente das palavras *usualmente, basicamente, normalmente, tipicamente, geralmente*, quase sempre seguidas da expressão *excepto quando*. Exemplo:
"*Normalmente* nós fazemos …, *excepto quando* acontece … que fazemos …"
De forma simples, pode-se dizer que implica a previsibilidade geral e disponibilidade constante em relação à mão-de-obra, máquinas, materiais e métodos — Os 4M's. O motivo é simples. Sem itens fundamentais, como disponibilidade das máquinas e recursos humanos adequados, não se pode operar uma linha de produção e obter fluxo perfeito em ritmo, de acordo com o planeado.

Para ser possível avaliar se a estabilidade é suficiente, é necessário responder a alguns requisitos chave:

- Há suficiente disponibilidade das máquinas para produzir, de acordo com o pedido dos clientes?
- Há material suficiente para todos os dias cumprir a produção?
- Há suficiente quantidade de funcionários qualificados para lidar com os processos actuais?
- Há métodos de trabalho implementados, tais como instruções básicas, definidas ou padrões estabelecidos?

Tentar responder aos pedidos dos clientes com operários despreparados, supervisão frágil ou poucas peças no lugar certo, será traçar o caminho para o desastre.**(Liker, 2005)**

Para se alcançar estabilidade, deve dar-se atenção aos quatro elementos chave, correspondentes ao 4M's:

❖ Mão-de-Obra

A estabilidade começa com mão-de-obra bem treinada. Felizmente, os funcionários tendem a conhecer o trabalho muito bem. A diferença pode ser feita mais na supervisão, melhoria das técnicas e habilidades dos grupos de trabalho.

A Toyota, nos anos 50, adoptou um programa de formação industrial que os E.U.A. usaram durante a Segunda Guerra Mundial: *Training Within Industry* (TWI). Este possuía três componentes: Instrução de Trabalho, Métodos de Trabalho e Relações no Trabalho. Cada componente era um curso de dez horas, que ensinava habilidades práticas de supervisão.

A "Instrução de Trabalho" mostrava aos supervisores como planear os correctos recursos necessários à produção, desdobrar as tarefas e ensinar os funcionários de maneira segura, correcta e consciente.

Os "Métodos de Trabalho" ensinavam como analisar tarefas e fazer melhorias simples, dentro do seu domínio de controlo. Cada e toda actividade era considerada para planear a melhoria. Os supervisores aprenderam a questionar porque é que uma actividade era feita de determinada maneira e se era possível ser eliminada, combinada com outra, reorganizada ou simplificada.

As "Relações no Trabalho" ensinavam aos supervisores a forma de tratar as pessoas como indivíduos e a resolver problemas comuns de relacionamento de trabalho, ao invés de ignorá-los.

Estas formações ajudaram os supervisores a criar rotinas básicas, disciplina e senso de justiça nos grupos de trabalho.

❖ Máquinas

Não são precisos equipamentos com disponibilidades perfeitas, mas é necessário conhecer os pedidos de clientes, a capacidade do processo e a média real de produção.

Para medir o verdadeiro potencial de produção de um processo durante um turno típico, a Toyota usa um documento básico chamado Folha de Capacidade de Processo. Se há capacidade teórica, assim como capacidade demonstrada para satisfazer os pedidos, então não há problemas.

A instabilidade relacionada com as máquinas, surge apenas quando não há capacidade de resposta para os clientes. Por exemplo, se existe um fluxo de encomendas de 700 unidades por turno e a sua produção real é somente de 500 unidades, apesar de ter capacidade para 1000 unidades, então é necessário maior disponibilidade.

❖ Materiais

Em geral, o objectivo é reduzir o desperdício e diminuir o tempo compreendido entre a recepção de um pedido e a respectiva entrega. Normalmente, isto requer a redução dos stocks no fluxo de valor, mas, em certos casos, é necessária a existência de stocks.

O motivo é que, para um processo em grandes lotes, é necessária uma quantidade de peças em stock, por forma a preencher o tempo em que outras peças estão a rodar ou ferramentas estão a ser trocadas. Este stock necessário é conhecido como "stock do ciclo" (quantidade de peças para cobrir uma pedido médio e o tempo de accionar o reabastecimento) ou "stock pulmão" (peças para cobrir variações de fluxo acima ou do cliente) e pelo "stock de segurança" (peças para cobrir perdas como refugos ou tempos de máquinas paradas, que ocorrem naturalmente). Falhas em estabelecer estes num ambiente instável, irá prejudicar a eficiência da linha de produção.

Numa forma resumida, nem todo o stock é desperdício, só aquele que vai além do necessário. Frequentemente o stock existe como sintoma de um problema no processo (resolvendo os problemas, já se poderá reduzir o stock).

❖ Métodos

Finalmente, para se obter estabilidade, são necessários, na manufactura, métodos "standard" ou "padrão". Aqui, o ponto-chave é a definição do que é um "padrão". A definição normal de padrão é uma regra ou método de realização dos assuntos. O efeito colateral involuntário é que as pessoas não são encorajadas a questionar ou mudar as regras.

Na Toyota, a definição de padrão é ligeiramente diferente. " Padrão" é uma "regra ou base para comparação" e funciona como ferramenta de medida e referência, quando se deseja mudar algo.

O pensamento *lean,* consiste em mudar os métodos de trabalho para eliminar os desperdícios e fazer melhorias. Os padrões devem ser ferramenta de comparação para verificar se as alterações foram eficazes.

(Tadashi*, 2010*)

2.4. Gestão da Qualidade que favoreça a melhoria contínua e a melhoria renovadora

Paralelamente às evoluções observadas na gestão industrial, a qualidade também evoluiu muito nas últimas décadas. No inicio a sua principal tarefa era o controlo da conformidade dos produtos, até que surgiu o interesse pela organização da estrutura da empresa, a fim de transmitir confiança aos clientes.

O papel da função de qualidade ultrapassa a mera qualidade do produto e passa também a interessar-se pelo desempenho da empresa, A empresa não deve limitar-se ao objectivo da conformidade do produto, mas antes tender para uma dinâmica de progresso através de várias áreas-chave.

A introdução do método *Six Sigma* traduz, em parte, evolução no sentido de uma mudança de ritmo no processo de melhoria da empresa. Procurou-se a melhoria inovadora em vez da melhoria permanente. De facto, a melhoria contínua é necessária, mas as lógicas utilizadas não permitem o faseamento. Para isso, é imprescindível pôr em causa aspectos fundamentais, é preciso recriar de raiz o processo ou o produto.

O método de resolução de problemas está estruturado em cinco etapas:

- ✓ **Definir.** Definição formal dos problemas, das oportunidades, dos objectivos, das reduções de custo e dos processos envolvidos
- ✓ **Medir.** Obter os dados iniciais ("baseline") do processo focado.
- ✓ **Analisar.** Determinar a relação entre causas e efeitos raiz.
- ✓ **Melhorar.** Propor, testar e implementar melhorias;
- ✓ **Controlar.** Controlar as causas - raiz críticas identificadas e monitorizar os seus efeitos

(Marques, JIT, Lean Management, Six Sigma)

2.5. Ferramentas/Técnicas *Lean Management*

Imagem 9 – Ferramentas/Técnicas Lean Management **(Sebrosa, 2008)**

2.4.1. TPM – Manutenção Produtiva Total

Conjunto de estratégias para minimizar o tempo de indisponibilidade das máquinas. A manutenção das máquinas e ferramentas é feita de uma forma pró-activa, de modo a manter o funcionamento com eficiência e com tempo de paragem reduzido.

Há cinco aspectos fundamentais que se combatem com o uso do TPM:

Avaria Inesperada – A paragem do equipamento traduz-se em perdas de tempo de produção e de mão-de-obra.

Configuração e Ajustes de Equipamento – Perdas de tempo de produção, durante a mudança para a produção de novo produto.

Paragem do Equipamento – Normalmente difíceis de serem registadas, estas paragens, que podem ir até cerca de 10 minutos de duração, são "escondidas" dos relatórios de eficiência, por serem integradas na capacidade produtiva das máquinas. Porém, no somatório causam perda de capacidade de produção.

Velocidade de Produção – Quando se pretende prevenir defeitos de fabrico, tem de reduzir-se a velocidade dos equipamentos. Estas situações, por norma, nem são registadas, dado que a máquina continua a operar.

Defeitos na Qualidade – Defeitos no produto final levam a que se tenha de proceder à respectiva correcção, o que implica quebra no ritmo de produção e representa mais do que a duplicação de esforço de trabalho ou, no pior dos casos, perda total do produto.

Tipicamente as empresas recorrem a quatro técnicas para a implementação do TPM:

Equipamento Eficiente – A melhor maneira de aumentar a eficácia de um equipamento é identificar as perdas, que interferem na performance. De forma a poder calcular-se a eficiência, é usado a *Overall Equipment Effectiveness* (OEE), que permite identificar onde deve ser focada a atenção, para melhorar.

Manutenção Eficiente – As rotinas de manutenção são um aspecto crítico do TPM. Os operadores devem ser treinados de forma a sentirem-se responsáveis pelas rotinas de manutenção da "sua" máquina e dos "seus" equipamentos. A manutenção autónoma desempenhada pelo trabalhador, poderá ir ainda mais longe, com correcções sugeridas pelos próprios, que permitam melhorar.

"À Prova de Erro" – Conhecido como *Poka-Yoke*, consiste na implementação de mecanismos simples contra enganos, desenhados para tornar o erro "impossível" ou, pelo menos, detectá-lo. (Ver – Qualidade na Origem).

Gestão de Segurança – O princípio fundamental da segurança no TPM e na gestão ambiental é o de eliminar condições e actividades potencialmente perigosas e que podem trazer custos inesperados. De forma a evitar mau funcionamento é comum o uso de metodologias como, por exemplo, criar uma lista de segurança para verificar a normalização das operações, a coordenação de tarefas não repetitivas ou a modificação de equipamentos.

2.4.2. Qualidade na Origem

Uma filosofia de qualidade, cujo objectivo é a responsabilidade de alcançar as especificações do cliente e as normas em cada ponto de produção. A ideia é corrigir os erros o mais cedo possível (o mais a montante possível), incluindo a fase de concepção e projecto. Quanto mais tarde se detectar o defeito, maior o desperdício.

Esta ferramenta é composta por três componentes:

- Inspecção na fonte para os materiais e componentes adquiridos;
- Os operadores auto-inspeccionam o seu trabalho, bem como o produto resultante dos operadores / processos anteriores (se não conforme, rejeitam);
- Instalação de dispositivos *Poka-Yoke* (á prova de erro) nos processos produtivos e nos próprios produtos.

Enquanto **os dois primeiros componentes** são de inspecção, *já que se verifica se o material de trabalho está em condições*, **os dispositivos Poka-yoke** dividem-se em duas categorias:

- Prevenção: um dispositivo de prevenção torna o erro impossível.
- Detecção: um dispositivo de detecção sinaliza a ocorrência de um erro, para que possa ser corrigido o mais rapidamente possível.

2.4.3. Os 5 S

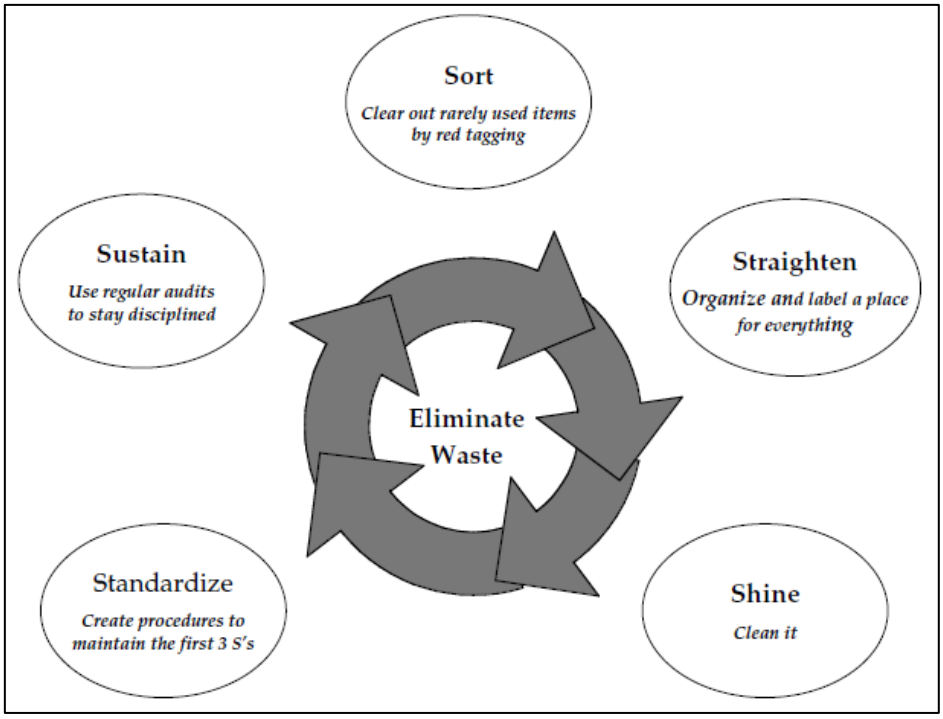


Imagem 10 – Os 5 S's **(Jeffrey, 2005)**

O correcto uso desta metodologia resulta num ambiente de trabalho bem organizado, com controlos visuais e com ordem.
Um espaço de trabalho limpo, seguro e organizado motiva os colaboradores, levando-os a sentirem-se comprometidos e com espírito de trabalho.
Desenvolvido no Japão, baseia-se em cinco etapas com designações começadas pela letra **S**:

1º *SEIRI* – Separar/Segregar – "é um método para libertar espaço nos locais de trabalho, eliminando objectos desnecessários tais como ferramentas obsoletas, recipientes inúteis, resíduos, excessos de matérias-primas, documentos/papel, dados e ficheiros informáticos" **(Silva, TPM – Formação Básica Pilar 1)**

2º *SEITON* – Arrumar/Organizar – "consiste em organizar os objectos, ferramentas e dados de forma racional, permitindo facilidade de fluxo de

pessoas e utilização dos mesmos com rapidez e segurança, a qualquer momento" **(Silva, TPM – Formação Básica Pilar 1)**

3º *SEISO* – Limpar – "Consiste em manter os locais de trabalho isentos de sujidades proporcionando um ambiente agradável, desenvolvendo a consciência de que o importante não é só limpar, mas sim eliminar as fontes de contaminação." **(Silva, TPM – Formação Básica Pilar 1)**

4º *SEIKETSU* – Normalizar – "Consiste em garantir a sistematização das actividades dos 3 primeiros S's e a melhoria da saúde a nível físico, mental e emocional." **(Silva, TPM – Formação Básica Pilar 1)**

5º *SHITSUKE* – Respeitar/Disciplinar – "Consiste na prática dos S's anteriores, procurando o seu constante aperfeiçoamento. É a procura do auto desenvolvimento e da melhoria contínua" **(Silva, TPM – Formação Básica Pilar 1)**

2.4.4. Gestão Visual

Pode-se definir Gestão Visual como um sistema que integra um conjunto de ferramentas visuais simples, que possibilitam, de forma rápida, verificar o estado da empresa, identificar os problemas e pensar estratégias para melhorar.

A forma como a informação é exposta, deve ser de modo a que sua interpretação não levante quaisquer dúvidas. Ao consultar, todos devem entender da mesma maneira.

Este tipo de metodologia facilita a comunicação entre equipas de trabalho, através de uma informação clara e fácil de interpretar. Permite uma resposta rápida a anomalias e diminuição de erros. Permite ainda uma maior autonomia dos operadores e do pessoal de manutenção.

2.4.5. Trabalho Padronizado

Traduz-se no processo de documentar e normalizar as tarefas ao longo da cadeia de valor. São normalmente de dois tipos: instruções do processo e procedimentos operacionais normalizados.

A maior parte das empresas não possui procedimentos documentados para operar os equipamentos e para produzir os produtos. O resultado é uma elevada variabilidade de produtos, custos elevados, paragens por falta do empregado " que sabe como se faz " e constantes incumprimentos dos planos de produção.

Através da padronização do trabalho em toda a fábrica, os produtos conseguem ser produzidos com qualidade e características constantes (menor variabilidade), devido a formas de proceder idênticas, independentemente de quem seja operador.

Os operadores aprendem mais fácil e correctamente novas tarefas, conseguindo substituir-se uns aos outros.

- Benefícios:

- Aumenta a eficácia da formação e treino;
- Suporta a melhoria dos processos e produtos;
- Reduz a variabilidade dos produtos;
- Reduz os custos do treino de novos empregados.

2.4.6. SMED – Redução do Setup

Do inglês *Single-Minute Exchange of Die* é um dos métodos da produção lean para redução do desperdício. Foi desenvolvido no Japão por Shigeo Shingo e tem provado a sua eficácia em muitas companhias, reduzindo o tempo de mudança de horas para minutos.

Neste método, o processo de mudança é separado em duas operações chave:

External Setup – Instalação e Configuração Externa – inclui as operações que podem ser realizadas, enquanto a máquina continua a operar.

Internal Setup – Instalação e Configuração Interna – inclui as operações que não podem ocorrer, enquanto a máquina está em funcionamento.

A forma de implementação depende muito de máquina para máquina e de produto para produto, mas existem ideias chave que ajudam:

Eliminar operações não essenciais - Ajustar e mudar apenas o que for essencial, tentando tornar todo o resto o mais universal possível.

Executar instalação e configurações externas – Organizar o processo de mudança, de modo a que se consiga fazer o máximo possível antes de se parar a máquina.

Simplificar a instalação e configuração interna – utilizar metodologias para reduzir os ajustes, substituir porcas e parafusos por botões, alavancas e pinças de alternância, etc.

Medir, Medir, Medir – Apenas medindo se pode verificar a eficácia das alterações efectuadas e possíveis desperdícios ainda existentes, que possam igualmente ser eliminados.

2.4.7. JIT – Just-In-Time

É a peça fundamental na produção Lean. No iníco o sistema de produção da Toyota era conhecido por JIT, precisamente pela extrema sua importância.

O JIT divide-se em três partes principais:

JIT Purchasing – Fornecimento JIT – O principal objectivo passa por ter a certeza de que se tem o material necessário, apenas quando é preciso. Aqui, os pedidos e os acordos com os fornecedores são limitados em quantidade, assegurando também a respectiva qualidade.

JIT Manufacturing – Manufactura JIT – Produção dos bens apenas quando são necessários. Esta necessidade tem origem nos pedidos dos clientes ou na própria produção da fábrica.

JIT Distributing – Os bens em pequenos lotes. Com a redução da dimensão dos lotes, poderá acontecer que vários sejam enviados para o mesmo cliente ou até mesmo para o mesmo sítio. O objectivo é facilitar o combate aos períodos de paragem do trabalhador (não haver o que lhe dar a fazer) e proporcionar o combate ao conceito de grande armazenamento.

É visível que nestas três etapas uma das peças fundamentais é o controlo de stock ou a sua redução, ao máximo. Embora não seja este o único objectivo, são óbvios os benefícios que se retiram, nomeadamente, menos espaço necessário para o trabalho; aumento da área livre que permite reorganizar os espaços, levando à redução da distância percorrida pela material; menos controlo de inventário, o que significa colocar mão-de-obra e tempo de trabalho em tarefas que, na realidade, trazem valor ao produto.

2.4.8. Produção Celular (em fluxo continuo)

Na produção celular os postos de trabalho e os equipamentos são dispostos de forma a suportar um fluxo contínuo de materiais e componentes, ao longo do processo de produção, minimizando, ao máximo, os atrasos e os transportes.

O objectivo da produção celular na movimentação do produto é feita peça a peça e a um ritmo que está sempre ligado aos pedidos dos clientes. Esta metodologia permite responder a variações do produto, de acordo com eventual pedido do cliente, sem que influenciem muito o fluxo de produção.

Nesta metodologia de fluxo peça a peça, procura-se reduzir o tempo que um produto leva a percorrer o processo de produção.

A transformação de um sistema de produção tradicional para um sistema de produção celular, implica que o trabalhador deixe de controlar apenas uma máquina, para controlar diversas. Implica também a substituição de máquinas grandes e de elevado volume de produção por máquinas flexíveis e de tamanho certo, para se encaixarem na célula, bem como modificar as máquinas de forma a pararem e assinalarem, quando um ciclo está completo ou quando ocorrem problemas, usando técnicas de automação (*Jidoka*)

- **Benefícios:**
    - Melhor utilização dos recursos humanos;
    - Facilidade em automatizar;
    - Facilidade no controlo;
    - Trabalhadores capazes de desempenhar múltiplas tarefas;
    - Movimentação mínima dos materiais;
    - Redução do tempo de Setup e do inventário em curso de produção.

2.4.9. *Takt-time* – Balanceamento da produção

Deriva da palavra alemã *Takzeit*, que traduzida significa tempo de ciclo.

Esta metodologia tem como objectivo marcar o ritmo da produção. Por exemplo, numa fábrica de automóveis, um automóvel que foi montado numa linha só passará para a próxima etapa após um determinado tempo *takt-time.*

Inicialmente, o cálculo do *takt-time T* pode ser calculado pelo tempo necessário à produção, a dividir pelo tempo de satisfação do pedido do cliente:

$$T = \frac{T_a}{T_d}$$

Na realidade este cálculo simplista tem de ser ajustado, devido a diversos factores:

Todo o sistema de produção não funciona sempre a 100% da sua eficácia, existindo paragens e abrandamentos quer dos operários, quer das máquinas. Por diversas vezes torna-se necessário aumentar o ritmo, no sentido de precaver estes cenários.

No caso de um departamento fornecer diversas linhas de produção, é normal que nestas o *takt-time* seja similar, de forma a conseguir-se estabelecer um fluxo estável e contínuo.

Como qualquer metodologia tem os seus prós e contras:

Benefícios:
Facilita a detecção de estrangulamentos ao longo da linha de produção, quer porque certas estações necessitam de mais tempo para executar as tarefas, quer porque não operam de forma fiável.
O tempo para realizar as tarefas é limitado, o que implica que, normalmente, exista uma motivação para eliminar tarefas que não adicionem valor ao produto.

Deixa de ser necessário que homens e máquinas tenham de se adaptar diariamente a novos processos, dado que executam sempre tarefas similares.

Como os produtos têm um determinado tempo para permanecer na linha de produção, não correm risco de se perderem, algures, no piso de produção.

Contras:
Quando os pedidos dos clientes aumentam demasiado, o *takt-time* tem de diminuir e certas tarefas têm de ser reorganizadas, de forma a poder ser dada resposta.
Se um determinado posto pára por qualquer motivo, também toda a linha fica em suspenso, a não ser que existam stocks capazes de superar esta paragem.
No reinício da operacionalidade do posto o *tak-time* deve ser ajustado durante algum tempo, no sentido de repor o stock gasto.
Se o *takt-time* for demasiadamente curto, pode elevar os níveis de stress dos trabalhadores, aumentando as paragens e baixando a motivação.

2.4.10. Heijunka - Nivelamento e alisamento da produção

No contexto, *lean heijunka* representa as técnicas de nivelamento e alisamento da produção, equilibrando a carga de trabalho nas diferentes etapas do processo.

Ao contrário dos métodos de produção tradicionais, que empregam grandes lotes de produtos para montar, nesta metodologia tem-se em conta o volume de encomendas num determinado período, para, depois, poder distribuir-se a produção de maneira nivelada, usando lotes pequenos, de modo a evitar picos ou abrandamentos e também flexibilizar o processo de produção.

Para a implementação do *heijunka,* controlando a produção e produzindo volumes de produtos mistos, os gestores têm ao seu dispor a "Caixa Heijunka". Esta foi inventada na Toyota para simplificar a sua produção baseada no *heijunka*.

Uma caixa Heijunka típica é em forma de tabela, com uma linha por cada família de produto e as colunas são os intervalos de tempo idênticos de produção. As células são preenchidas com os cartões *kanban* de controlo, que representam as necessidades de produção num determinado intervalo de tempo.

Depois, em intervalos de tempo pré-definidos, um trabalhador responsável recolhe os *kanban* e entrega ao "marcador de passo", que irá processar os mesmos. Assim, os pedidos de produção são realizados de forma consistente e em pequenos intervalos de tempo e quantidade.

2.4.11. Sistemas " No local da utilização"

São sistemas físicos ou de organização de diversos tipos, que têm como objectivo eliminar várias formas de desperdícios, bem como aumentar a produtividade e qualidade.

O princípio é ter tudo o que for necessário para o desempenho da tarefa (ferramentas, dispositivos de fixação, normas de qualidade, ajudas visuais, etc.), o mais perto possível dos trabalhadores,

Por exemplo, dois postos de trabalho, posto A e posto B, usam as mesmas ferramentas, encontrando-se estas localizadas muito mais perto do A que do B. Para evitar deslocações dos trabalhadores, podem ser usadas duas técnicas: ou se duplicam as ferramentas, ou se juntam os postos lado a lado e se guardam as ferramentas em local perto de ambos.

2.4.12. Kanban – Sistemas de "puxar"

Embora a filosofia do lean seja a de ter um fluxo de produção sem a existência de stock, há casos de variações entre processos, nos quais é necessária a existência de um mínimo de stock, com a criação de supermercados, através do uso de *flow recks* e *kanban,* como sistema de sinalização de pedidos.
Kanban é uma expressão de origem japonesa, que significa registo ou placa visível. É usado como sinal de pedidos de material, que se propaga por toda a cadeia de fornecimento. Esta característica é aproveitada para assegurar melhor gestão, bem como menores quantidades de stock intermédio, presente na cadeia de produção.
Taiichi Ohno, mentor deste sistema, afirma que o seu uso, para ser eficaz, deve seguir regras restritas e que deve existir uma tarefa de monitorização permanente, de forma a assegurar que as mesmas sejam cumpridas.

Para além dos *Kanban* físicos, usualmente cartões de diversas cores ou caixas, existe também o uso dos *E-Kanban,* que muitas empresas têm implementado e que permite a interacção directa em todo sistema informático de gestão, reduzindo ainda mais os desperdícios.

2.4.13. *Kaizen* – Melhoria Contínua

Kaizen é uma palavra de origem japonesa, que, traduzida à letra, significa mudança para melhor. Já no contexto lean, representa um conjunto de práticas e ferramentas, que têm como objectivo a melhoria contínua.

Na filosofia *Kaizen* está sempre presente no "pensamento" de que em todo o tempo é possível fazer melhor. Hoje melhor que ontem e amanhã melhor que hoje. Esta metodologia traz resultados concretos, quer directamente no processo de produção, quer no ambiente de trabalho e no grau de dificuldade das tarefas.

Com o *kaizen* não se procura fazer grandes mudanças de uma só vez. Começa-se, sim, por realizar pequenas experiências, que depois, em caso de sucesso, serão integradas como melhorias.

A implementação do *Kaizen* é composta por cinco etapas, que se repetem constantemente, formando um ciclo:

- Normalizar o processo.
- Fazer medições para análise da normalização realizada.
- Confrontar as medições com os requisitos.

- Inovar para satisfazer os requisitos.
- Normalizar o novo processo.

2.4.14. Mapeamento de Fluxo de Valor

Mapeamento de Fluxo de Valor (VSM – Value Stream Mapping) é uma ferramenta visual que representa o fluxo de informação e de materiais entre processos.

O objectivo a obter no final, é criar um mapa com o mínimo de atrasos, enquanto se observa o processo e se pensa a maneira de o melhorar. Normalmente, estes mapas são desenhados à mão e a lápis, para manter o VSM simples e facilitar correcções. No entanto, já existem no mercado de software, ferramentas que possibilitam realizar as tarefas de forma fácil.

A implementação do VSM é composta por cinco etapas, que se repetem num ciclo infinito:

1. Identificar o produto, a família de produtos e serviços a analisar.
2. Desenhar um VSM do estado actual, que irá realçar os paços, os atrasos e o fluxo de informação do processo, para se obter o serviço ou produto final.
3. Avaliar o estado actual, pensando em criar ou melhorar o fluxo e eliminando desperdícios.
4. Desenhar um VSM do estado futuro.
5. Trabalhar para alcançar o que foi desenhado no ponto 4.

2.5. Lean no desenvolvimento de Software

Desde início, a indústria de desenvolvimento de software sentiu necessidade de implementar metodologias, para a respectiva produção (software). Estas cumprem a norma ISO 12207, que descreve o método de selecção, implementação e monitorização do ciclo de vida do software.

A procura e implementação de melhores metodologias têm, como objectivo, a procura de processos previsíveis e repetitivos, para melhorar a produtividade e a qualidade. Algumas tentam formalizar a tarefa desregrada de escrever código, outras aplicam técnicas de gestão de projectos, para a escrita de código.

O conceito de desenvolvimento de software Lean foi introduzido, pela primeira vez, por Mary e Tom Poppendieck, no livro *"Lean Software Development: An Agile Toolkit*" em 2003.

Este conceito está desenvolvido em Anexo, pois não é um tema directamente ligado ao objectivo proposto e pretendido para este trabalho.

Ver Anexo I

3. Ferramentas existentes no mercado

Após análise de diversas ferramentas concorrentes, foi verificado que todas elas têm um conjunto de ferramentas idênticos, diferendo no modo como interagem umas com as outras, na instalação, na licença e na localização.
Nos pontos que se seguem tentarei fazer um pequeno resumo das potencialidades, assim como aspectos a favor e contra de cada uma das ferramentas referidas.

3.1. Tuppas (Tuppas, 2011)

A Tuppas possuí um conjunto de módulos de software e que estão divididos em três grupos:

- Software de Manufactura
- Sistemas ERP (*Enterprise Resource Planning*)
- Sistemas Governamentais

Todos os grupos possuem a mesma arquitectura de sistema. O software encontra-se alojado num servidor e o uso do sistema é feito via explorador Web (Google Chrome, Mozilla Firefox, Internet Explorer, etc). Isto significa que podemos usar as aplicações em qualquer plataforma que suporte exploradores Web, sem que se esteja restringido a determinado tipo de máquina ou de sistema operativo. Não sendo ainda assim necessário uma ligação permanente, já que, em alguns casos, podem ser usadas também como aplicações *stand alone.*
Os módulos podem ser alterados e adaptados às pretensões únicas de cada cliente, integrados com outros sistemas através da importação e exportação de dados.

Módulos integrantes:

- ✓ Módulo de Programação da Produção
- ✓ Módulo de Relatórios de Produção
- ✓ Módulo de OEE (*Overall Equipment Effectiveness*)
- ✓ Módulo de Qualidade
- ✓ Módulo de Relatórios de paragens
- ✓ Módulo de Monitorização de defeito
- ✓ Módulo de Calculo de custos
- ✓ Módulo de MRP
- ✓ Módulo de monitorização de tarefas
- ✓ Módulo de relatórios de sucata

❖ A favor

- Por ser modular e em cada módulo poder haver desenvolvimento à medida, o cliente leva o que pretende.
- Possui integração com outros softwares.
- Solução bastante completa e multiplataformas.

❖ Contra

- Não possui representante em Portugal pelo que pode tornar o processo de diálogo com o cliente difícil.
- As realidades variam de país para país e o software desenvolvido pela tuppas apesar de versátil e de se poder ser à medida é assente na realidade dos USA e não na realidade portuguesa.
- A localização, pelo menos pelo que é apresentado apenas se encontra disponível a versão em inglês. Para a realidade de muitas micro e pequenas empresas isto pode ser um problema.

3.2. Sage ERP x3 (Sage, 2011)

Tal como o software apresentado no ponto anterior o Sage ERP x3 é modular e também possuí desenvolvimento à medida.

- ✓ Gestão da produção
- ✓ Gestão financeira
- ✓ Gestão comercial
- ✓ CRM
- ✓ Gestão de stocks

❖ A favor

- Solução modular e com desenvolvimento à medida.
- Solução bastante completa e multiplataformas.
- Em português.

❖ Contra

- Vocacionada para grandes empresas.

3.3. SIA – ERP: Gestão de Produção (Alidata, 2011)

Uma solução bastante completa, que integra diversas ferramentas. Pelas informações constantes na página de apresentação do produto, difere das soluções anteriores por não ter a possibilidade de desenvolvimento à medida. Tem ainda a possibilidade de integração com outras ferramentas que fazem parte do SIA – ERP.

❖ A favor

- Solução que pode ser integrada com outras ferramentas e que possuem objectivos diferentes.
- Solução bastante completa.
- Em português.

❖ Contra

- Vocacionada para industrias de produção em série.
- Não possuí desenvolvimento há medida.

3.4. Gestão de Produção (Coolsoft, 2011)

Outra solução bastante completa, mas ao contrário das soluções anteriores, não é modular nem possuí possibilidade de desenvolvimento à medida.
As ferramentas estão agrupadas nas seguintes categorias:

- Planeamento e Gestão de Produção
- Engenharia Produto e Processo
- Gestão de Stocks
- Encomendas Clientes / Fornecedores
- Produtos Acabados
- Matérias-primas e Subsidiárias

❖ A favor

- Solução que pode ser integrada com outras ferramentas e que possuem objectivos diferentes.
- Solução bastante completa.
- Em português.

❖ Contra

- Por não ser modular, contém muitas ferramentas que podem não ser úteis ao cliente e até dificultar o seu uso.
- Não possuí desenvolvimento há medida.

3.5. Gestão de Produção (Sistrade, 2011)

Solução bastante completa e que ao contrário das anteriores não é modular. As ferramentas apresentadas são sempre parte integrante do software.

Possui como principais características:

- Estruturar os métodos de produção
- Planear e controlar as diversas fases de fabrico
- Acompanhamento das encomendas em produção, previsão de entrega e lançamento dos produtos em Stock
- Apuramento dos custos de produção
- Análise de eficiências por Linha, Secção, Máquina e Empregado
- Redução de custos de produção
- Manutenção da Informação
- Planeamento da Produção
- Simulação do Planeamento de Produção
- Execução da Produção
- Controlo da Produção

- Análise da Produção – Consultas;
- Custeio da Produção
- Orçamentação
- Actualização de Tempos de Produção

❖ A favor

- Solução bastante completa.
- Em português.

❖ Contra

- Por não ser modular, contém muitas ferramentas que podem não ser úteis ao cliente e até dificultar o seu uso.
- Não possuí desenvolvimento há medida.

3.6. Gestão de Produção (ARP Sistemas Informáticos, 2011)

Solução integrada com o ERP GIN e com base num modelo predefinido, este módulo foi desenhado para ser adaptado aos processos produtivos específicos de cada organização. O GIN Gestão da Produção encontra-se assente em 6 conceitos base importantes:

- Desenvolvimento de Produto
- Orçamentação / Ficha de Custo
- Controle de Produção
- Planeamento de Produção
- Controlo de Expedição
- Gestão de Custos

❖ A favor

- Em português.

❖ Contra

- Não possuí todas as funcionalidades pretendidas
- Por não ser modular, contém muitas ferramentas que podem não ser úteis ao cliente e até dificultar o seu uso.
- Não possuí desenvolvimento há medida.

3.7. IfProd (IfThen, 2011)

Solução de planeamento e controlo de produção genérico, parametrizado de acordo com a área de negócio pretendida, mediante uma análise prévia do modelo de negócio e das suas necessidades;

O IFPROD optimiza processos, transforma dados sem valor em informação valiosa, disponibilizando-a a quem dela necessita; promove uma solução paper-free na sua organização e é fácil de utilizar e implementar.

O software tem como principais funcionalidades:

-Gestão de Encomendas
-Orçamentação
-Planeamento de Produção
-Cálculo de Necessidades
-Emissão de Ordens de Fabrico
-Registo da Produção (materiais e/ou tempos)
-Monitorização da Produção
-Identificação por código de barras
-Custos de Produção
-Controlo de Stocks
-Inventários por terminais portáteis
-Planeamento da Expedição e Registo da Expedição
-Rastreabilidade
-Gestão de obras

❖ A favor

- Em português.

❖ Contra

- Não possuí todas as funcionalidades pretendidas
- Por não ser modular, contém muitas ferramentas que podem não ser úteis ao cliente e até dificultar o seu uso.
- Não possuí desenvolvimento há medida.

3.8. Conclusão

A tabela seguinte mostra o comparativo das ferramentas que foram analisadas.

Software	Modular	Costumizável	Em Português	Integração
Tuppas	Sim	Sim	Não	Sim – com outras ferramentas do grupo
Sage ERP x3	Sim	Sim	Sim	Sim – com outras ferramentas do grupo
SIA – ERP: Gestão de Produção	Não	Não	Sim	Sim – Só com restantes ferramentas do SIA
CoolSoft: Gestão de Produção	Não	Não	Sim	Sim – com outras ferramentas do grupo
Sistrade: Gestão de Produção	Não	Não	Sim	Não
ERP Gin: Gestão de Produção	Não	Não	Sim	Sim – Só com restantes ferramentas do ERP Gin
IfThen: IfProd	Não	Sim	Sim	Informação não disponível

Tabela 1 – Comparativo Softwares Gestão de Produção

De todas as ferramentas analisadas, apenas uma não pode ser considerada concorrente, dado que não possuí as funcionalidades que se pretendem de gestão de produção, que é o ERP Gin.

O Alidata é porventura a que mais se aproxima ao que se pretende que seja desenvolvido, pois faz a cobertura de todos os elementos essenciais à gestão de produção.

Nota: A avaliação foi feita pela apresentação que as próprias empresas fazem das suas soluções via internet.

4. Desenvolvimento da solução (GEMA)

Nos pontos anteriores foi feita uma pesquisa sobre o que se pretende modelar e como é que estes se encontram implementados no terreno, se for esse o caso.

As ferramentas lean são inúmeras, pelo que uma das tarefas mais importantes passa por criar uma base de dados escalável e com todos os dados necessário para, depois de analisados, proporcionar a melhor base para se tomar decisões capazes de influenciar todo o processo produtivo.

Para o levantamento de requisitos, há que ter em conta também os objectivos que se pretendem alcançar. No final a solução deverá permitir:

- Gerir Produtos
- Gerir Equipamentos
- Gerir Processos
- Gerir Operadores
- Gerir Armazenamentos
- Gerir Ordens de Trabalho
- Gerir Bancadas
- Gerir Espaços
- Gerir Segurança do Sistema
- Realizar Cálculo Kanban
- Realizar Simulações de cálculo Kanban, Necessidades e Custos
- Gerir e Lançar Alertas

4.1. Levantamento de Requisitos

Um bom levantamento de requisitos inicial, permite poupar tempo no que respeita a modificações e implementação, para além de ir ao encontro do que o cliente realmente pretende.

Face ao constante dos pontos anteriores, verifica-se que a matéria a cobrir é muito vasta e, por isso, é necessário adoptar a "velha", mas actual táctica: *divide and conquer* (dividir e conquistar).

Nota: Nos requisitos funcionais, deve entender-se "Gerir" como acções de adicionar, editar, bloquear e apagar registos

4.1.1. Requisitos funcionais: Produtos

As metodologias lean recaem sobre o modo como os produtos são produzidos, transformados ou consumidos, bem como, onde e como são armazenados É importante ter sempre informação actualizada sobre os mesmos e um histórico que permita, por exemplo, verificar anomalias ou variação na procura por parte dos clientes.

Requisitos funcionais: Produto
Gerir produtos
Gerir famílias
Gerir modelos
Gerir classificações
Gerir classes
Dar entrada de stock
Definir regra cálculo preço compra
Determinar preço de referência de compra de produto
Ver stocks
Registar movimentos de produtos
Ver estatísticas de produção: Quantidade produzida, rejeitado
Listar compras a fornecedores
Ver fornecedores de produto
Definir defeitos comuns e possíveis soluções
Gerir encomendas

Tabela 2 - Requisitos funcionais: Produto

4.1.2. Requisitos funcionais: Processos

Tal como os produtos, os processos são essenciais e, consequentemente, elementos relevantes no que respeita ao desenvolvimento.
Os processos representam os diversos passos que é necessário percorrer até se obter o produto final. Nestes devem constar, por exemplo, os procedimentos, os vários elementos que nele participam e outros dados com importância nos cálculos das ferramentas de produção magra.
Os processos relacionam-se de forma dependente até ao processo final, formando uma linha de produção.

É através de kanbans que se marca o passo do processo. Aqueles são ferramenta essencial do JIT (*Just In Time*), que é a base da produção magra.

Assim, o requisito mais importante é o cálculo de kanbans, visto que vai definir o passo de produção do processo, evitando stocks para além do necessário e sinalizando o momento em que deve reiniciar-se a produção.

Requisitos funcionais: Processos
Gerir processos
Gerir Linhas de produção
Gerir cartões kanban
Calcular custos do processo
Calcular necessidades do processo
Calcular quantidade de kanbans
Ver estado produção
Ver ordens de trabalho
Calcular tempo de execução de ordem de trabalho
Simular o comportamento do processo de produção mudando configurações
Registo de paragens não programadas

Tabela 3 - Requisitos funcionais: Processos

4.1.3. Requisitos funcionais: Elementos do Processos

Os equipamentos, tal como as bancadas, são elementos intervenientes do processo. A capacidade, a taxa de ocupação, as manutenções e avarias são factores a ter em conta no cálculo da quantidade de kanbans. A gestão destes elementos é muito importante, para que os valores calculados possam ser os mais precisos e para que se obtenha muito pouca ou nenhuma diferença, entre o valor obtido e o valor real de produção.

Em relação às bancadas, neste ponto e para uma primeira fase, somente interessa conhecer as características no seu todo. Detalhar elementos da bancada, como por exemplo a sua disposição (em U ou em linha) iria aumentar a complexidade e, consequentemente, o tempo de desenvolvimento.

Assim, importa apenas conhecer os valores no seu todo, ficando em aberto a possibilidade de ser desenvolvido um módulo que detalhe e faça tratamento, bem como simulações destes elementos.

Requisitos funcionais: Elementos do Processo
Gerir Equipamentos
Gerir Máquinas

Gerir Marcas
Gerir Avarias Tipo
Gerir bancadas
Agendar Manutenção
Registar Manutenção

Tabela 4 - Requisitos funcionais: Elementos do Processo

4.1.4. Requisitos funcionais: Utilizadores

Aqui devem ser incluídos utilizadores que, directa ou indirectamente, interagem com o sistema.

Directamente temos os operadores, responsáveis pela introdução dos diversos dados e os administradores de sistema, responsáveis por todas as configurações. O que cada um pode ou não fazer, dependerá sempre das permissões que foram dadas à conta de entrada no sistema.

Indirectamente temos todas as pessoas que participam no processo de produção. Interessa saber o número de turnos disponíveis e as horas reais disponíveis em cada turno, dados importantes para se determinar custos e disponibilidade para produção.

Requisitos funcionais: Utilizadores
Gerir utilizadores
Gerir turnos
Gerir perfis
Gerir funções
Gerir contactos

Tabela 5 – Requisitos funcionais: Utilizadores

4.1.5. Requisitos funcionais: Espaços

O modo como estão dispostos os diferentes elementos do chão de fábrica é outro dos aspectos importantes da produção magra.

Para o âmbito deste trabalho apenas interessa ter a "planta" da fábrica, ou seja, saber, por exemplo, em que locais se encontram as zonas de armazenamento, as bancadas e/ou os equipamentos.

Os transportes internos são outro aspecto que poderia ser desenvolvido, mas a complexidade que traria para o sistema era grande. Assim, pelo motivo referido na especificação das bancadas, será um possível ponto a desenvolver, caso se pretenda

Requisitos funcionais: Espaços
Gerir espaços
Gerir pontos onde se encontram os elementos
Gerir distâncias entre pontos
Gerir secções

Tabela 6 - Requisitos funcionais: Espaços

4.1.6. Requisitos funcionais: Armazenamentos

Quando não constam do processo, os produtos têm de ser guardados, quer por um curto espaço de tempo, quer como stock, até serem necessários.

Cada espaço tem a sua localização, capacidade e formas de acondicionamento (embalagens, paletes, etc.

Requisitos funcionais: Armazenamentos
Gerir armazenamentos
Gerir tipos de armazenamentos
Gerir supermercados
Gerir embalagens

Tabela 7 - Requisitos funcionais: Armazenamentos

4.1.7. Requisitos funcionais: Picagem

Por forma a poder monitorizar-se o que se passa no processo produtivo, é necessário "picar" o trabalho realizado, a partir das respectivas ordens (de trabalho).
As ordens de trabalho são os registos dos pedidos de produção. A origem dos mesmos pode partir de encomendas de clientes ou então de procura de processos clientes.
A picagem das ordens de trabalho permite obter vários dados estatísticos relativos à procura *(histórico e flutuações na procura de determinado produto),* à produção real *(saber em que estado se encontra a ordem de trabalho: em produção, concluída, cancelada, etc.)* e a outros elementos, que ajudarão a melhor planear a agenda de produção.

Requisitos funcionais: Picagem
Gerir ordens de trabalho
Registar a evolução das ordens de trabalho, podendo ser registo no total ou descriminado por trabalhador
Agendar a produção, podendo ver a disponibilidade
Visualizar ordens de produção
Registo de rejeitado e os motivos, dividindo em retrabalho e sucata.

Tabela 8 - Requisitos funcionais: Picagem

4.1.8. Requisitos funcionais: Parceiros

Os parceiros são todas as empresas e pessoas com quem se trabalha. Embora esta gestão esteja fora do âmbito do trabalho, é necessário um conjunto mínimo de elementos, que sirvam de ponte para outras ferramentas, que gerem as relações com os parceiros.

Requisitos funcionais: Parceiros
Gerir parceiros
Gerir contactos
Listar produtos por fornecedor
Registo de encomendas de clientes
Registo histórico de compras a fornecedores
Listar vendas a clientes
Consultar estado de encomenda de cliente
Calcular tempo provável para entrega de encomenda

Tabela 9 - Requisitos funcionais: Parceiros

4.1.9. Requisitos funcionais: Segurança

Na segurança do sistema, pretende-se que sejam acautelados os aspectos de permissão de acesso a registo de introdução e alteração de dados. Estes podem ser agrupados em três pontos:

- Garantir que só tem acesso às ferramentas do sistema, quem tem permissão para as mesmas nos locais e interface que estiver configurado para tal.
- Saber quem, quando e onde entrou no sistema
- Ser possível verificar quem adicionou ou alterou um determinado registo, em base de dados e quando foi executada qualquer dessas operações

Requisitos funcionais: Segurança
Gerir postos
Definir acções dos perfis de utilizadores
Definir acções e interfaces possíveis de serem executadas em determinado posto

Tabela 10 - Requisitos funcionais: Segurança

4.1.10. Requisitos funcionais: Sistema

Pretende-se que o sistema monitorize um conjunto de situações configuráveis e que, a partir destas, lance alertas. Os alertas serão recebidos pelos destinatários definidos.
Há necessidade de saber quem criou ou alterou dados, em base de dados e quem entrou no sistema.

Requisitos funcionais: Sistema
Configurar os tipos de alertas que o sistema irá gerar sempre que as condições definidas forem cumpridas
Criar alerta manual
Consultar histórico de alertas
Restringir as funcionalidades disponíveis, dependendo da regra aplicada ao perfil de utilizador e ao posto
Registo automático de quem acede ao sistema, onde o fez e com que aplicação
Registo automático de quando é que o registo foi criado e alterado

Tabela 11 - Requisitos funcionais: Sistema

4.2. Casos de Utilização

Usando a mesma divisão do levantamento de requisitos, os casos de utilização serão agrupados em pacotes, no sentido de ser mais fácil visualizar o sistema.
Na figura seguinte estão ilustrados todos os pacotes que integram a solução que se pretende implementar e que possui os requisitos referidos no ponto anterior:

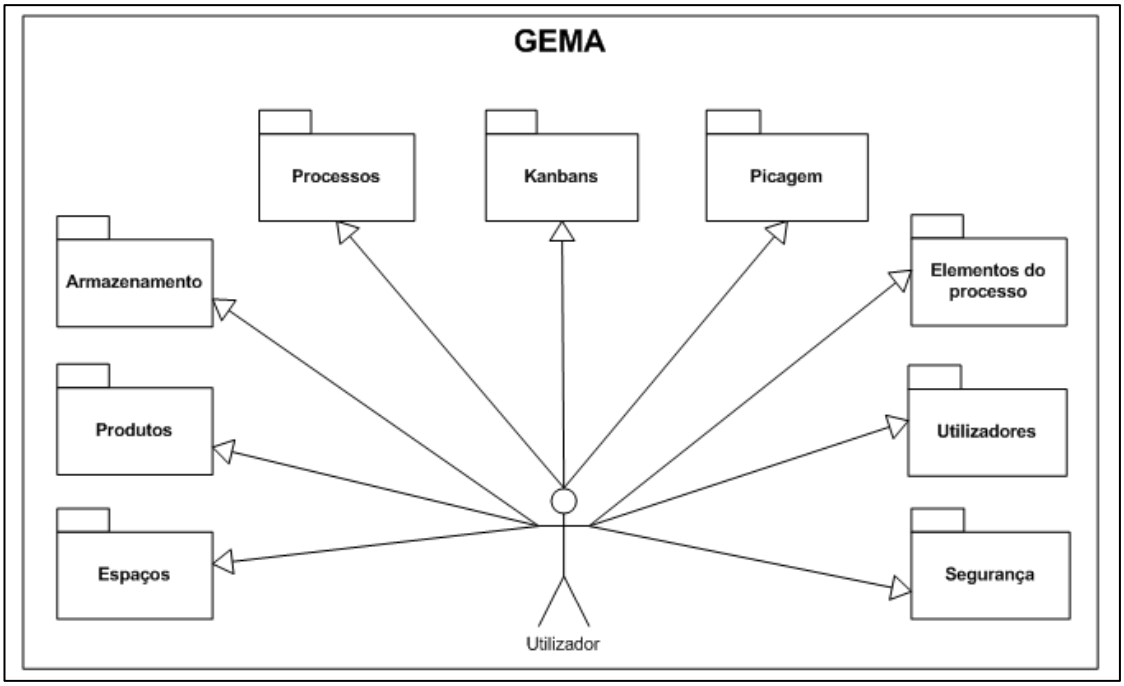


Imagem 11 – Diagrama de Casos de Utilização: GEMA

Nos pontos que se seguem serão apresentados os respectivos diagramas de casos de utilização.
Uma vez que a maioria são simples, serão detalhados apenas os mais complexos.

Para se perceber melhor a relação que existe entre os requisitos e os casos de utilização, será também apresentada uma tabela de correspondência.

4.2.1. Produtos

Imagem 12 – Diagrama de Casos de Utilização: Produtos

Requisitos <-> Casos Utilização	
Gerir produtos	Gerir produtos
Gerir famílias	Gerir famílias
Gerir modelos	Gerir modelos
Gerir classificações	Gerir classificações
Gerir classes	Gerir classes
Dar entrada de stock	Inserir movimento manual
Definir regra cálculo preço compra	Configurar regra cálculo preço
Determinar preço de referência de compra de produto	Calcular preço compra de referência
Ver stocks	Consultar stocks
Registar movimentos de produtos	Inserir movimento manual
	Consultar movimentos
Ver estatísticas de produção: Quantidade produzida, rejeitado	Configurar cálculo automático estatístico
	Corrigir dados estatísticos
	Forçar cálculo estatístico
	Consultar dados estatísticos
Listar compras a fornecedores	Consultar compras
	Inserir compras
Ver fornecedores de produto	Consultar fornecedores
Definir defeitos comuns e possíveis soluções	Gerir defeitos
Gerir encomendas	Gerir encomendas

Tabela 12- Relação Requisitos com Casos de Utilização: Produtos

Embora simples, há necessidade de clarificar alguns casos de utilização:

- **Configurar regra cálculo preço** - Dependendo da opção do gestor, a regra para o cálculo pode ser por número de compras (ex: as últimas 10 compras), entre datas ou apenas anterior a uma determinada data.

- **Calcular preço compra de referência -** a regra configurada, que é obrigatória, lista o registo de compras e calcula o preço à unidade.

- **Configurar cálculo automático estatístico -** A regra de cálculo de dados estatísticos poderá ser entre datas ou, então, anterior a uma determinada data. É necessário e obrigatório configurar o intervalo do cálculo dos dados estatísticos (ex: uma vez por mês).

4.2.2. Processos

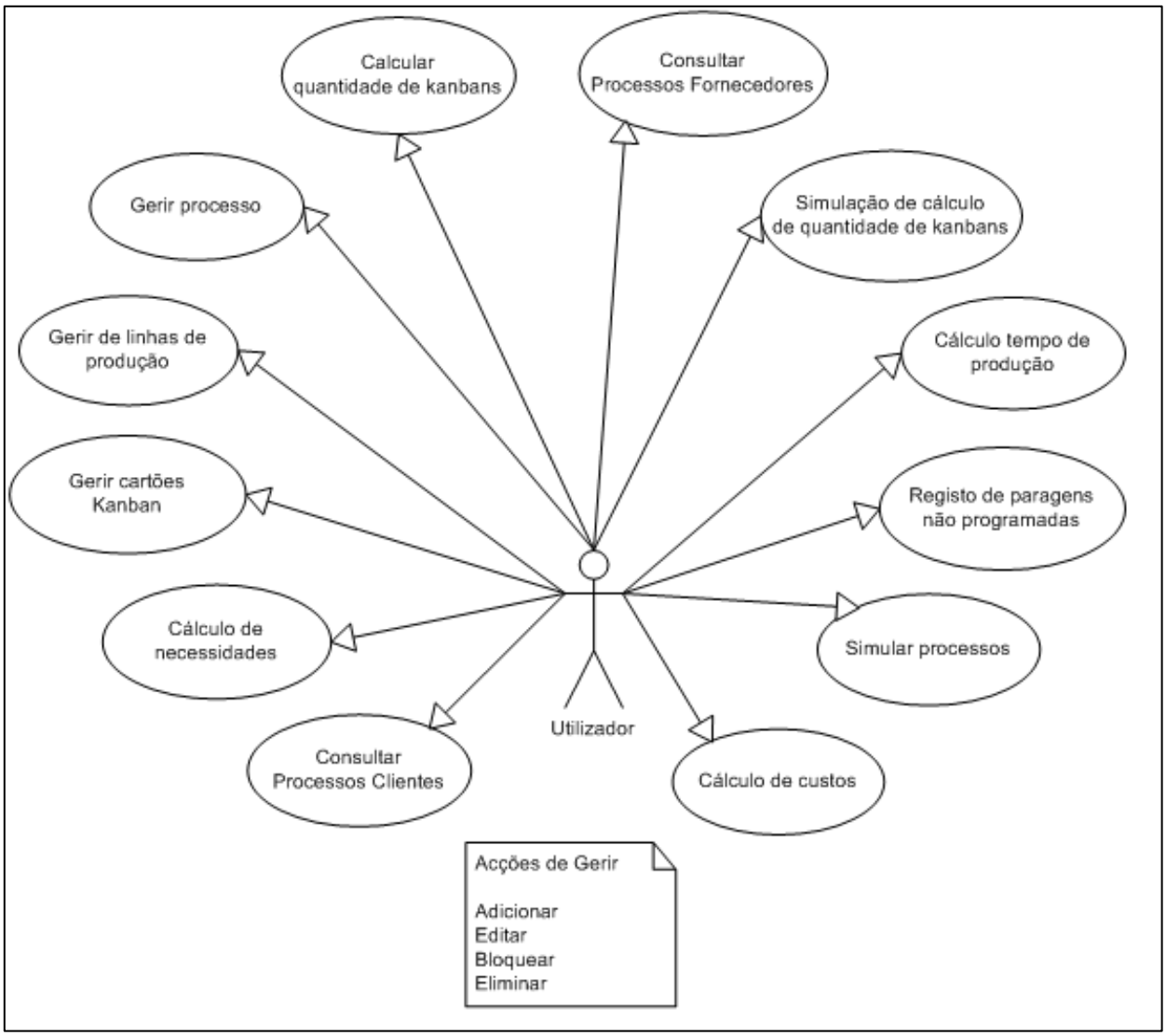


Imagem 13 – Diagrama de Casos de Utilização: Processos

Requisitos <-> Casos Utilização	
Gerir processos	Gerir processos
	Consultar processos fornecedores
	Consultar processos clientes
Gerir Linhas de produção	Gerir linhas de produção
Gerir cartões kanban	Gerir cartões kanban
Calcular custos do processo	Cálculo de custos
Calcular necessidades do processo	Cálculo de necessidades
Calcular quantidade de kanbans	Simulação de cálculo de quantidade de kanbans
	Calcular quantidade de Kanbans
Ver estado produção	
Ver ordens de trabalho	
Calcular tempo de execução de ordem de trabalho	Cálculo tempo de produção
Simular o comportamento do processo de produção mudando configurações	Simular processos
Registo de paragens não programadas	Registo de paragens não programadas

Tabela 13 - Relação Requisitos com Casos de Utilização: Processos

Para além dos casos de utilização detalhados (tabelas 13,14,15), existem outros que irei explicar:

Consultar Processos Fornecedores e **Consultar Processos Clientes** - processos fornecedores são aqueles de que depende um processo, ou seja, fornecem o produto necessário. No inverso, os processo clientes são aqueles que dependem de um processo, ou seja, "consomem" o produzido por este.

Cálculo Tempo de Produção - este cálculo traduz-se no tempo real que levará a ser produzida a quantidade desejada, caso haja disponibilidade total.

Simulação de Cálculo de Quantidade de Kanbans - os cálculos realizados serão idênticos aos de um cálculo normal. Diferem apenas quando marcados como dados de simulação e não entrarão nas contas para o cálculo real.

Nota: Os requisitos **"ver estado produção"** e **"ver ordens de trabalho"** também se encontram no pacote **"Picagem"**

Casos de utilização detalhados:

Nome:	**Calcular de quantidade de Kanbans**
Âmbito:	Gestão
Finalidade:	Simular, configurar ou novo fluxo
Actores:	Utilizador
Pré-condições:	Utilizador está identificado no sistema e tem permissões para a operação
Sequência típica dos eventos:	1. O utilizador abre o quadro de cálculo de quantidade de Kanbans 2. Escolhe adicionar novo ou editar registo já guardado 3. Introduzir/Editar os valores pretendidos e carrega em guardar 4. O sistema adiciona nova coluna ou actualiza dados da já existente e realiza o cálculo de quantidade de kanbans. 5. O utilizador carrega em gravar. 6. Os dados são gravados em base de dados e o sistema apresenta mensagem de sucesso
Sequências alternativas e extensões:	5a: O utilizador carrega em Sair 1. O sistema descarta os dados introduzidos/editados.

Tabela 14 – CU Detalhado: Calcular de quantidade de Kanbans

Nome:	**Cálculo de custos**
Âmbito:	Gestão
Finalidade:	Cálculo de custos
Actores:	Utilizador
Pré-condições:	Utilizador está identificado no sistema e tem permissões para a operação
Sequência típica dos eventos:	1. O utilizador escolhe Cálculo de Custos 2. Escolhe o produto e a quantidade pretendida. 3. O sistema verifica as necessidades e os custos inerentes às mesmas. 3. O sistema calcula o tempo que irá ser gasto para produzir o pretendido e calcula os custos de operação dos equipamentos e operadores que participam no processo. 4. O Sistema detalha estes custos e apresenta ao utilizador

Tabela 15 – CU Detalhado: Cálculo de custos

Nome:	**Cálculo de necessidades**
Âmbito:	Gestão
Finalidade:	Cálculo de necessidades
Actores:	Utilizador
Pré-condições:	Utilizador está identificado no sistema e tem permissões para a operação
Sequência típica dos eventos:	1. O utilizador escolhe Cálculo de Necessidades 2. Escolhe o produto e a quantidade pretendida. 3. O sistema verifica a quantidade de produtos necessárias. 4. O Sistema detalha a lista e a quantidade, apresentando o resultado ao utilizador

Tabela 16 – CU Detalhado: Cálculo de necessidades

Nome:	**Simulação processos**
Âmbito:	Gestão
Finalidade:	Proceder a uma simulação
Actores:	Utilizador
Pré-condições:	Utilizador está identificado no sistema e tem permissões para a operação
Sequência típica dos eventos:	1. O utilizador vai a simulações 2. Introduz os dados de configuração de simulação: → Produto → Linha de Produção → Quantidade → Prioritário → Dados para cálculo de quantidade de kanbans 3. Escolhe executar simulação 4. O Sistema verifica necessidades, custos e tempo para produção do pretendido. Se necessário recalcula a quantidade de kanbans, dependendo dos dados introduzidos. 5. O utilizador carrega em gravar. 6. Os dados são gravados em base de dados e o sistema apresenta mensagem de sucesso
Sequências alternativas e extensões:	5a: O utilizador carrega em Sair 1. O sistema descarta os dados da simulação. 5b: O Utilizador escolhe iniciar produção 1. O sistema atribui os novos valores usados na simulação e cria ordens de trabalho. 5c: O Utilizador escolhe atribuir novos valores de configuração 1. O sistema atribui novos valores e recalcula os fluxos de produção.

Tabela 17 – CU Detalhado: Simulação processos

4.2.3. Elementos do Processo

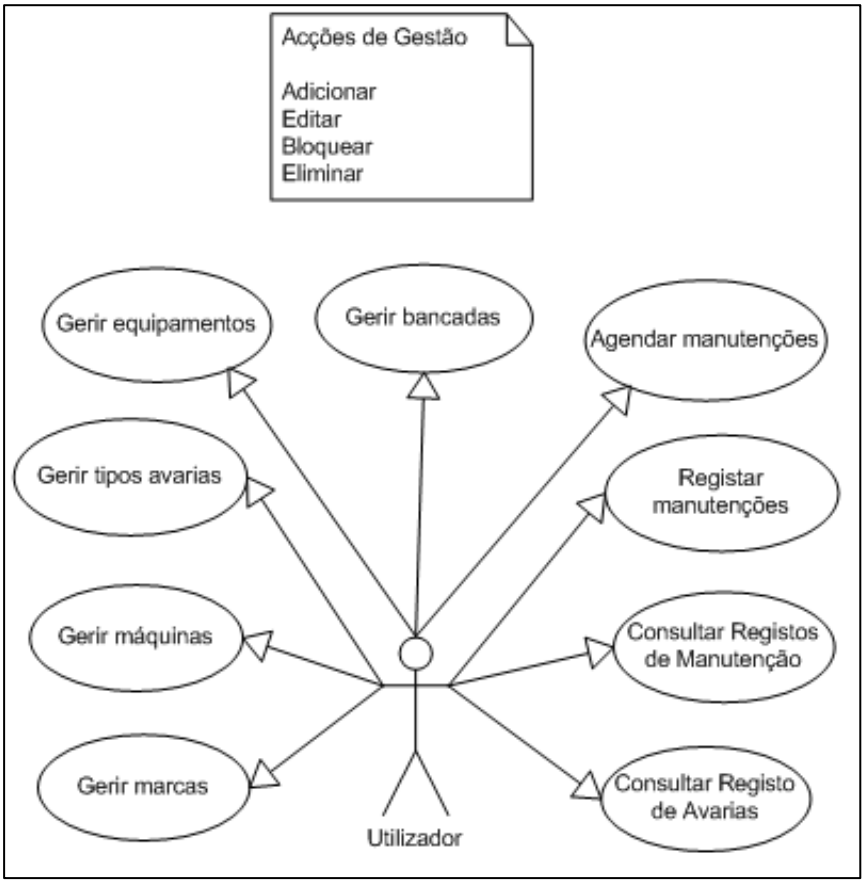


Imagem 14 – Diagrama de Casos de Utilização: Elementos do Processo

Requisitos <-> Casos de Utilização	
Gerir Equipamentos	Gerir equipamentos
Gerir Máquinas	Gerir máquinas
Gerir Marcas	Gerir marcas
Gerir Avarias Tipo	Gerir tipos avarias
	Consultar registo de avarias
Gerir bancadas	Gerir bancadas
Agendar Manutenção	Agendar manutenções
Registar Manutenção	Registar manutenções
	Consultar registos de manutenções

Tabela 18 - Relação Requisitos com Casos de Utilização: Elementos do Processo

4.2.4. Utilizadores

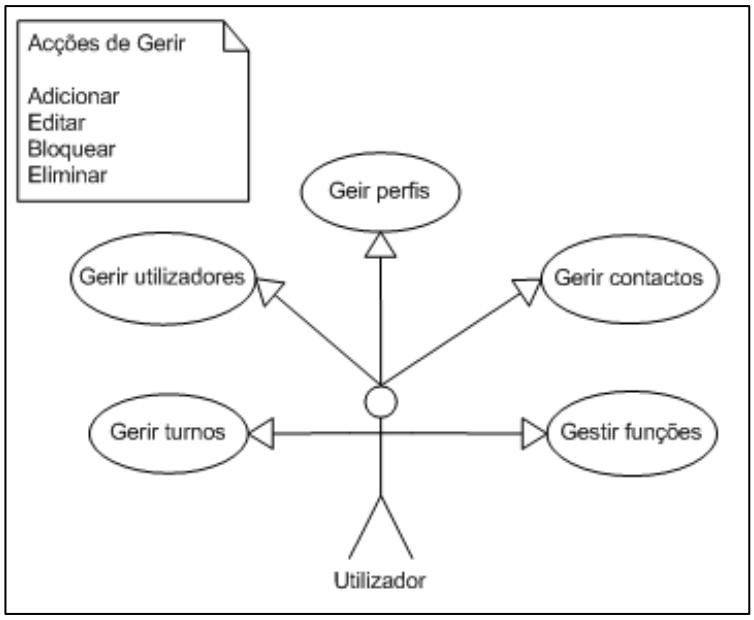


Imagem 15 – Diagrama de Casos de Utilização: Utilizadores

Requisitos <-> Casos de Utilização	
Gerir utilizadores	Gerir utilizadores
Gerir turnos	Gerir turnos
Gerir perfis	Gerir perfis
Gerir funções	Gerir funções
Gerir contactos	Gerir contactos

Tabela 19 - Relação Requisitos com Casos de Utilização: Utilizadores

4.2.5. Espaços

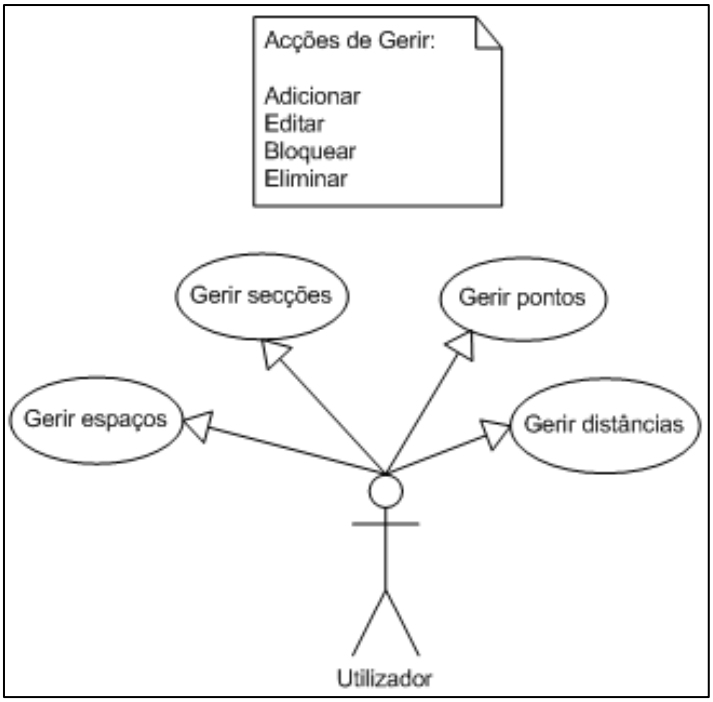


Imagem 16 – Diagrama de Casos de Utilização: Espaços

Requisitos <-> Casos de Utilização	
Gerir espaços	Gerir espaços
Gerir pontos onde se encontram os elementos	Gerir pontos
Gerir distâncias entre pontos	Gerir distâncias
Gerir secções	Gerir secções

Tabela 20 - Relação Requisitos com Casos de Utilização: Espaços

4.2.6. Armazenamento

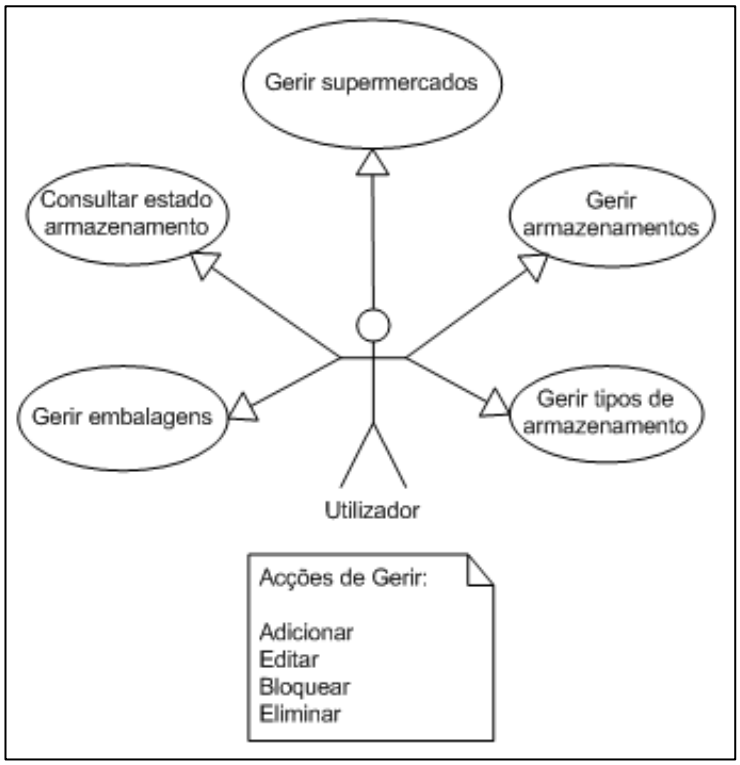


Imagem 17 – Diagrama de Casos de Utilização: Armazenamento

Requisitos <-> Casos de Utilização	
Gerir armazenamentos	Gerir armazenamentos
	Consultar estado armazenamento
Gerir tipos de armazenamentos	Gerir tipos de armazenamentos
Gerir supermercados	Gerir supermercados
Gerir embalagens	Gerir embalagens

Tabela 21 - Relação Requisitos com Casos de Utilização: Armazenamento

4.2.7. Picagem

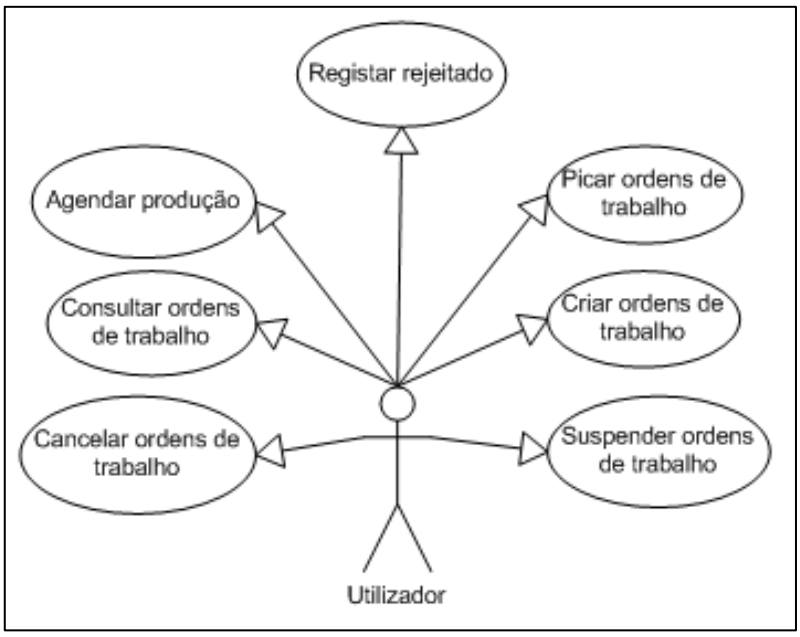


Imagem 18 - Diagrama de Casos de Utilização: Picagem

Requisitos <-> Casos de Utilização	
Gerir ordens de trabalho	Criar ordens de trabalho
	Cancelar ordens de trabalho
	Suspender ordens de trabalho
Registar a evolução das ordens de trabalho, podendo ser registo no total ou descriminardo por trabalhador	Picar ordens de trabalho
Agendar a produção, podendo ver a disponibilidade	Agendar produção
Visualizar ordens de produção	Consultar ordens de trabalho
Registo de rejeitado e os motivos, dividindo em retrabalho e sucata.	Registar rejeitado

Tabela 22 - Relação Requisitos com Casos de Utilização: Picagem

Nome:	**Picar ordens de trabalho**
Âmbito:	Gestão
Finalidade:	Registo de Produção
Actores:	Utilizador
Pré-condições:	Utilizador está identificado no sistema e tem permissões para a operação
Sequência típica dos eventos:	1. O utilizador abre a lista de ordens de trabalho que estão em produção 2. Escolhe a ordem de trabalho que pretende e carrega em registar evolução. 3. O sistema apresenta um formulário onde consta os registos anteriores já realizados e o utilizador introduz o valor total ou por operador e carrega em guardar. 4. O sistema verifica se a ordem já foi cumprida, ou seja se a soma dos valores já registados para a ordem de trabalho corresponde ao valor pedido 5. O estado da ordem de trabalho é alterado para Concluída sendo os dados são gravados em base de dados e o sistema apresenta mensagem de sucesso.
Sequências alternativas e extensões:	4a: A soma dá um valor é inferior ao pedido 1. O estado permanece sem ser concluído 4b: A soma dá um valor superior 1. O sistema apresenta mensagem de erro e não guarda os dados.

Tabela 23 – CU Detalhado: Simulação processos

4.2.8. Parceiros

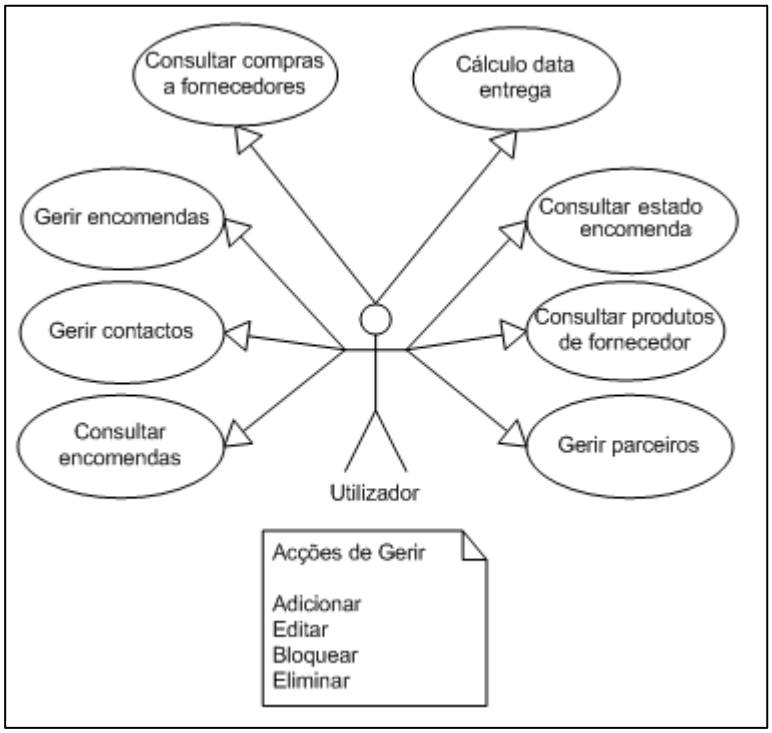


Imagem 19 - Diagrama de Casos de Utilização: Parceiros

Requisitos funcionais - Parceiros	
Gerir parceiros	Gerir parceiros
Gerir contactos	Gerir contactos
Listar produtos por fornecedor	Consultar produtos de fornecedor
Registo de encomendas de clientes	Gerir encomendas
	Consultar encomendas
Registo histórico de compras a fornecedores	Consultar compras a fornecedores
Listar vendas a clientes	Consultar vendas a clientes
Consultar estado de encomenda de cliente	Consultar estado encomenda
Calcular tempo provável para entrega de encomenda	Cálculo data entrega

Tabela 24 - Relação Requisitos com Casos de Utilização: Parceiros

4.2.9. Segurança

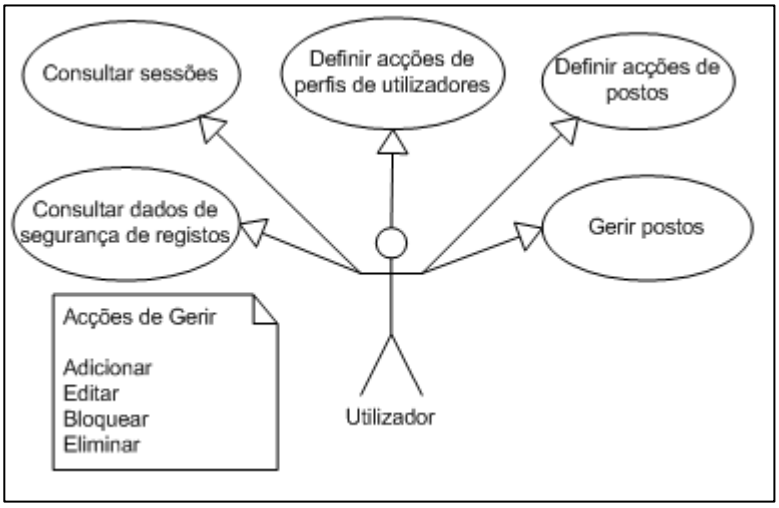


Imagem 20 - Diagrama de Casos de Utilização: Segurança

Requisitos funcionais - Segurança	
Gerir postos	Gerir postos
Definir acções dos perfis de utilizadores	Definir acções de perfis de utilizador
Definir acções e interfaces possíveis de serem executadas em determinado posto	Definir acções de posto

Tabela 25 - Relação Requisitos com Casos de Utilização: Segurança

4.2.10. Sistema

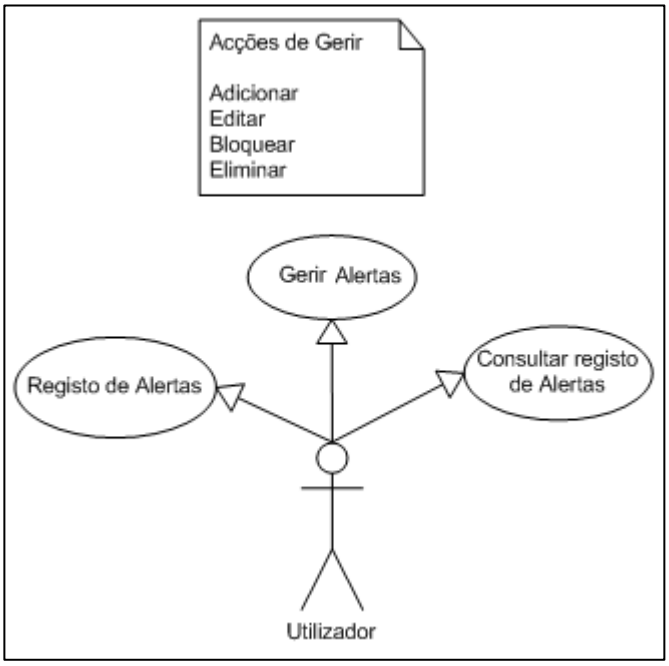


Imagem 21 - Diagrama de Casos de Utilização: Sistema

Requisitos funcionais - Sistema	
Configurar os tipos de alertas que o sistema irá gerar sempre que as condições definidas forem cumpridas	Gerir alertas
Criar alerta manual	Gerir alertas
Consultar histórico de alertas	Consultar registo de alertas
Restringir as funcionalidades disponíveis, dependendo da regra aplicada ao perfil de utilizador e ao posto	
Registo automático de quem acede ao sistema, onde o fez e com que aplicação	
Registo automático de quando é que o registo foi criado e alterado	

Tabela 26 - Relação Requisitos com Casos de Utilização: Sistema

Os três requisitos que não possuem casos de utilização são executados de forma automática.

4.3. Criação de uma base de trabalho

Após análise dos dados resultantes do levantamento de requisitos e dos casos de utilização, iniciou-se a etapa de modelação do sistema.

O desenvolvimento foi dividido em três fases distintas:

- Na primeira fase criou-se uma base de trabalho, que irá fazer a ligação entre os dados armazenados e as classes e interfaces da solução. Realizará também um conjunto de operações comuns.

- Na segunda fase criou-se uma estrutura comum a todos os elementos, promovendo, assim, um standard para o desenvolvimento.

- Na terceira fase foi pensada uma interface intuitiva e simples de usar, mas com todas as ferramentas necessárias à concretização das operações pretendidas.

4.3.1. Camada de Dados

É responsável pelas ligações à base de dados e por converter os dados lidos em classes, que seguem uma configuração standard para a segunda camada.

Todos os objectos da base de dados que são lidos, têm de implementar a interface IDBObject, a qual possui os campos necessários aos elementos de segurança: utilizador, interface, posto, data de criação do registo e data em que foi alterado, bem como a chave do estado em que o registo se encontra.

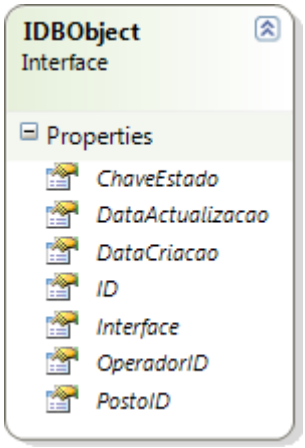


Imagem 22 - IDBObject

Para além da interface, todas as classes herdam de uma classe "genérica".

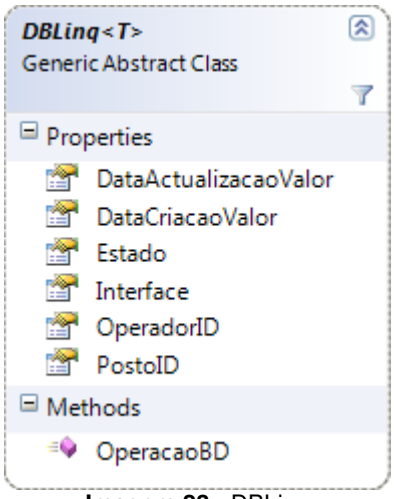


Imagem 23 - DBLinq

- **Data Actualização Valor -** Data e hora em que o registo foi editado, em base de dados.

- **Data Criação Valor -** Data e hora em que o registo foi criado, em base de dados.

- **Estado -** Estado em que se encontra, podendo assumir os valores: Normal, Bloqueado e Apagado.

- **Interface -** Interface com que o registo foi criado ou alterado.

- **OperadorID -** ID do utilizador que criou ou alterou o registo.

- **PostoID -** ID do posto em que o registo foi criado ou alterado

O método OperacaoBD tem como parâmetro de entrada TipoOperacaoBD, que é um enumerável que indica o tipo de operação que pretendemos executar. Pode assumir os valores: NovoRegisto, ActualizarRegisto, BloquearRegisto, DesbloquearRegisto, ApagarRegisto, RemoverRegistoBD.

Como retorno, tem o objecto ResultadoAccao que se encontra na imagem seguinte:

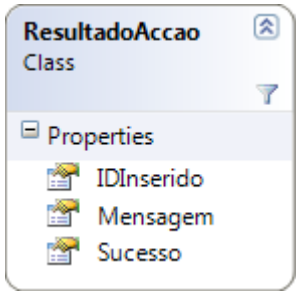


Imagem 24 – Classe ResultadoAccao

- **IDInserido -** No caso de tratar-se de novo registo e a operação ter tido sucesso, este é o valor da chave em base de dados.
- **Mensagem -** No caso de a operação não ter tido sucesso. Possui a mensagem de erro.
- **Sucesso -** Se a operação foi ou não bem sucedida.

4.3.2. Camada de lógica

Para esta camada é transparente a forma como é feito o "diálogo" com a base de dados. Representa as classes que, a partir dos casos de utilização, se verificou serem necessárias.
No entanto, todas estas classes herdam de uma classe genérica, que converte as classes da 1ª camada para a actual.

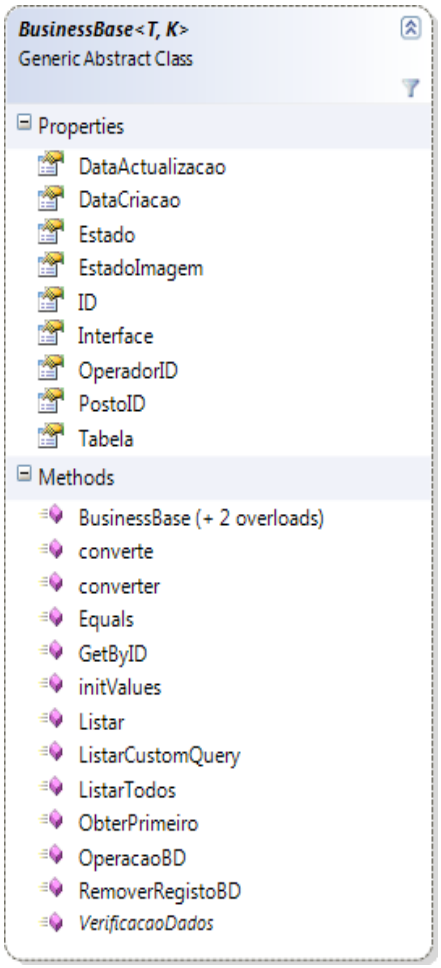


Imagem 25 - BusinessBase

As propriedades são as mesmas que já se encontram explicitadas em DBLinq. Apenas possui a imagem do estado e este será apresentado em listas.

A classe implementa um conjunto de operações que são transversais às classes desta camada:

- **Converte:** Recebe, como parâmetro, um elemento da camada de dados e retorna um elemento da segunda camada.
- **Converter:** O mesmo que Converte, mas com listas.
- **Equals:** Compara dois objectos pelos seus ID's.

- ➔ **GetByID:** Recebe, como parâmetro, um ID e retorna um objecto da segunda camada, se existir registo com ID ou null, caso não exista.

- ➔ **Listar:** dada uma expressão com as condições que se pretendem, retorna uma lista.

- ➔ **ListarCustomQuery:** o mesmo que o método anterior, mas passamos como parâmetro uma query.

- ➔ **ListarTodos**: Lista todos os elementos com estado normal.

- ➔ **ObterPrimeiro:** Obtém o primeiro valor retornado pela expressão, que é passada em parâmetro.

- ➔ **OperacaoBD:** Chama a função com o mesmo nome da 1ª camada, após executar a função verificarDados.

- ➔ **RemoverRegistoBD:** Apaga o registo da base de dados.

- ➔ **initValues:** método a implementar para inicialização de valores e variáveis locais.

- ➔ **verificarDados:** Método a ser implementado pelas classes.

De forma a facilitar as operações foram criados três tipos de dados:

Dinheiro: O seu valor é decimal, sempre arredondado à centésima e apresentado com o símbolo da moeda em uso. No caso português, o euro (€). Permite as operações binárias mais comuns (soma, subtracção, divisão e multiplicação).

Percentagem: Idêntico ao Dinheiro, mas com o símbolo de percentagem (%).

Quantidade: Possui duas propriedades que são o valor propriamente dito e as unidades, que indicam a ordem de grandeza com que se está a trabalhar. Permite as mesmas operações binárias que as classes anteriores, bem como comparações. Em qualquer operação, no caso da ordem de grandeza ser diferente, o valor de maior grandeza é convertido para o de menor grandeza e o resultado apresentado nesta.

Nota: A implementação das classes e interfaces atrás referidos, encontram-se no Apêndice.

4.4. Diagramas de Classes e Físicos

A partir dos casos de utilização, bem como da imposição da estrutura idealizada, foram obtidos os seguintes diagramas de classes e correspondentes diagramas Físicos, que se encontram em correlação com a estrutura de dados persistentes.
Devido à dimensão, os pacotes foram mais divididos, de forma a facilitar a visualização dos diagramas.

4.4.1. Produtos

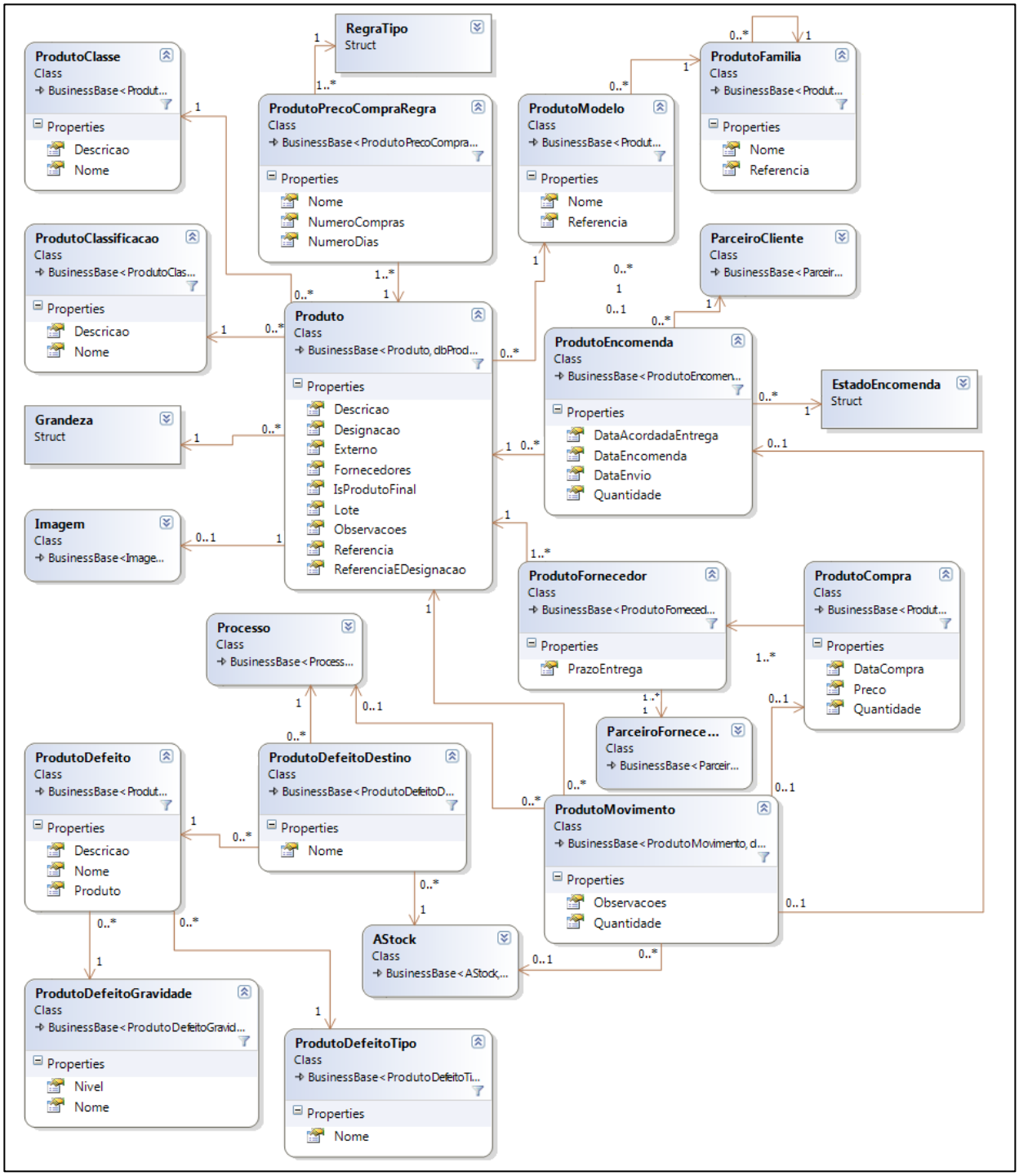


Imagem 26 – Diagrama de classes: Produto

Não há muito a clarificar, pois o diagrama é bastante explícito. Quase todas as classes representam os dados guardados em base de dados e os métodos são somente os que herdam da classe base.
As excepções vão ser a seguir discriminadas:

EstadoEncomenda: Enumerável que pode assumir os valores "Espera", "Producao", "Cancelada","Parada", "Pronta", "Expedida".

Grandeza: Enumerável que pode assumir os valores "Nenhuma", "Tempo", "Comprimento", "Area", "Volume", "Peso".
Quando a grandeza de um produto assume o valor "Nenhuma", indica que estamos a trabalhar com peças.

RegraTipo: Enumerável que pode assumir valores "PorNumeroCompras",

"PorNumeroDias", "PorIntervaloDatas", "AntesDaData".

ProdutoFamilia: Irá permitir agrupar os modelos de produtos em famílias e sub-familias.

ProdutoCompra: É nesta classe que está implementado o cálculo do preço de referência de um produto.

A classe "ProdutoMovimento" irá registar todos os movimentos de produtos. Por este motivo, tornar-se-á uma tabela que vai crescer muito rapidamente e "pesar" no processamento.

De forma a conseguir lidar com esta situação, existem duas classes, uma que possui as estatísticas relativas à produção e a outra o saldo de produtos em stock.

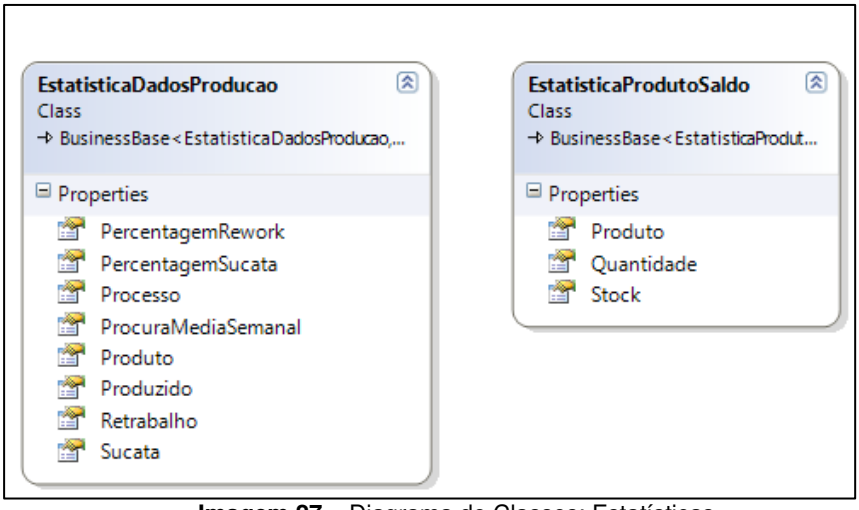


Imagem 27 – Diagrama de Classes: Estatísticas

No diagrama seguinte podemos ver a estrutura de dados persistentes, respeitante ao diagrama de classes dos produtos:

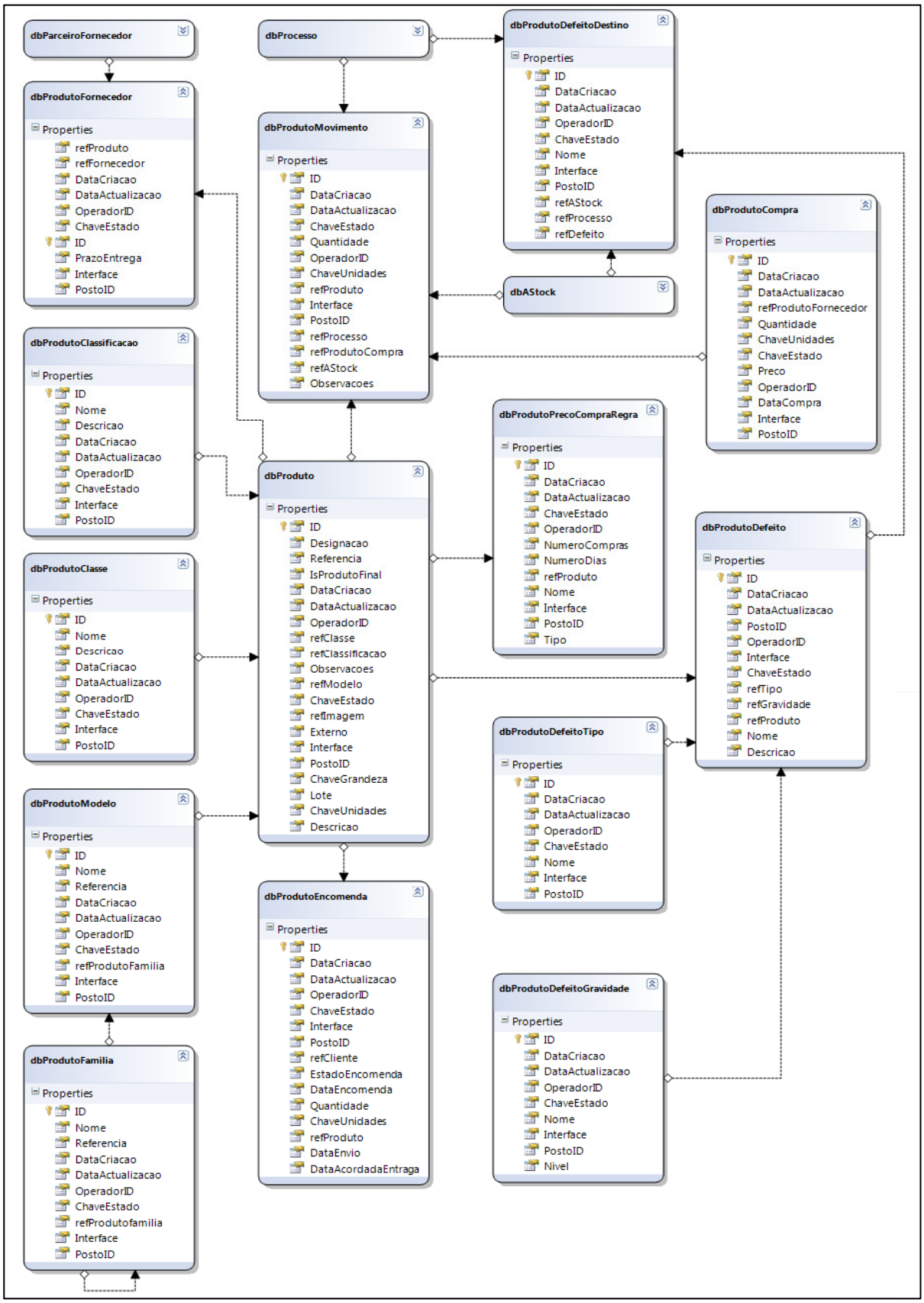


Imagem 28 – Diagrama físico: Produtos

Diagrama físico da Estatística:

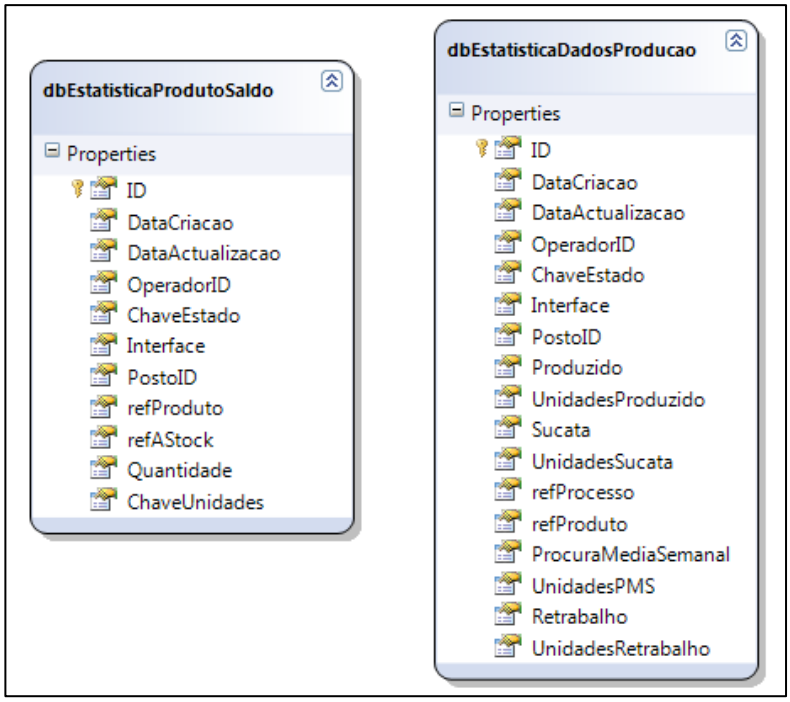


Imagem 29 – Diagrama físico: Estatísticas

Nota:

As classes que possuem propriedades do tipo Quantidade, apresentam dois valores, um decimal e outro inteiro, que é a chave da unidade de grandeza do valor guardado.

Usando como exemplo o diagrama anterior, temos "ProcuraMediaSemal" e "UnidadesPMS".

4.4.2. Processos

Face ao tamanho com que ficaria o diagrama, dividiu-se o mesmo em dois:

Diagrama de classes processos:

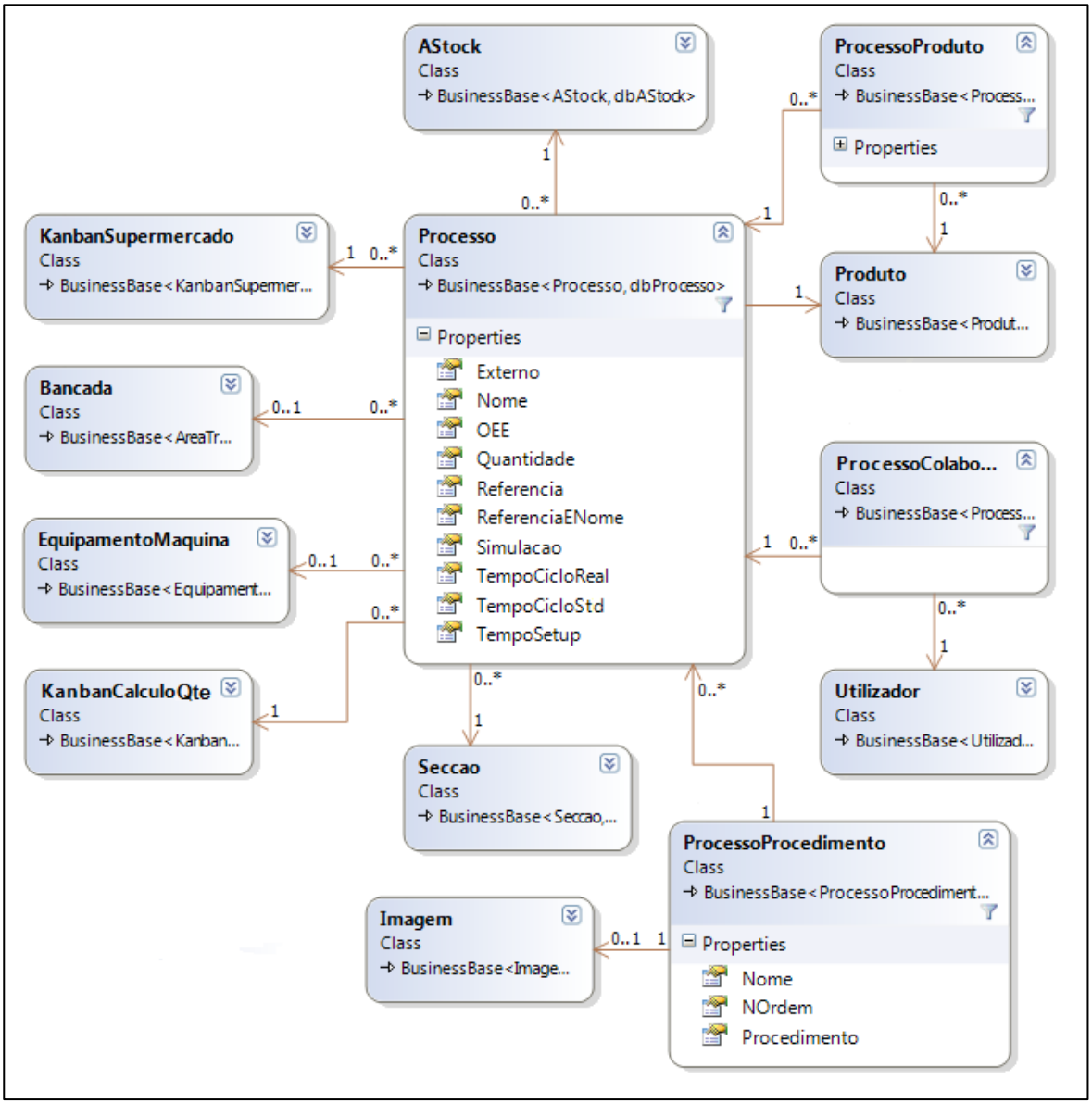


Imagem 30 – Diagrama de classes: Processos

Apesar de simples, no que a propriedades diz respeito, a classe Processos possui um número significativo de métodos, os quais passo a descrever:

- **ArvoreDependenciasProcessos**

  Método recursivo que cria, como o próprio nome indica, a árvore de dependências do processo, em que é chamada a função.
  Recebe, como parâmetros, uma lista de processos e a quantidade que se deseja produzir.
  A lista devolve um conjunto de elementos do tipo ProcessoCalculo(1), os quais representam a cadeia de dependências dos processos, que precedem o processo em causa. No caso da quantidade a produzir ser diferente de 0, serão também calculados os custos.

- **DarBaixaNecessidades**

  Recebe, como parâmetro, a quantidade produzida, calcula a quantidade de produtos gastos e regista o respectivo consumo

- **CalculoCustosColaboradores**

  Recebe uma quantidade de produto e calcula os custos da mão-de-obra associados à produção da mesma (quantidade)

- **CalculoCustosEquipamento**

  Recebe uma quantidade de produto e calcula o custo do equipamento associado à produção da mesma (quantidade)

- **CalculoCustosProdutos**

  Recebe uma quantidade de produto e calcula o custo dos produtos necessários à produção da mesma (quantidade)

- **CalculoCustosTotais**

  Recebe uma quantidade de produto e calcula os custos com os Colaboradores, Equipamentos e Produtos, retornando a sua soma.

- **NecessidadesPorUnidadeProduzida**

  Devolve a quantidade necessária do produto passado por parâmetro, para cada unidade produzida no processo

- **NecessidadesProdutos**

  Determina a quantidade de produtos, que é necessária para a quantidade da produção desejada

(1) Ver a implementação da classe no anexo V

Diagrama físico Processos:

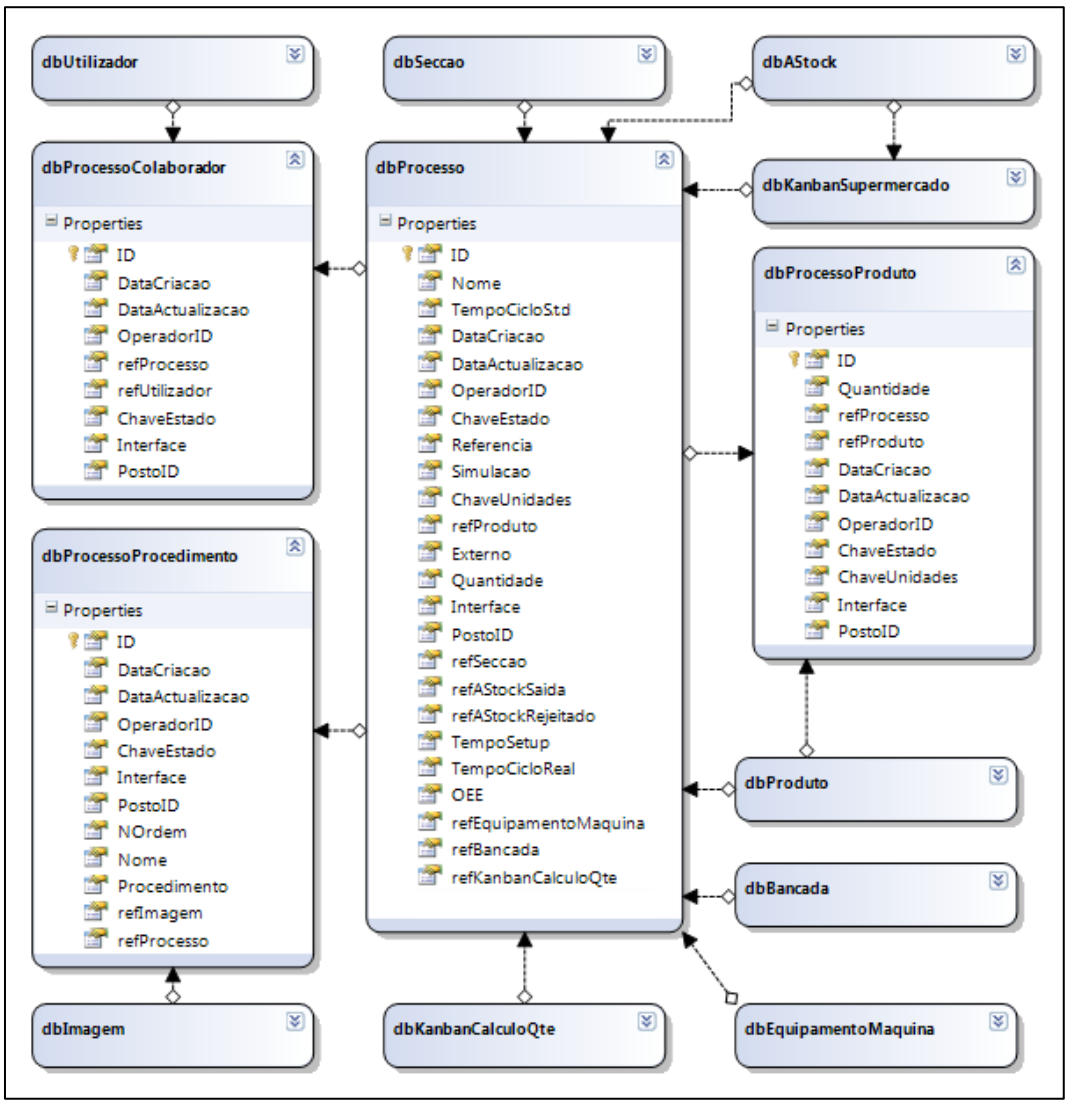


Imagem 31 – Diagrama físico Processos

Diagrama de classes Kanban:

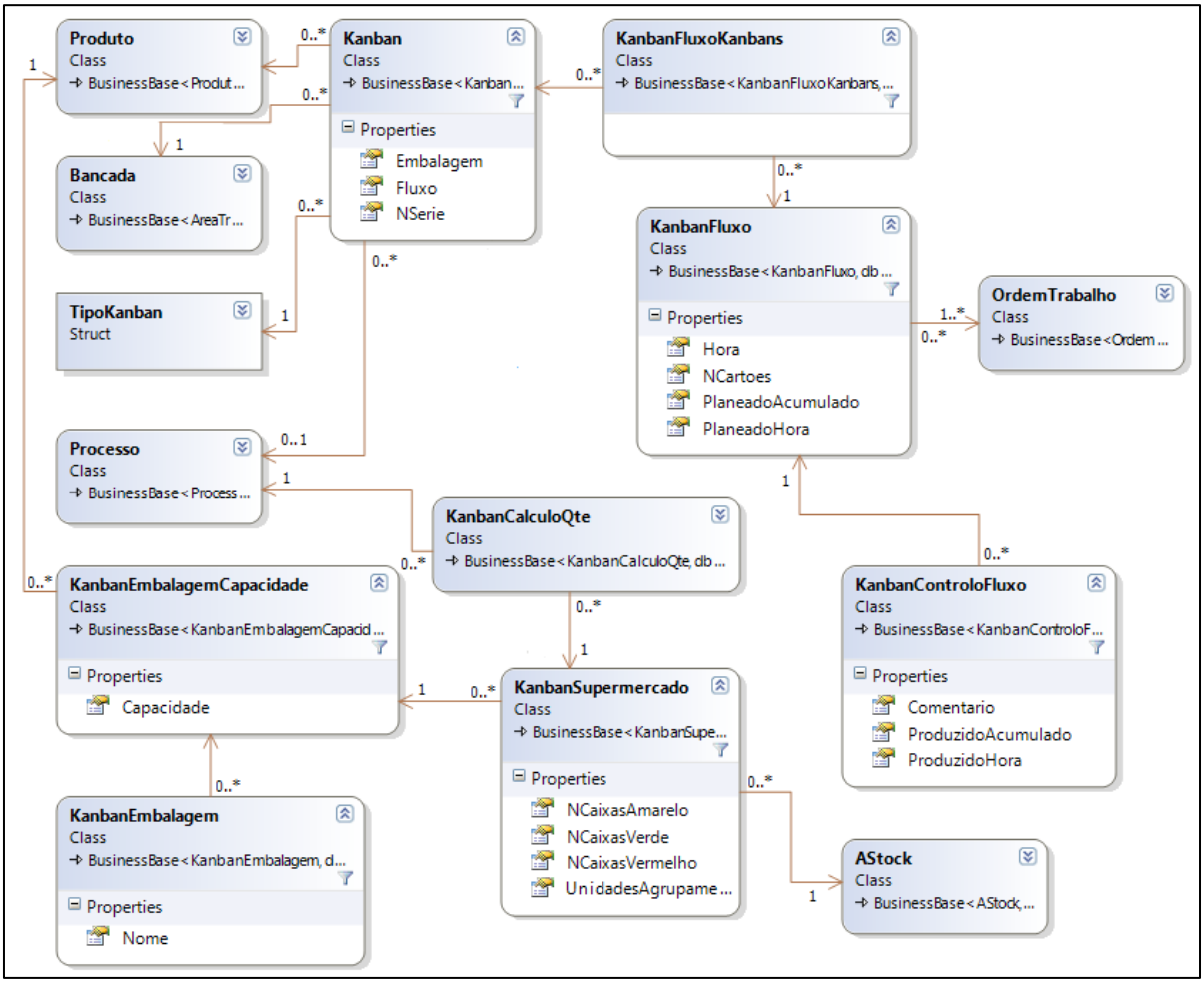


Imagem 32 – Diagrama de classes Kanbans

À excepção da classe KanbanCalculoQte, quase todas as classes são muito simples.

Um dos casos de utilização é o cálculo de quantidade de kanbans, ferramenta muito importante da produção magra. A quantidade de kanbans está dividida em três patamares:

- ➔ **Verde – Lote de Produção:** Enquanto houver kanbans no verde não é urgente de iniciar produção.
- ➔ **Amarelo – Lote de Reposição:** Ainda não é urgente, mas é conveniente iniciar produção.
- ➔ **Vermelho – Lote 2S Segurança:** É urgente iniciar produção, sob pena de faltar stock e ter(em) de parar(em) o(s) processo(s) clientes dependentes.

Para este cálculo são necessários vários dados, uns que vêm dos diversos elementos configurados, outros que são estatísticos e ainda outros que devem ser calculados na altura.

Na imagem que se segue podemos ver o conjunto de propriedades, que depois explicarei, bem como o modo como são feitos os cálculos:

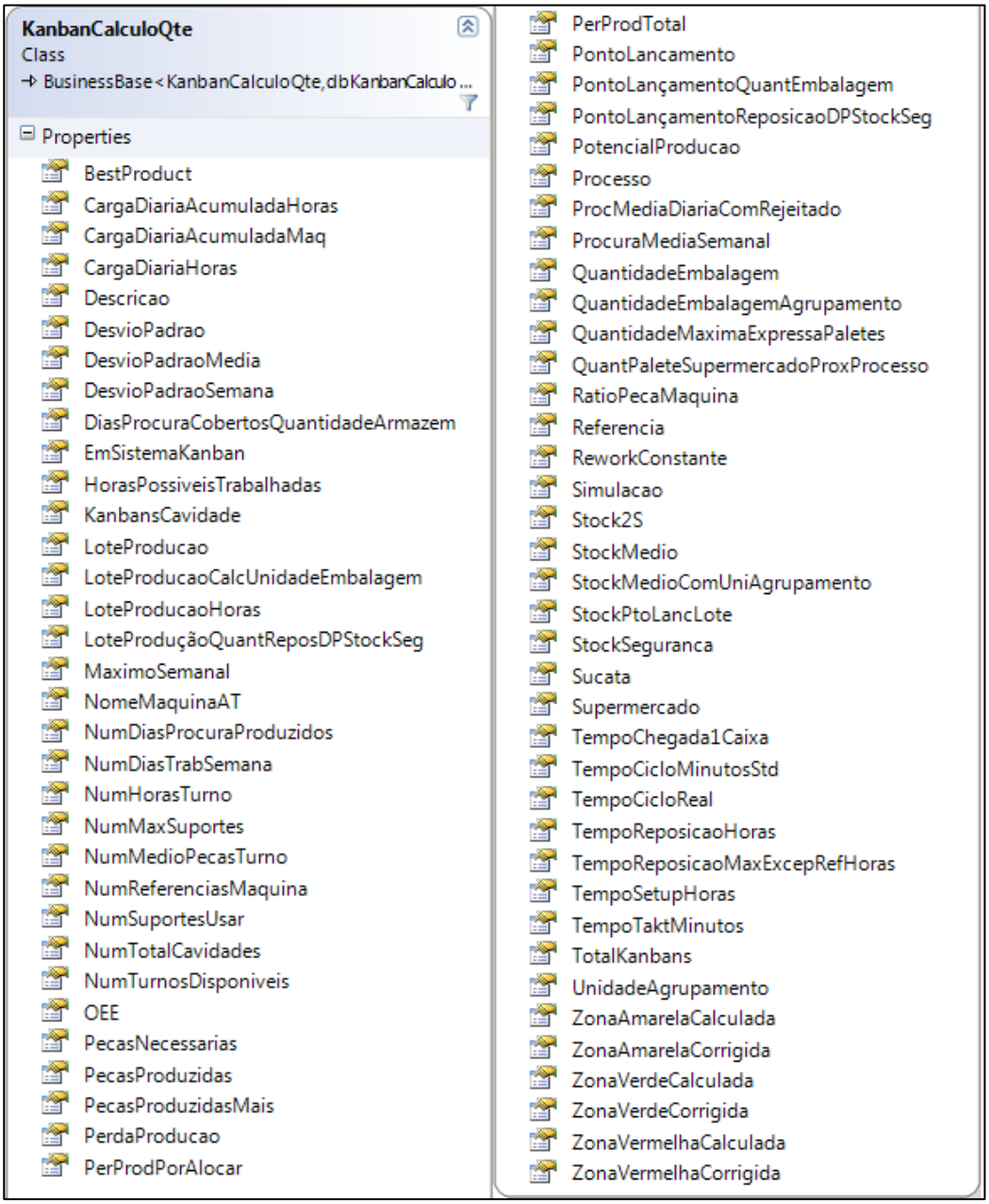


Imagem 33 – Classe KanbanCalculoQte

A implementação desta classe pode ser consultada no Apêndice VI.

Na tabela seguinte são descritas as propriedades desta classe e os valores que elas assumem.

ID	Nome propriedade	Descrição	Cálculo
1	**Referencia**	Referência do produto	Propriedade do produto
2	**Descrição**	Descrição do produto	Propriedade do produto
3	**NomeMaquinaAT**	Nome da máquina ou bancada a usar	Propriedade da máquina ou bancada, dependendo da configuração do processo
4	**PerProdTotal**	Percentagem de produção alocada a este processo	Valor previamente configurado
5	**PerProdPorAlocar**	Percentagem de Produção que falta alocar	Valor previamente configurado
6	**ProcuraMediaSemanal**	Procura Média Semanal	Valor previamente configurado
7	**DesvioPadraoMedia**	Média do desvio padrão	Valor previamente configurado
8	**DesvioPadraoSemana**	Desvio padrão por semana	**inteiro arredondado para cima de [6] * [7]**
9	**MaximoSemanal**	Máximo semanal	**[6] + [8]**
10	**ProcMediaDiariaComRejeitado**	Procura média diária com % de sucata e retrabalho	**inteiro arredondado para cima de ([6] * ([17] + [18])) / [21]**
11	**DesvioPadrao**	Desvio padrão	**inteiro arredondado para cima de [10] * [11]**
12	**NumReferenciasMaquina**	Número de referências fabricadas na máquina / bancada	**percorrer todos os registos e contar**
13	**TempoCicloMinutosStd**	Tempo ciclo standard em minutos	Propriedade do processo
14	**TempoCicloReal**	Tempo de Ciclo Real afectado do OEE em minutos	Propriedade do processo
15	**NumMedioPecasTurno**	Nº Médio de Peças por turno	**inteiro arredondado para cima de ([23] * 60)) / [14]**
16	**OEE**	OEE	Propriedade do processo
17	**Sucata**	Percentagem de Sucata	Propriedade das estatísticas
18	**ReworkConstante**	Percentagem de Retrabalho	Propriedade das estatísticas
19	**CargaDiariaHoras**	Carga diária em horas	**[10] * [14]**
20	**CargaDiariaAcumuladaHoras**	Carga diária acumulada em horas	**soma da carga diária de todas as**
21	**NumDiasTrabSemana**	Número de dias Trabalhados numa semana	Propriedade do turno
22	**NumTurnosDisponiveis**	Número de Turnos Disponíveis	Propriedade do turno
23	**NumHorasTurno**	Número de Horas por Turno	Propriedade do turno
24	**CargaDiariaAcumuladaMaq**	Percentagem ocupação da referência na máquina ou bancada	**([20] / ([22] * [23]))*100**
25	**NumMaxSuportes**	Número máximo de suportes	Valor previamente configurado
26	**NumSuportesUsar**	Número de suportes a usar	Valor previamente configurado
27	**Processo**	Processo	Valor previamente configurado
28	**RatioPecaMaquina**	Percentagem do peso da referência na carga total da máquina ou bancada	[19] / [20] * 100
29	**HorasPossiveisTrabalhadas**	Horas Possíveis ou Trabalhadas	**Se [20] > ([22] * [23]) então [28] * [22] * [23] senão [19]**
ID	**Nome propriedade**	**Descrição**	**Cálculo**

30	**PecasProduzidas**	Peças Produzidas	**inteiro arredondado para cima de ([29] * 60) / [14]**
31	**PecasNecessarias**	Peças necessárias	**[10]**
32	**PerdaProducao**	Perda de produção	**[10] - [30]**
33	**PotencialProducao**	Potencial de produção	**inteiro arredondado para cima de Se [20] <= ([22]*[23]) então ([28]*([22]*[23]))-[24] senão 0**
34	**BestProduct**	Melhor produto a ser produzido caso não exista sobrecarga	**inteiro arredondado para cima de Se [20] <= ([22]*[23]) então ([22]*[23])-[24] senão 0**
35	**NumDiasProcuraProduzidos**	Número de dias de procura produzidos sempre que entra em produção	Valor previamente configurado
36	**QuantidadeEmbalagem**	Quantidade por Embalagem	Propriedade Supermercado
37	**UnidadeAgrupamento**	Unidade de agrupamento	Propriedade Supermercado
38	**QuantidadeEmbalagemAgrupamento**	Quantidade por Embalagem com unidade de agrupamento	**[36] * [37]**
39	**TempoSetupHoras**	Tempo de setup em horas	Propriedade Processo
40	**LoteProducao**	Lote de produção em peças	**inteiro arredondado para cima de** [10] * [35]
41	**LoteProducaoHoras**	Lote de produção em horas	**[40] * [14]**
42	**LoteProducaoCalcUnidadeEmbalagem**	Lote de produção calculado na unidade de Embalagem	**inteiro arredondado para cima de [40] / [38]**
43	**PecasProduzidasMais**	Peças produzidas a mais	**[42] * [36] - [40]**
44	**TempoChegada1Caixa**	Tempo chegada 1ª caixa em horas	Valor previamente configurado
45	**TempoReposicaoHoras**	Tempo de reposição em horas	**[41] +[39] +[44]**
46	**TempoReposicaoMaxExcepRefHoras**	Tempo de reposição máximo exceptuando a referência em análise para esta máquina em horas	Percorrer todos os elementos com o mesmo equipamento ou bancada e ver qual o maior
47	**TempoTaktMinutos**	Tempo takt em minutos	**([22] * [23] * 60) / [10]**
48	**PontoLancamento**	Ponto de lançamento em peças	**inteiro arredondado para cima de ([46] * 60) / [47]**
49	**PontoLancamentoQuantEmbalagem**	Ponto de lançamento *só para a reposição, em quantidade por embalagem*	**inteiro arredondado para cima de [49] / [38]**
50	**StockPtoLancLote**	Stock ponto lançamento + lote	**[49] + [42]**
51	**Stock2s**	Stock 2S	**(2* [11] * [35]) / [36]**
52	**StockSeguranca**	Stock Segurança	**(([10] + (2 * [11])) * [35] * [17]) / [36]**
53	**PontoLancamentoReposicaoDPStockSeg**	Ponto de lançamento *para a reposição + DP + Stock segurança*	**[51] + [52] + [49]**
54	**LoteProcucaoQuantReposDPStockSeg**	Lote de Produção + Quantidade de Reposição *+ DP + Stock segurança*	**[42] + [53]**
55	**StockMedio**	Stock médio	**([41] + (2 * [11]) + [48]) * (1 + [17]) / 2**
56	**StockMedioComUniAgrupamento**	Stock médio com unidade de agrupamento	**[55] / [38]**
57	**QuantPaleteSupermercadoProxProcesso**	Quantidade por palete no supermercado/próximo processo	Valor previamente configurado
ID	**Nome propriedade**	**Descrição**	**Cálculo**
58	**QuantidadeMaximaExpressaPaletes**	Quantidade máxima expressa em paletes	**inteiro arredondado para cima de ([54] * [40]) / [57]**

59	**DiasProcuraCobertosQuantidadeArmazem**	Dias de procura cobertos pela quantidade em armazém	**([58] * [57])** / **[10]**
60	**ZonaVerdeCalculada**	Zona Verde Calculada	**[42]**
61	**ZonaAmarelaCalculada**	Zona Amarela Calculada	**inteiro arredondado para cima de [54] - [60] - [62]**
62	**ZonaVermelhaCalculada**	Zona Vermelha Calculada	**inteiro arredondado para cima de [51] + [52]**
63	**ZonaVerdeCorrigida**	Zona Verde Corrigida	Valor a ser preenchido caso Sena necessário correcção ao valor de [60] senão assume valor deste
64	**ZonaAmarelaCorrigida**	Zona Amarela Corrigida	Valor a ser preenchido caso Sena necessário correcção ao valor de [61] senão assume valor deste
65	**ZonaVermelhaCorrigida**	Zona Vermelha Corrigida	Valor a ser preenchido caso Sena necessário correcção ao valor de [62] senão assume valor deste
66	**TotalKanbans**	Total de Kanbans	**[63] + [64] + [65]**
67	**KanbansCavidade**	Kanbans por cavidade	**inteiro arredondado para cima de [57]** / **[38]**
68	**NumTotalCavidades**	Número Total de Cavidades	**inteiro arredondado para cima de [66]** / **[67]**
69	**Supermercado**	Supermercado onde irá ficar o produto a ser controlado	Valor previamente configurado
70	**EmSistemaKanban**	Se o produto se encontra em sistema kanban	Valor a definir
71	**Simulacao**	Se o calculo é uma simulação	Valor a definir

Tabela 27 – Propriedades da classe KanbanCalculoQte

Como podemos ver, apesar de complexa em termos de dimensão, a estrutura de dados persistente é simples e, por isso, encontra-se juntamente com as outras classes. A imagem seguinte representa o diagrama físico resultante do diagrama de classes.

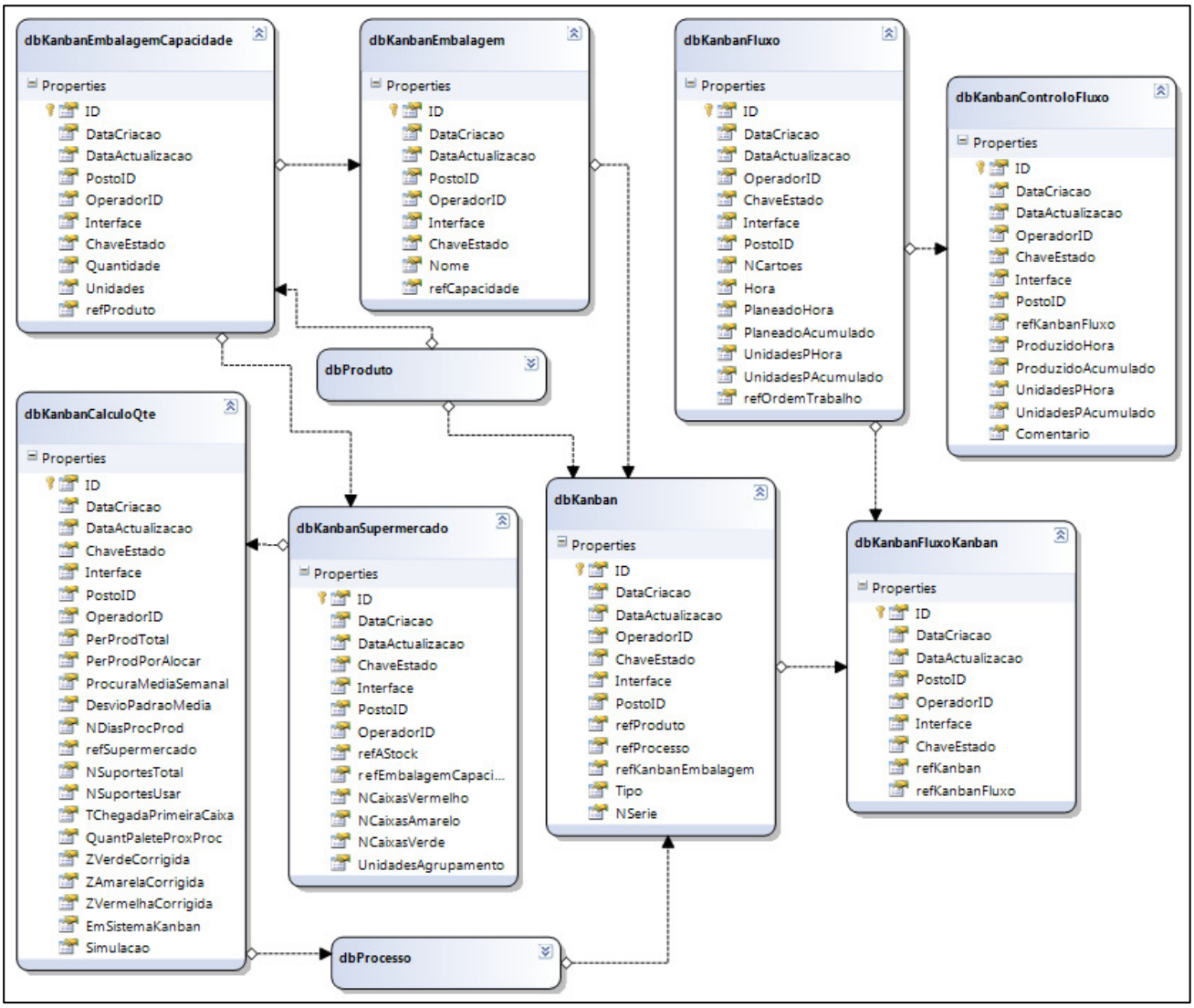


Imagem 34 – Diagrama físico: Kanbans

4.4.3. Elementos do Processo

São dois os tipos de elementos do processo:

Bancadas

Como já foi referido anteriormente, este é um elemento de futuro desenvolvimento, mas para a solução apresentada apenas precisamos de identificar a bancada e o local onde se encontra.

Diagrama de classes:

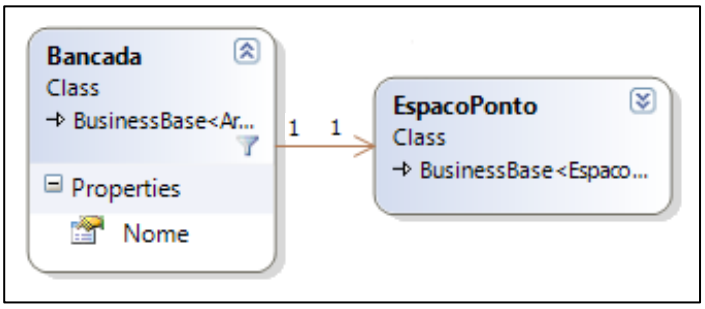


Diagrama físico:

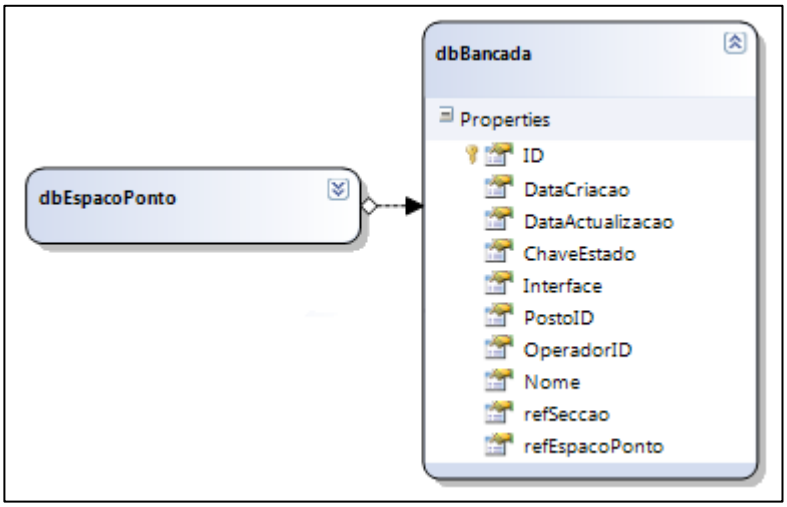


Imagem 36 – Diagrama físico: Bancadas

Equipamentos:

Quanto aos equipamentos, trata-se apenas de elementos de configuração e de registos. À excepção de algumas listagens que se pretendam tirar, não tem outras funcionalidades além da que é comum a todas as classes e que já foi explicada.

Diagramas de classes:

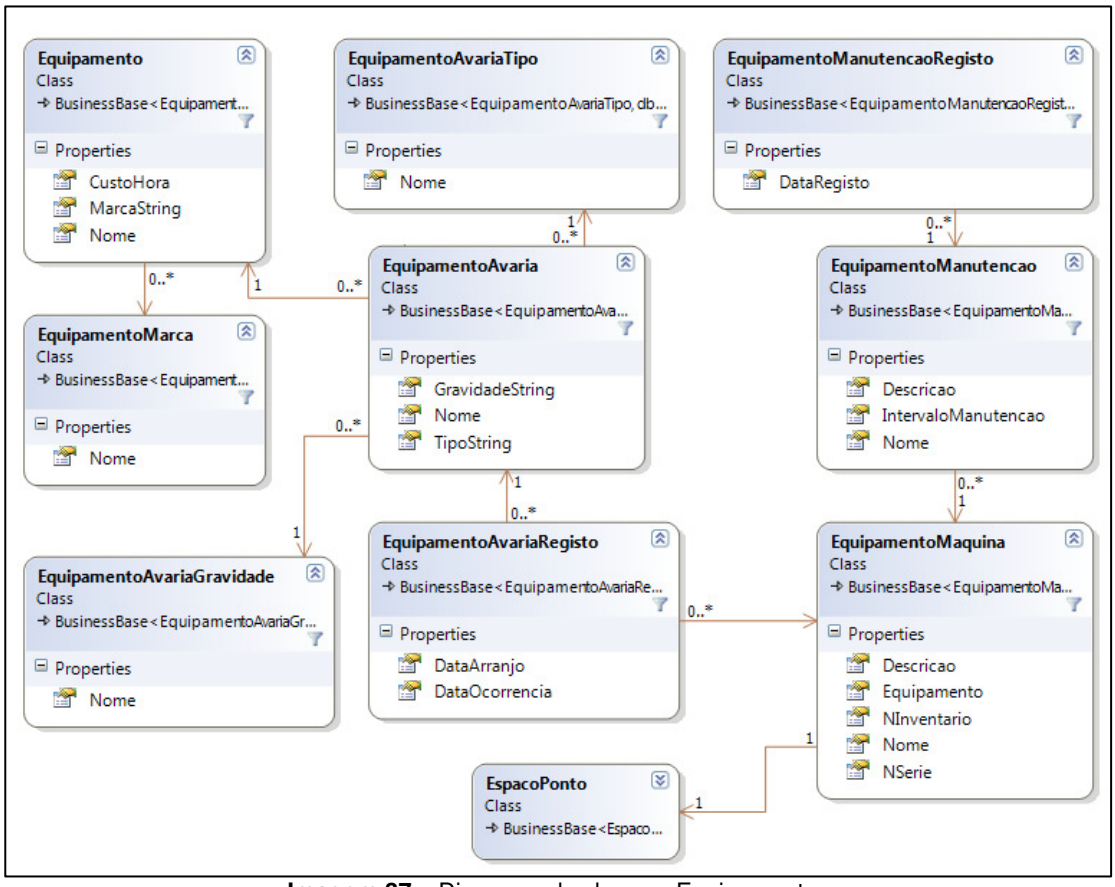


Imagem 37 – Diagrama de classes: Equipamentos

Diagrama físico:

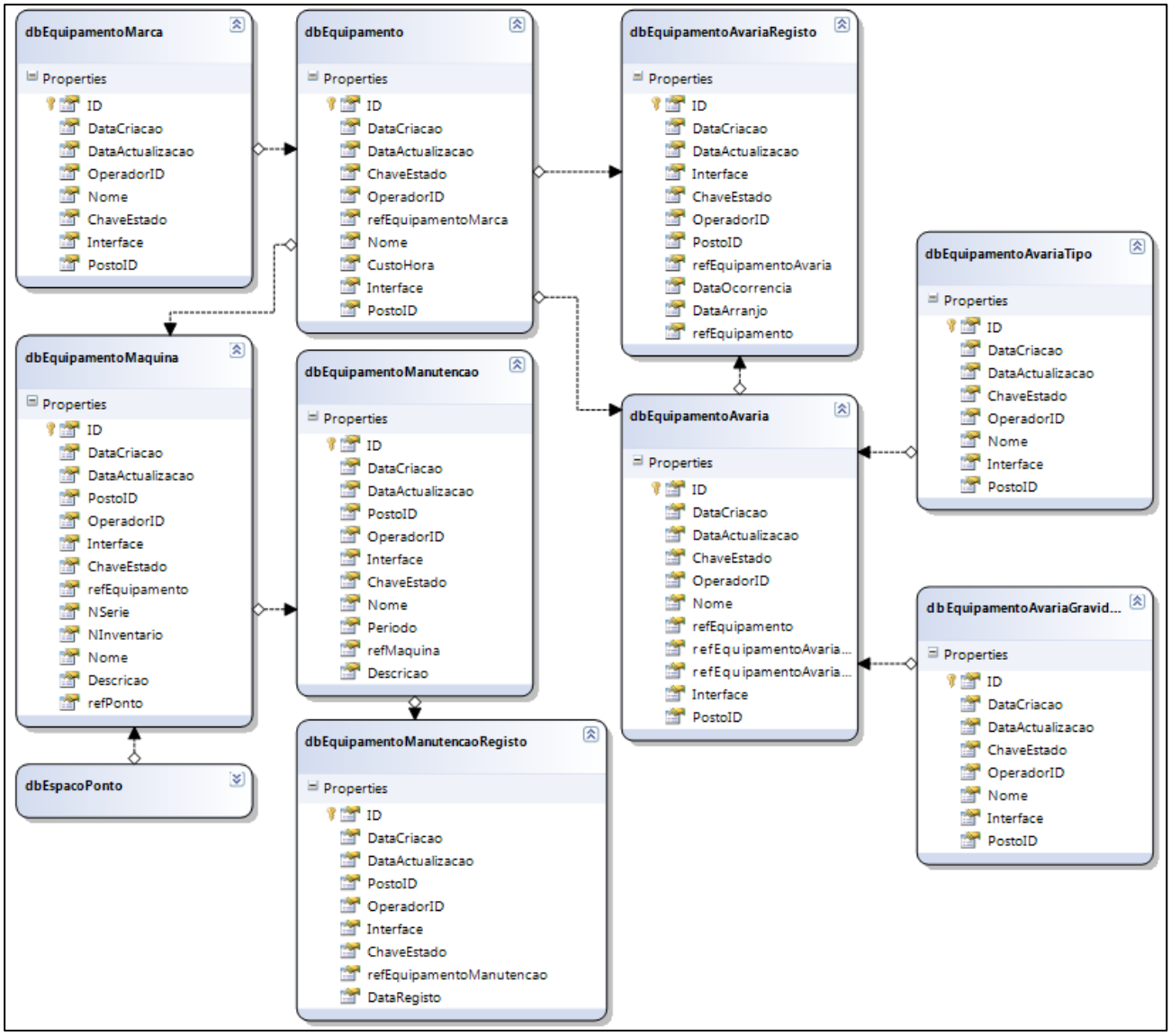


Imagem 38 – Diagrama físico: Equipamentos

4.4.4. Utilizadores

A gestão de utilizadores é, tal como os elementos do processo, apenas de configuração e de gestão.

Diagrama de classes:

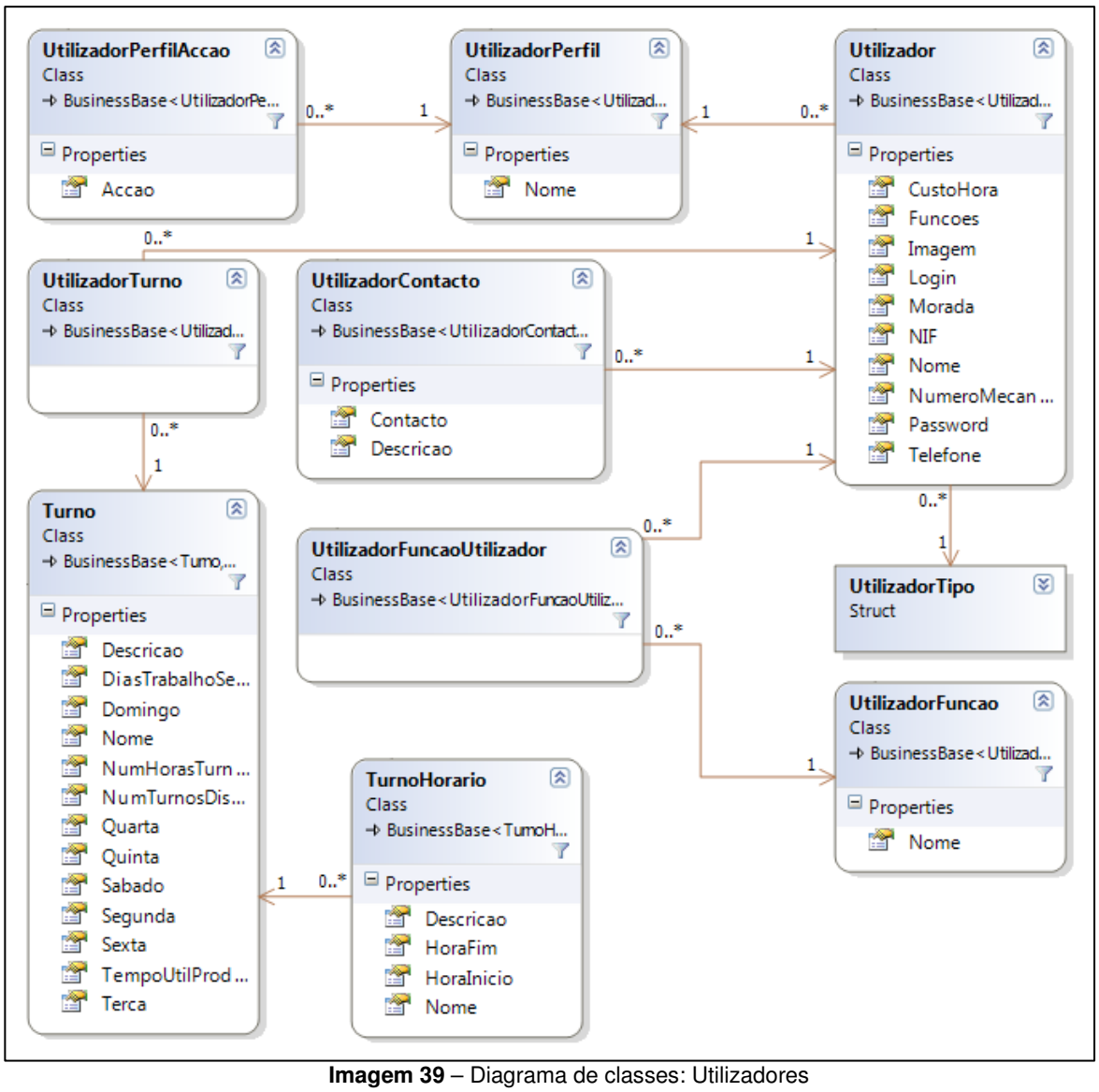


Imagem 39 – Diagrama de classes: Utilizadores

Diagrama físico:

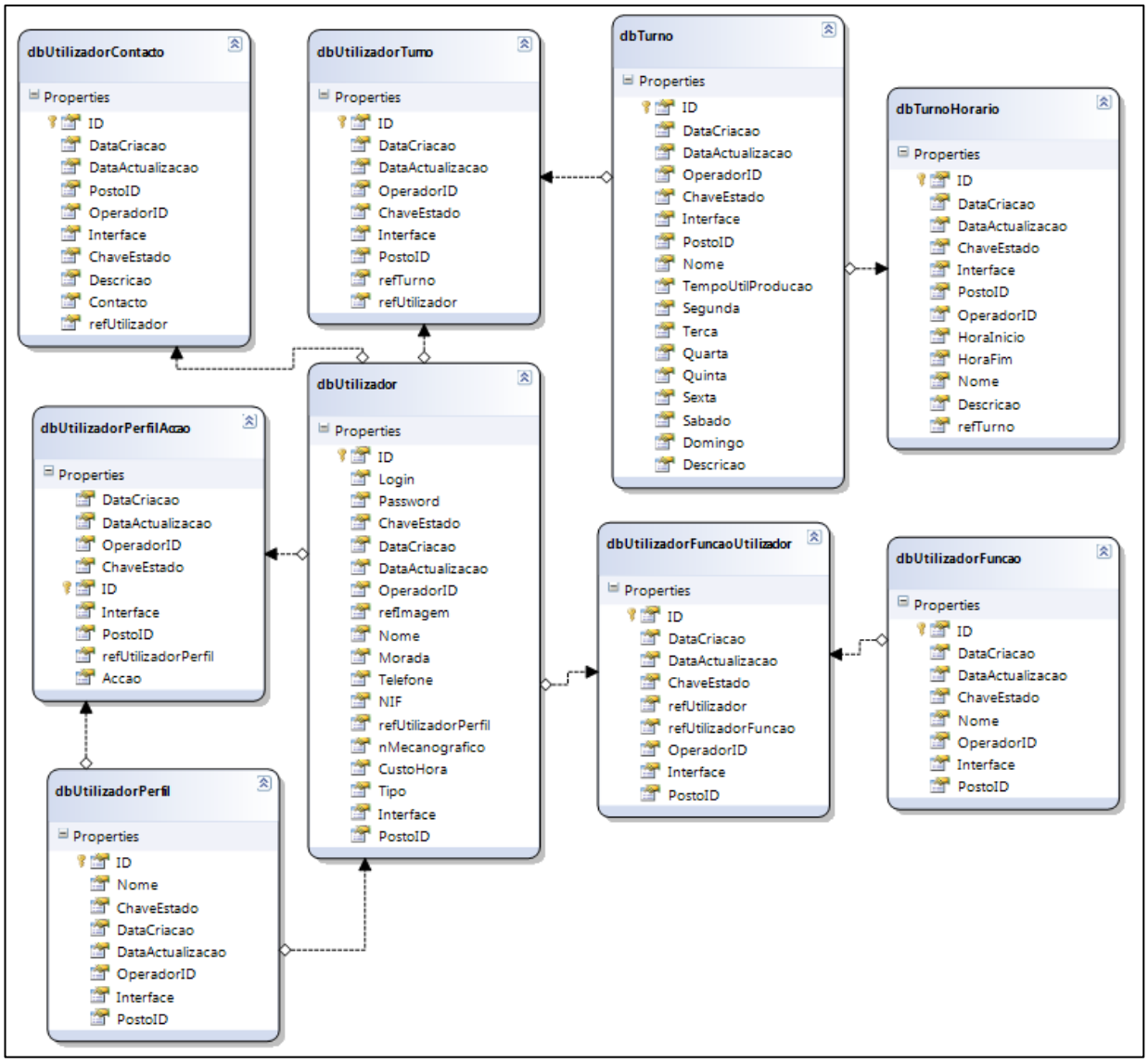


Imagem 40 – Diagrama físico: Utilizador

4.4.5. Espaços

A gestão de espaços pode ser outro ponto de expansão no futuro, com a implementação de gestão de transportes, rotas de abastecimento interno, tempos de entrega e análise aos movimentos de stock.
No contexto da solução actual pretende-se ter a informação bem organizada, permitindo saber onde estão os diferentes elementos.

Diagrama de classes:

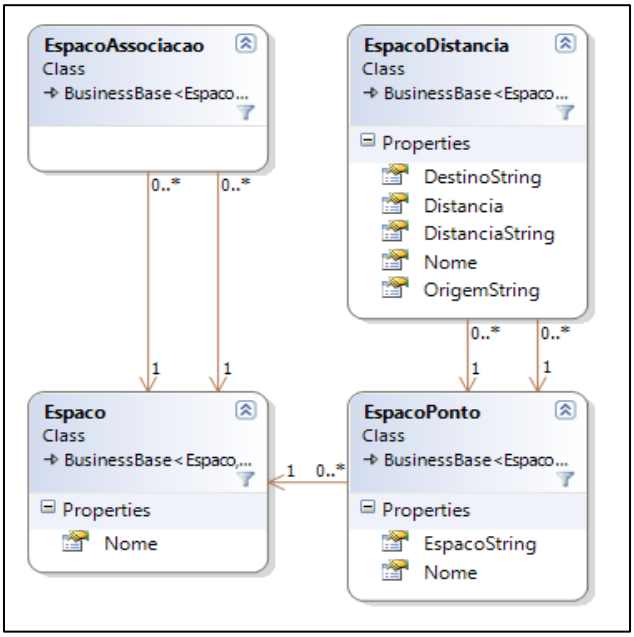


Imagem 41 – Diagrama de classes: Espaços

Diagrama físico:

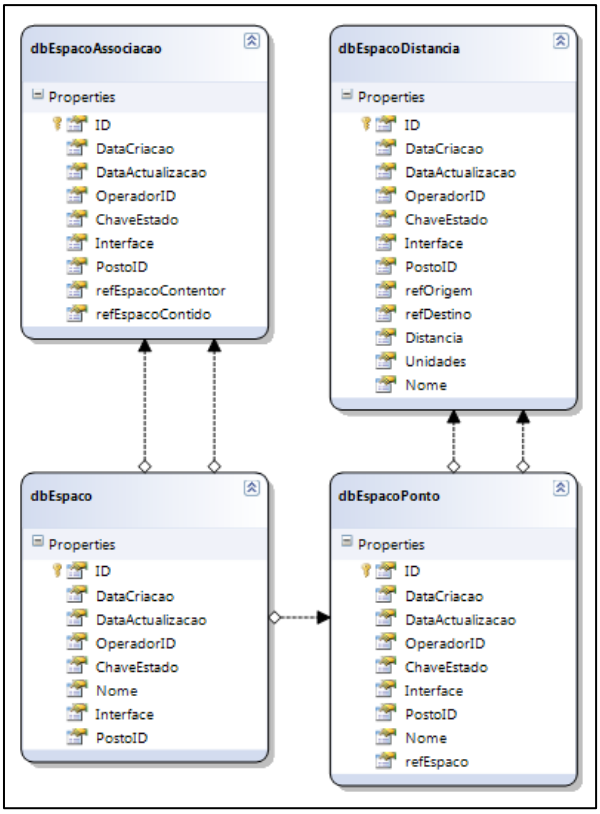


Imagem 42 – Diagrama físico: Espaços

4.4.6. Armazenamento

Em termos de estrutura é simples, mas uma correcta configuração é necessária para se saber a localização dos produtos e se a capacidade de armazenamento está ou não a ser excedida.

Diagrama de classes:

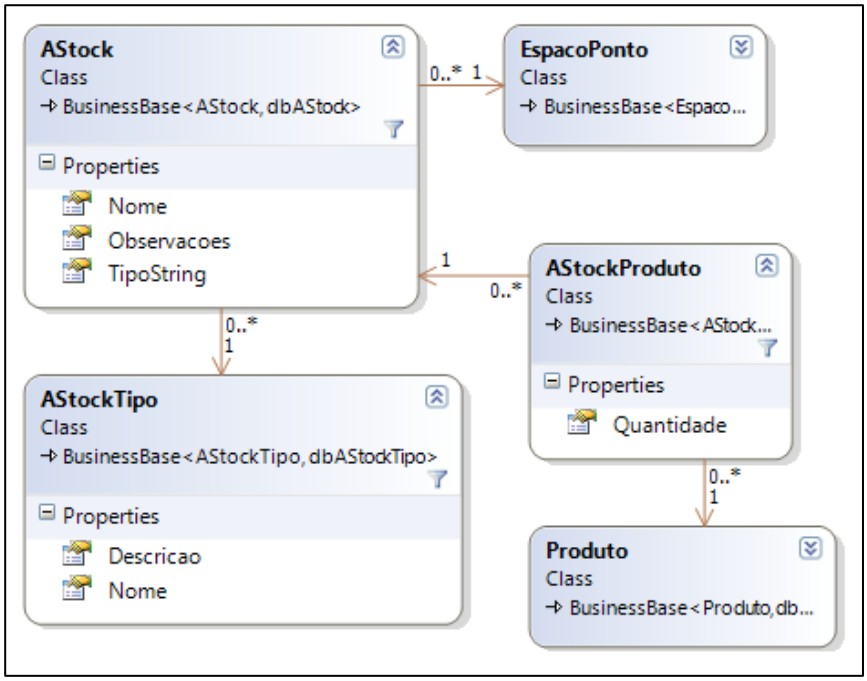


Imagem 43 – Diagrama de classes: Armazenamento

Diagrama físico:

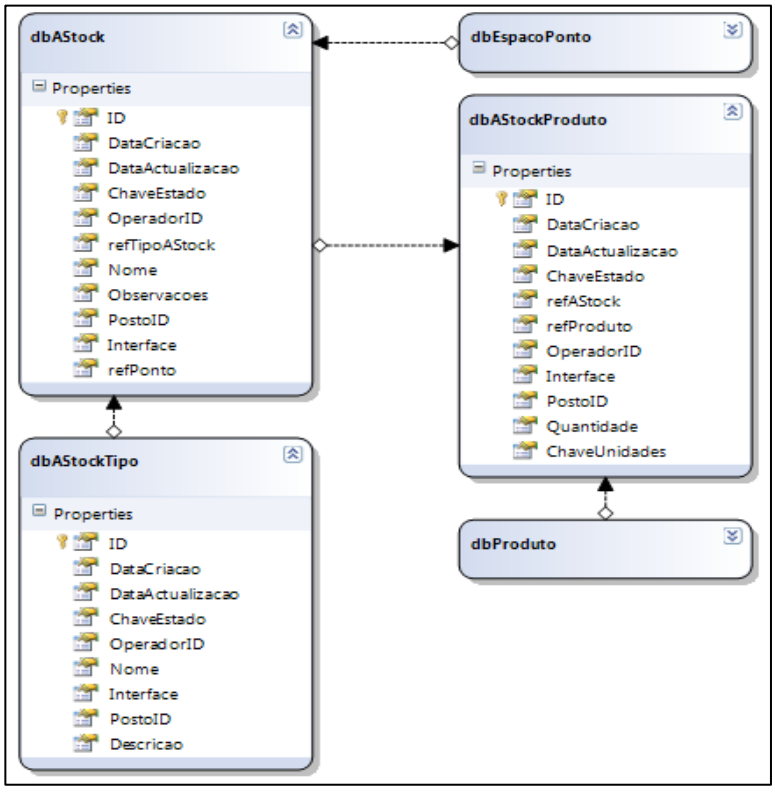


Imagem 44 – Diagrama físico: Armazenamento

4.4.7. Picagem

A situação mais complexa é o registo de evolução, ou seja, a picagem propriamente dita.
A classe OrdemTrabalhoDetalhe representa o registo da picagem, no qual se pode identificar o colaborador e a quantidade produzida pelo mesmo, ou, então, apenas a quantidade produzida.
O registo da Ordem trabalho será marcado como "concluída", quando a quantidade for igual à soma dos registos de OrdemTralhoDetalhe.

Diagrama de classes:

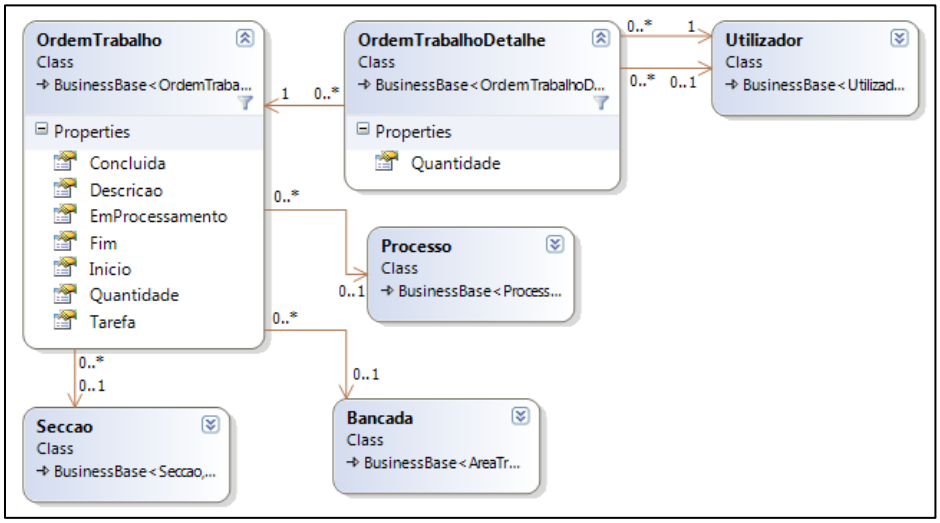


Imagem 45 – Diagrama de classes: Picagem

Diagrama físico:

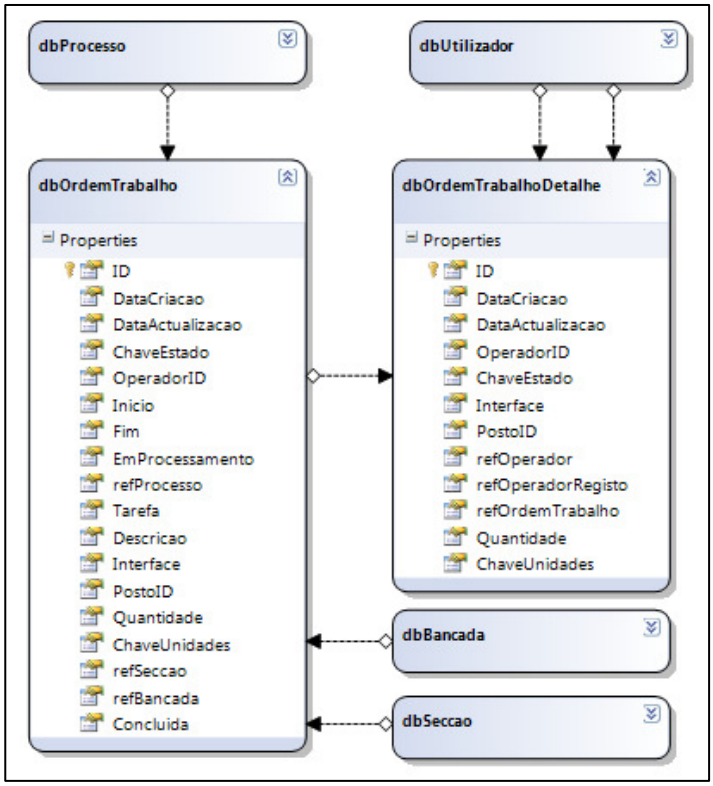


Imagem 46 – Diagrama físico: Picagem

Parceiros

A gestão de parceiros não é um dos objectivos da solução apresentada, mas tem de existir um mínimo para permitir determinadas tarefas.
Além disso, poderá servir de ponte para outras ferramentas que, essas sim, têm o objectivo de gestão de clientes. Por exemplo: softwares CRM – Customer Relationship Management, ou de fornecedores.

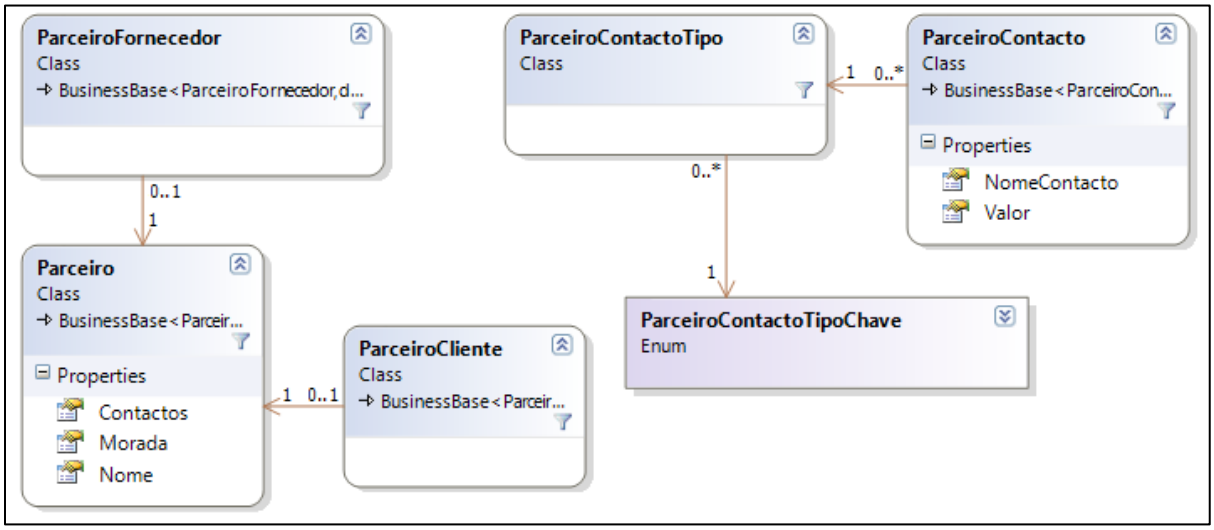


Imagem 48 – Diagrama de classes: Parceiros

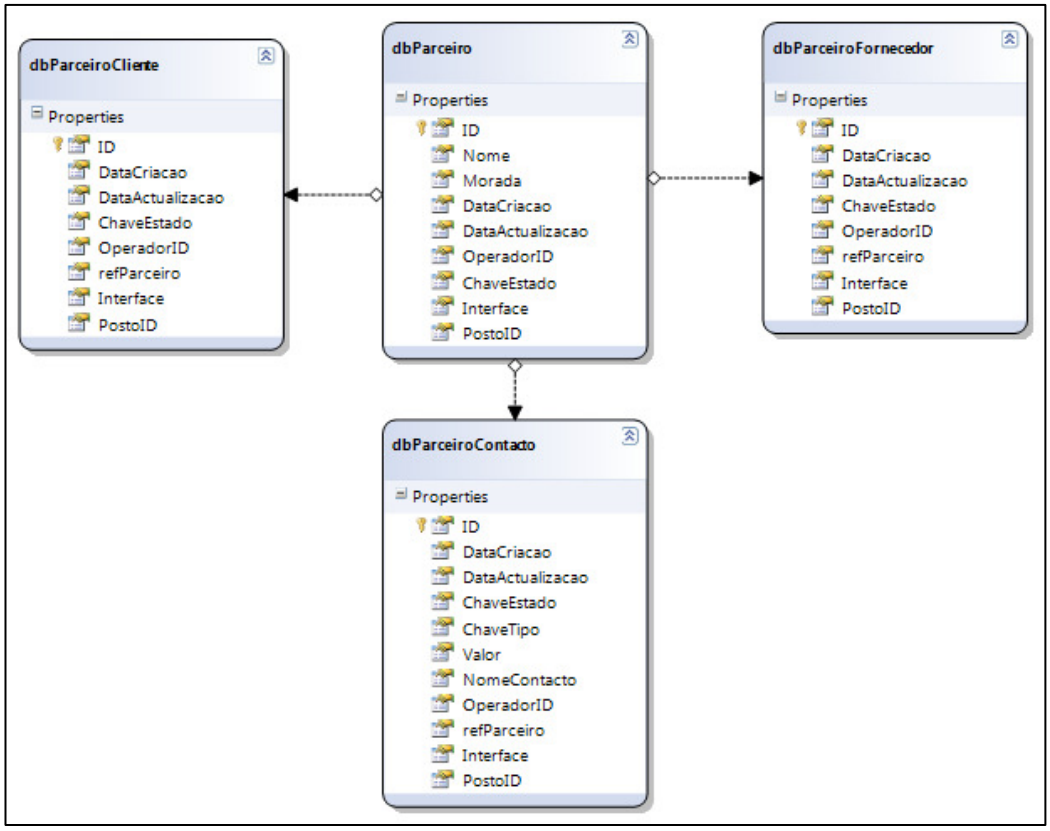


Imagem 47 – Diagrama físico: Parceiros

4.4.8. Segurança

A configuração de segurança tem dois grandes objectivos:

- ➔ Apenas pessoas com permissão acedem ao sistema e a determinadas funcionalidades, que estão associadas ao perfil de utilizador.
- ➔ Só em computadores que estejam configurados é que se consegue iniciar a aplicação e também aceder às funcionalidades permitidas no mesmo.

As permissões do posto são carregadas quando a aplicação inicia e a do utilizador, após o mesmo se ter validado no sistema:

Diagrama de classes:

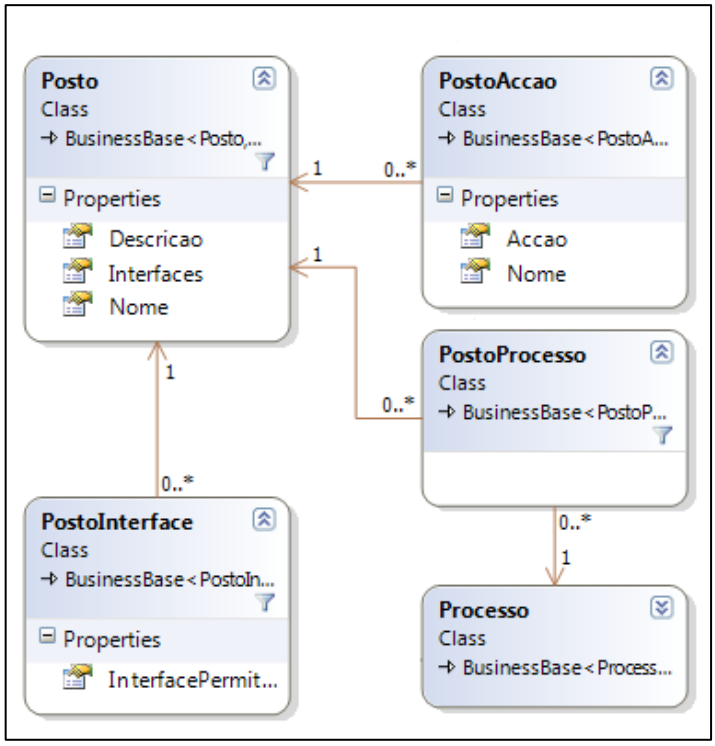


Imagem 49 – Diagrama de classes: Segurança

Diagrama físico:

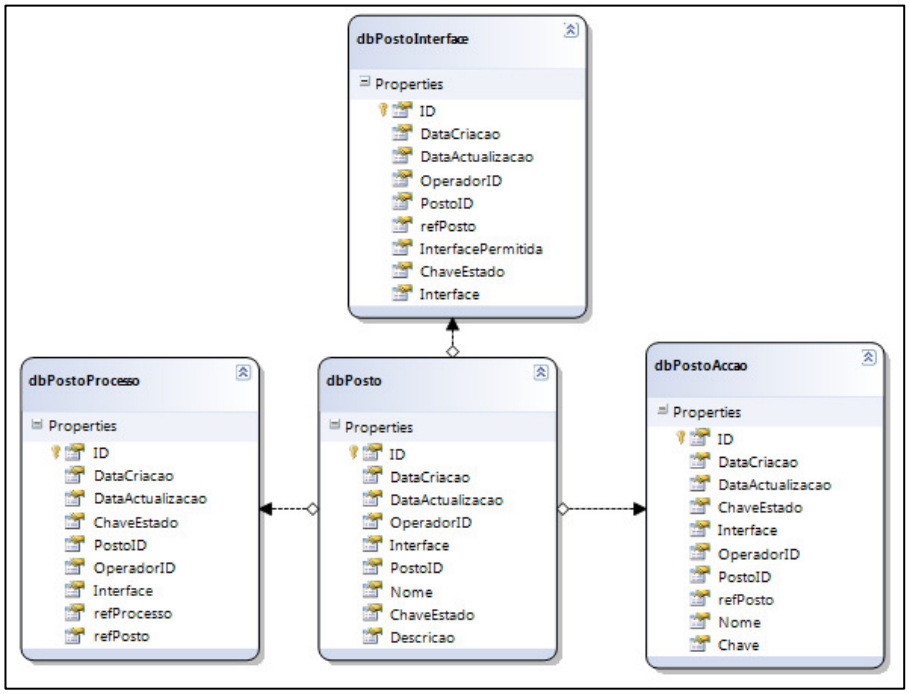


Imagem 50 – Diagrama físico: Segurança

4.4.9. Sistema

Os alertas permitem enviar avisos às pessoas que se encontrem configuradas em sistema.
As condições em que este registo acontece são, também, configuráveis, estando esta solução limitada a avarias de equipamento, excesso de stock na área de armazenamento ou grande diferença entre a produção esperada e a real.
Os registo automático de entrada de sessão e dos elementos dos registos (data de criação, data de actualização, posto e operador) estão explicitados no ponto 6.3.1. e, por isso, não há referências dos mesmos no diagrama de classes seguinte:

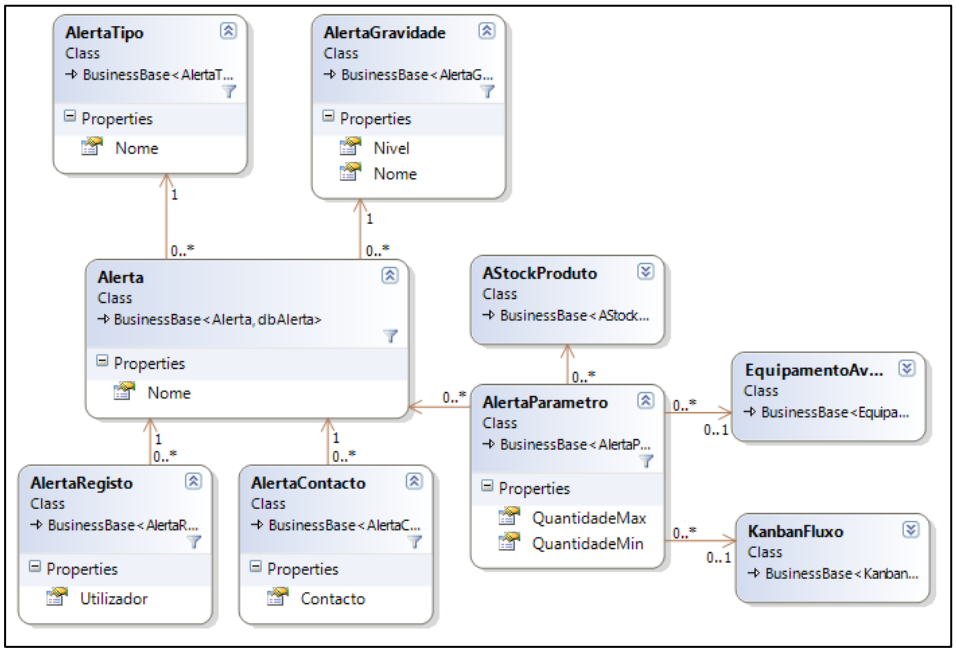


Imagem 51 – Diagrama de classes: Sistema

Diagrama físico:

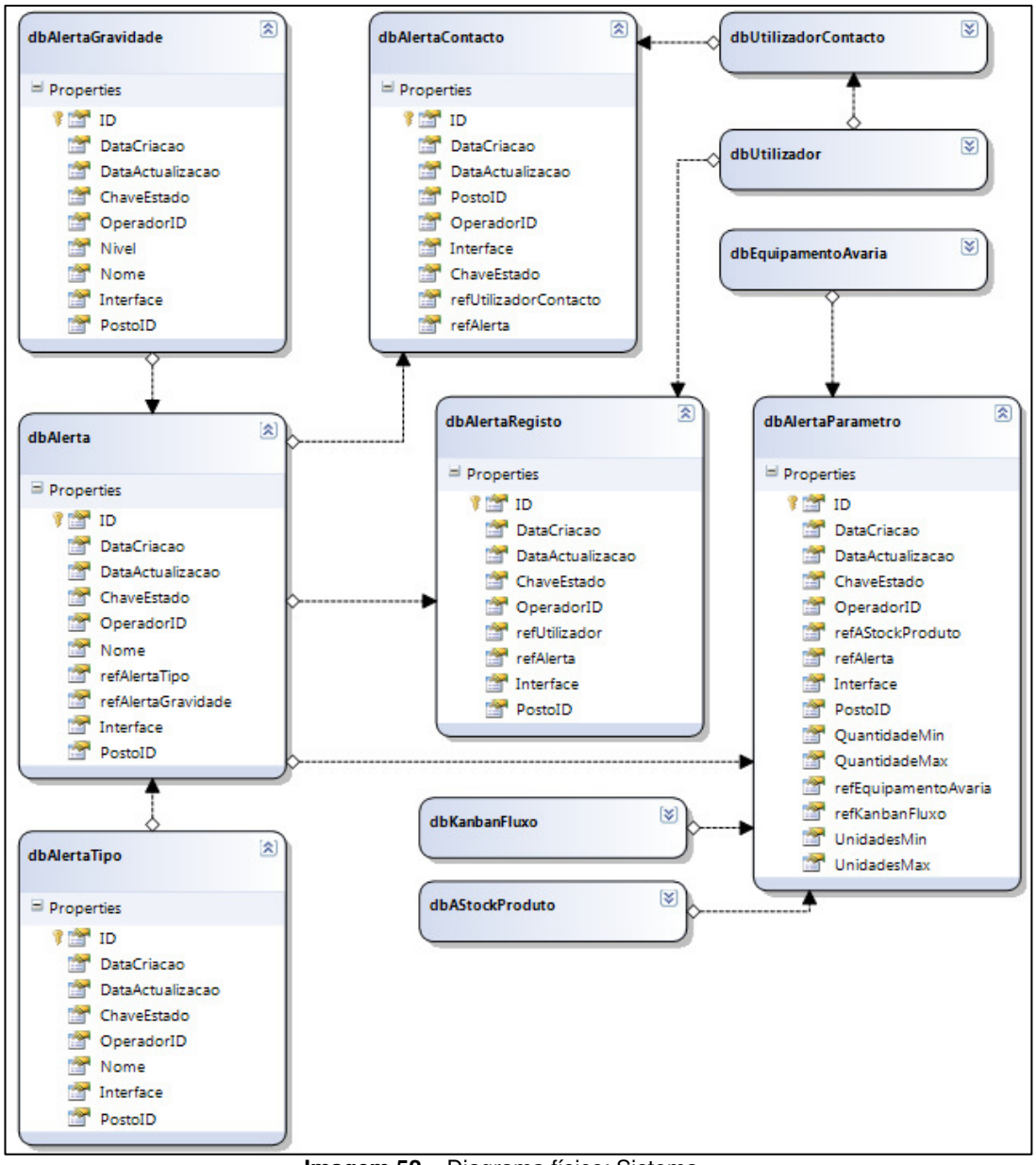


Imagem 52 – Diagrama físico: Sistema

4.5. Diagrama de sistema

A proposta de desenho do sistema foi pensada para poder satisfazer várias necessidades, oferecendo interfaces diferentes mas que usam a mesma lógica.

No servidor recomenda-se que estejam presentes, tanto os serviços, como a base de dados. No entanto, caso se pretenda, os mesmos (base de dados e serviços) poderão estar em máquinas diferentes.

Para permitir integração mais rápida com hardware que se pretenda vir a usar, o emprego de uma aplicação é o mais adequado.

A interface Web aparece no diagrama como ferramenta para os parceiros, mas, caso se pretenda, poderá também ser desenhada uma interface Web de gestão, para que os gestores possam aceder sem necessidade de uma ligação vpn.

O que foi referido encontra-se apresentado de forma simples no diagrama seguinte:

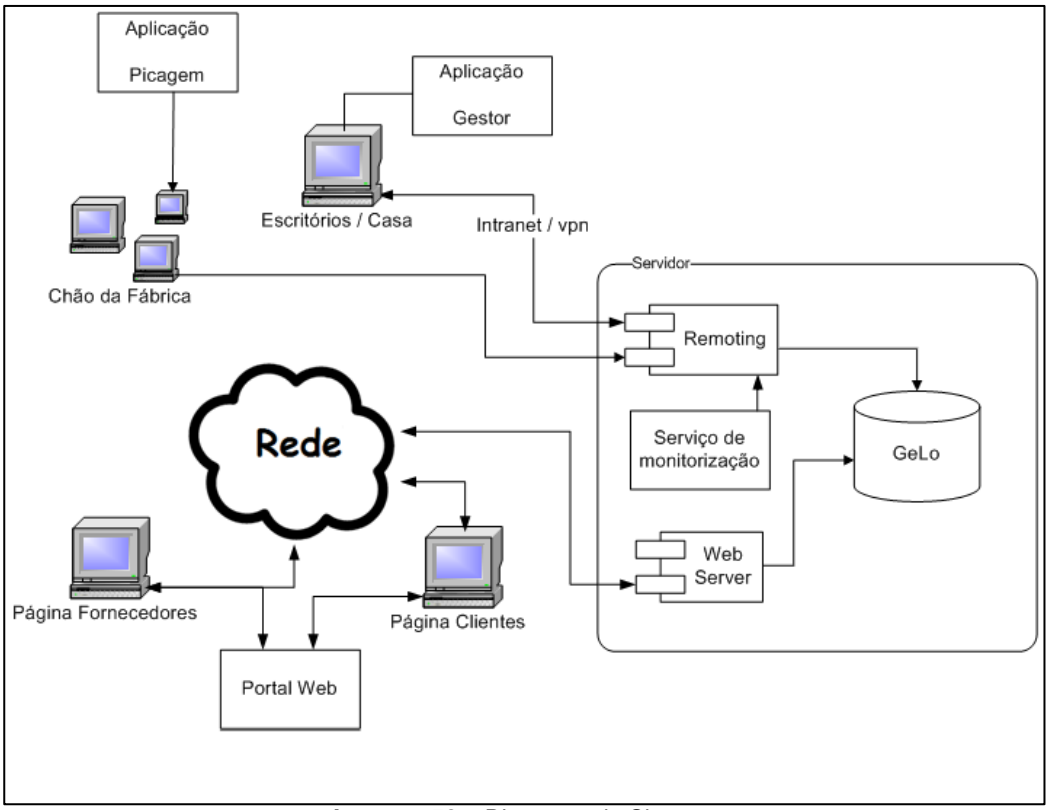


Imagem 53 – Diagrama de Sistema

Há requisitos funcionais que necessitam que exista um processo autónomo, que tenha ligação permanente à base de dados. Por uma questão de performance, recomenda-se que este serviço se encontre na mesma máquina em que está a base de dados.

Este serviço irá ter dois tipos de funções:

- ➔ Monitorizar, verificando se ocorrem determinadas condições previamente configuradas e, em caso positivo, lançar alertas.
- ➔ Nos casos de actualizações estatísticas deverá, nas alturas programadas, despoletar esta acção.

4.6. Proposta de interfaces

Para não causar um impacto negativo, um dos aspectos importantes de um software é ter uma aparência simples, desprovida de muitas opções, logo à partida.
Mesmo que o programa possa ser complexo e possa ter, efectivamente, muitas opções, só o facto das mesmas estarem bem organizadas dá ao utilizador a sensação de simplicidade. Evita, assim, o sentimento de um uso algo penoso.

De conformidade com o referido nos parágrafos anteriores, iremos ver, na imagem seguinte, o interface proposto, com uma apresentação simples, mas com as ferramentas necessárias à distância de um clique:

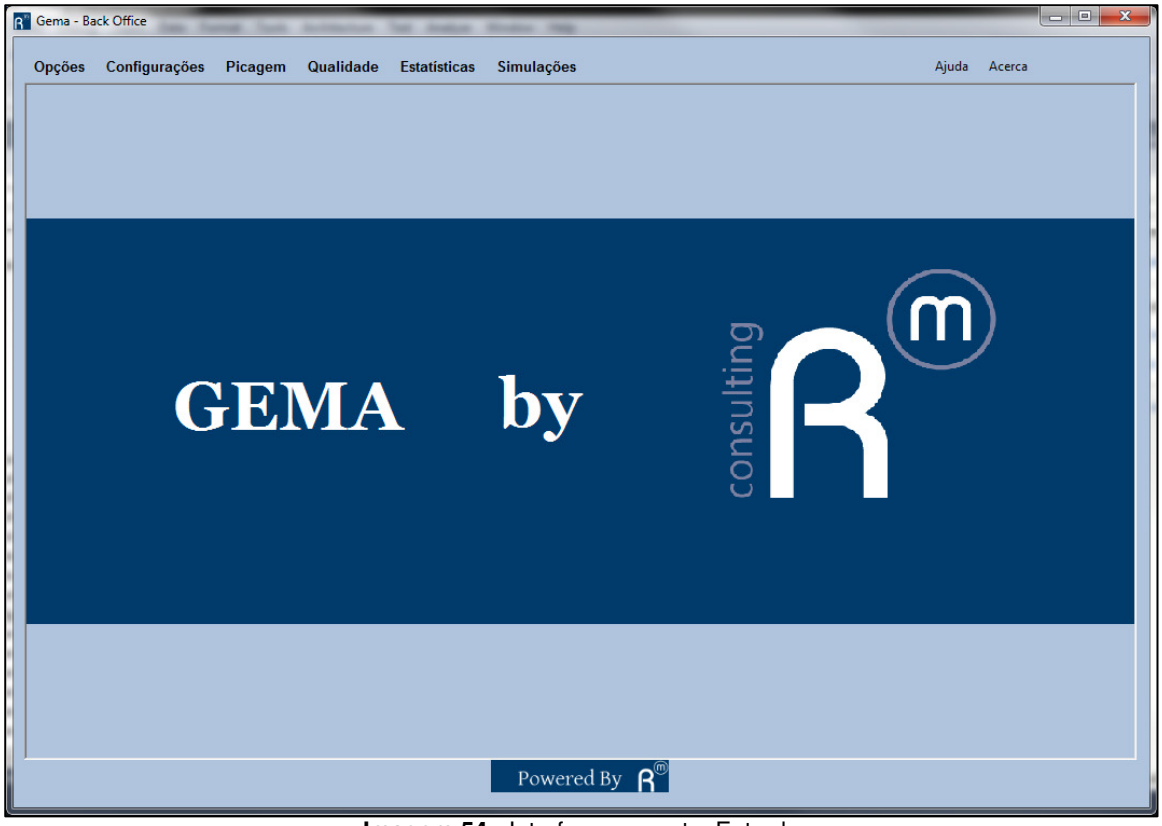


Imagem 54 - Interface proposta: Entrada

Ao entrar em qualquer opção, o fundo é substituído pelo painel do item escolhido, podendo, depois, o utilizador regressar ao painel de entrada.

O painel de entrada, neste caso é estático, mas poderá evoluir para um mais dinâmico, com informações e opções úteis ao utilizador.

A imagem seguinte mostra a interface proposta para as configurações

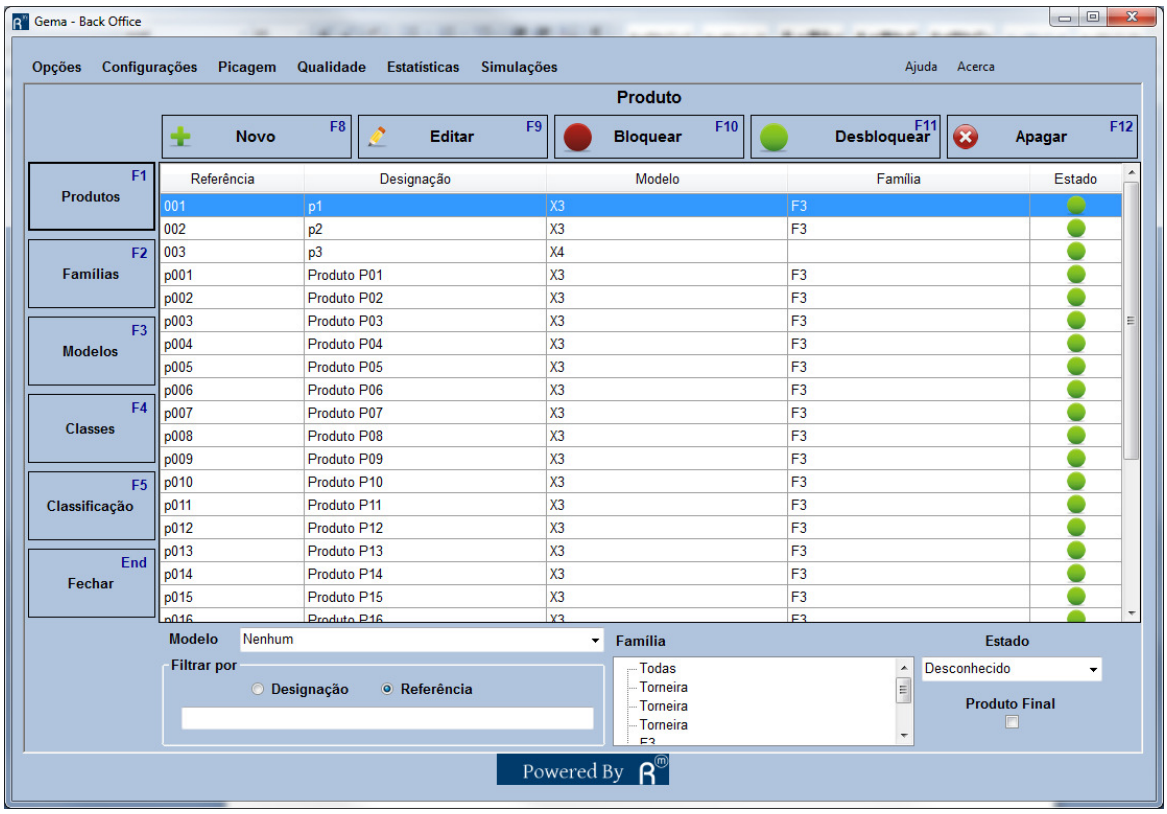


Imagem 55 – Interface proposta: Configurações

No caso da interface de configurações é apresentada uma lista com todos os elementos configurados em sistema, que não estejam marcados como "apagados".
Em cima encontram-se as operações directas que se podem executar; no lado esquerdo aparecem os elementos que pertencem à configuração dos produtos; em baixo podemos filtrar a lista que nos é apresentada.

Na interface proposta para a picagem desaparecem os botões em cima, passando as opções para o lado esquerdo. A lista será das ordens de trabalho que ainda não estão terminadas ou canceladas. Esta, dependendo das permissões do utilizador e do posto, poderá ser filtrada logo á partida.

A interface para a qualidade, difere da interface da Picagem na lista e nas opções laterais.
A grelha apresenta a lista dos stocks, onde se encontram produtos rejeitados e que ainda não foram triados, podendo ser identificados como "retrabalho" ou "sucata".

Para a estatística é um pouco diferente. Apresenta uma tabela com os dados constantes em base de dados e disponibiliza apenas um botão para correcção de algum dado, que tenha sido determinado como incorrecto.

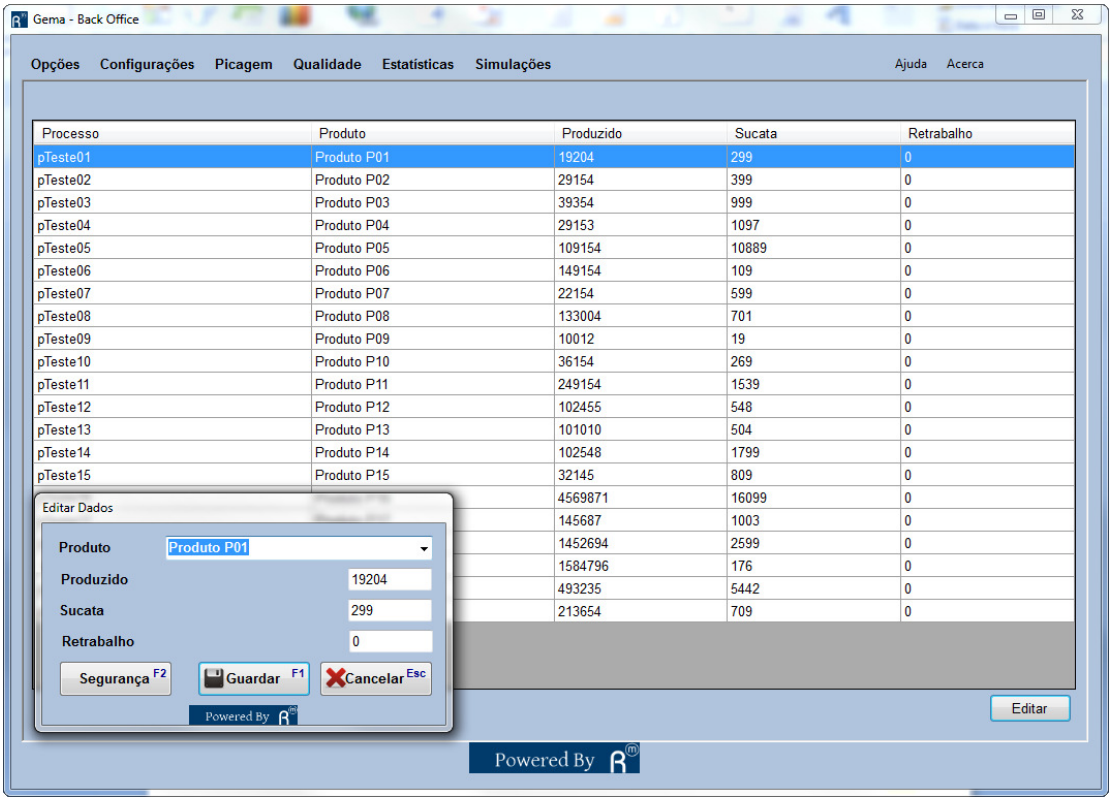


Imagem 56 - Interface proposta: Estatísticas

No caso da simulação a interface muda bastante, como mostra a imagem que se segue:

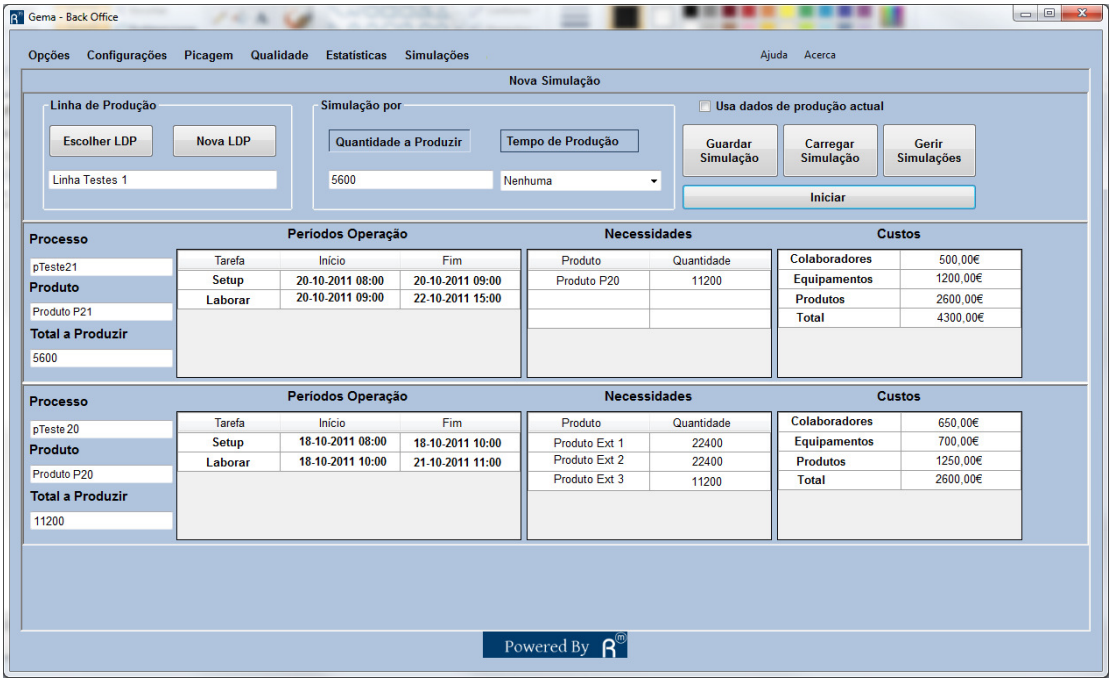
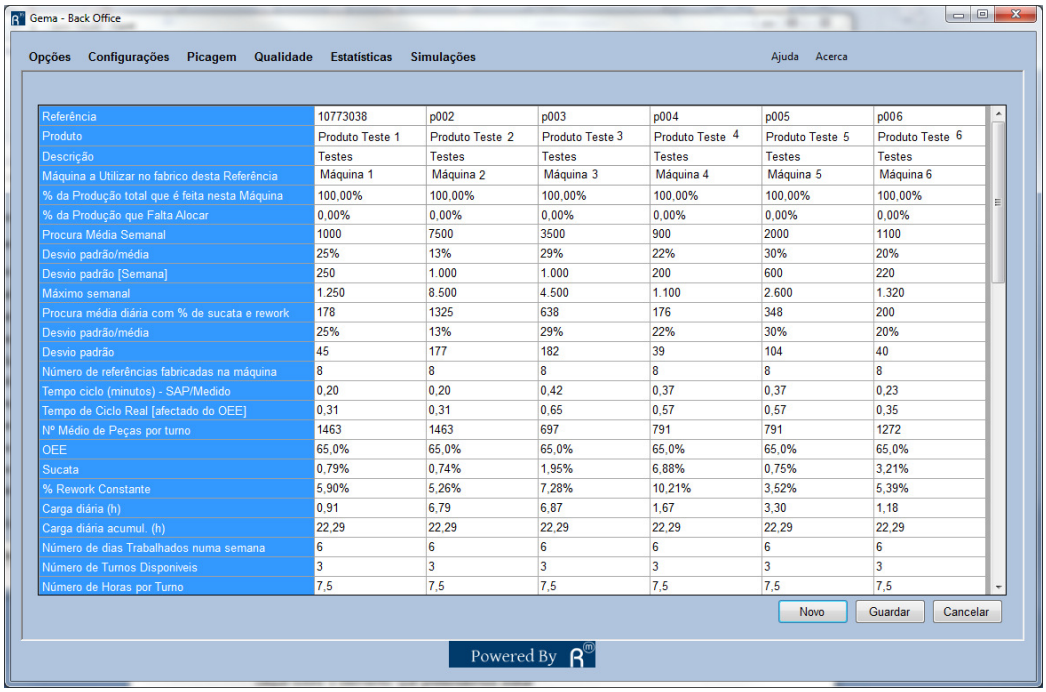


Imagem 57 – Interface proposta: Simulação

Em cima temos as diferentes opções para a simulação.
Realizada a simulação, o painel é populado com elementos que, de forma simples, mostram toda a informação necessária.

Finalmente, em baixo pode ver-se a interface do cálculo da quantidade de kanbans:



Referência	10773038	p002	p003	p004	p005	p006
Produto	Produto Teste 1	Produto Teste 2	Produto Teste 3	Produto Teste 4	Produto Teste 5	Produto Teste 6
Descrição	Testes	Testes	Testes	Testes	Testes	Testes
Máquina a Utilizar no fabrico desta Referência	Máquina 1	Máquina 2	Máquina 3	Máquina 4	Máquina 5	Máquina 6
% da Produção total que é feita nesta Máquina	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%
% da Produção que Falta Alocar	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%
Procura Média Semanal	1000	7500	3500	900	2000	1100
Desvio padrão/média	25%	13%	29%	22%	30%	20%
Desvio padrão [Semana]	250	1.000	1.000	200	600	220
Máximo semanal	1.250	8.500	4.500	1.100	2.600	1.320
Procura média diária com % de sucata e rework	178	1325	638	176	348	200
Desvio padrão/média	25%	13%	29%	22%	30%	20%
Desvio padrão	45	177	182	39	104	40
Número de referências fabricadas na máquina	8	8	8	8	8	8
Tempo ciclo (minutos) - SAP/Medido	0,20	0,20	0,42	0,37	0,37	0,23
Tempo de Ciclo Real [afectado do OEE]	0,31	0,31	0,65	0,57	0,57	0,35
Nº Médio de Peças por turno	1463	1463	697	791	791	1272
OEE	65,0%	65,0%	65,0%	65,0%	65,0%	65,0%
Sucata	0,79%	0,74%	1,95%	6,88%	0,75%	3,21%
% Rework Constante	5,90%	5,26%	7,28%	10,21%	3,52%	5,39%
Carga diária (h)	0,91	6,79	6,87	1,67	3,30	1,18
Carga diária acumul. (h)	22,29	22,29	22,29	22,29	22,29	22,29
Número de dias Trabalhados numa semana	6	6	6	6	6	6
Número de Turnos Disponiveis	3	3	3	3	3	3
Número de Horas por Turno	7,5	7,5	7,5	7,5	7,5	7,5

Imagem 58 – Interface proposta: Cálculo quantidade Kanbans

Ao contrário das listas normais, em que a cada novo registo é adionada uma nova linha, aqui, em cada novo registo é criada uma nova coluna. A ideia é que se consiga ter uma boa descrição, o que em linhas seria problemático.
Além da criação de um novo elemento de cálculo, possibilita a edição, fazendo duplo clique na coluna pretendida.
De forma a evitar erros do utilizador, apenas serão adicionadas ou alteradas as acções, quando se escolher “guardar”.

5. Conclusão

A solução apresentada cumpre, praticamente, todos os objectivos inicialmente propostos. O software permite gerir os diversos elementos que, de alguma forma, participam no processo produtivo, bem como gerir o uso de algumas ferramentas da produção magra, como é o caso do cálculo da quantidade de kanbans.

No inicio o trabalho foi estruturado e dividido em três fases:

- **Pesquisa Bibliográfica**

Após longa pesquisa na internet, conseguiu-se um conjunto de referências das quais foi extraído um resumo dos princípios e metodologias de produção magra. Há ainda diversos aspectos que poderiam ser referidos, mas para o objectivo que se pretende não eram importantes.

- **Ferramentas Concorrentes**

Nas ferramentas concorrentes, das soluções concorrentes duas mostraram-se muito adequadas ao que era proposto para este trabalho. Contudo há diferenças que se pretendiam ser aplicadas e que também estas foram alcançadas.
Contudo a diferenciação vai sempre depender de até onde se pretender ir.

- **Modelação do sistema**

A solução apresentada está pensada para poder usar os recursos já existentes, evitando, assim, despesas na aquisição de novos equipamentos, a menos que os mesmos sejam imprescindíveis. Se necessário, poderá correr numa máquina comum, desde que equipada com o sistema operativo Microsoft Windows Xp, Windows Vista ou Windows 7.

- **Estrutura de dados**

A solução apresenta uma estrutura modular e escalável. O objectivo passa pela possibilidade de integração de mais ferramentas e de outros elementos, que se pretenda que sejam geridos.

Neste momento permite:

- Gerir Produtos
- Gerir Equipamentos
- Gerir Processos
- Gerir Operadores
- Gerir Armazenamentos
- Gerir Transportes
- Gerir Ordens de Trabalho
- Gerir Bancadas
- Gerir Segurança do Sistema
- Gerir e Lançar Alertas

❖ Ferramentas implementadas

Nos objectivos traçados existem duas ferramentas que foram implementadas com êxito. Embora ainda de forma simples, as mesmas permitem realizar a gestão de produção com uma filosofia lean:

- Realizar Cálculo Kanban:

  O cálculo é realizado usando uma grelha, como podemos verificar na imagem 58.

  A criação um novo registo pressupõe o correcto preenchimento dos campos do formulário apresentado a seguir:

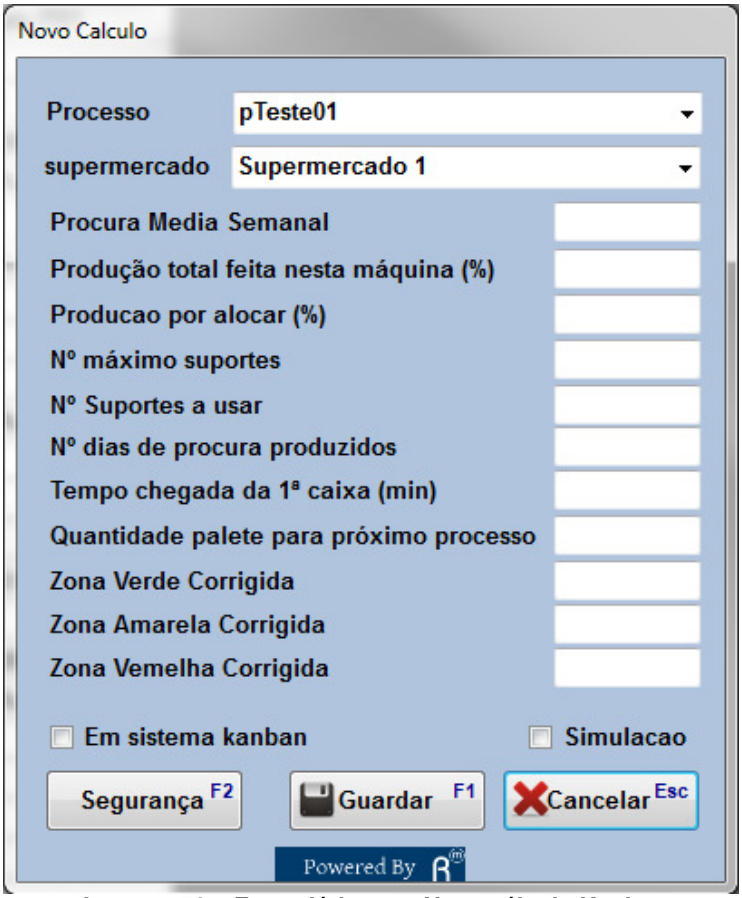


Imagem 59 – Formulário para Novo cálculo Kanban

O sistema permite ainda alterar definições. Para o efeito, faz-se duplo clique no registo pretendido e realiza-se novo cálculo de quantidade.

Possibilita, ainda, guardar registos simulados sem que estes afectem os cálculos reais.

- Realizar Simulações, Necessidades e Custos

A simulação permite ver toda a informação da linha de produção, indicando a quantidade que se pretende simular.
Existem dois modos de simulação:

O primeiro modo não tem em conta a concorrência, ou seja, não verifica a ocupação dos equipamentos e bancadas. Isto irá permitir detectar a existência de engarrafamentos.

- Este primeiro modo encontra-se implementado pelo uso do método ArvoreDependenciasProcessos na classe Processo.

O segundo modo tem em conta a ocupação. Olha também para as ordens de trabalho, que se encontram em espera, para produção. O objectivo é conhecer a capacidade e prazos necessários para produzir a quantidade pretendida.
A complexidade deste cálculo e o tempo disponível não permitiu a sua implementação.

❖ Ideia final

Como já foi referido, foram atingidos os principais objectivos, inicialmente previstos.
A totalidade dos mesmos revelou-se demasiado ambiciosa para o tempo disponível, mas a solução apresentada está assente numa estrutura modular, que permite expandir-se até ao que não foi atingido, bem como a outras ferramentas que se pretendam vir a implementar.

Pela dimensão já atingida, entendo que o desenvolvimento deste software não poderá ser tão abrangente.
Implicaria que os respectivos custos se tornassem elevados e, consequentemente, que os preços para as empresas clientes não fossem tão acessíveis quanto o desejado.

6. Bibliografia:

Mirsky, Jeannette – Eli Whitney [Consulta 15-10-2011]. Disponível em WWW: http://www.britannica.com/EBchecked/topic/642887/Eli-Whitney

Papesh, Mary Ellen – Frederick Wonslow Taylor [Consulta em 15-10-2011]. Disponível em WWW: http://www.stfrancis.edu/content/ba/ghkickul/stuwebs/bbios/biograph/fwtaylor.htm

Gelderman, Carol W. – Henry Ford, Biography [Consulta em 17-10-2011]. Disponível em WWW: http://www.biography.com/people/henry-ford-9298747?page=5

Lee, Quarterman & Snyder, Brad - **Value stream & process mapping: genesis of manufacturing strategy**. Bellingham: Enna Products Corporation, 2006. ISBN 9781897363430

Womak, James P. & Jones, Daniel T. & Roos, Daniel - **The Machine That Changed the World**. Simon and Sghuster, 2007. ISBN 9780743299794

Liker, Jeffrey & Meier, David - **The Toyota Way Fieldbook** - McGraw-Hill, 2005. ISBN 9780071448932

Eng. António Marques – **Lean Manufacturing.** [PDF] .[Aveiro]: Universidade de Aveiro.

Eng. António Marques – **JIT, Lean Management, Six Sigma.** [PDF] .[Aveiro]: Universidade de Aveiro.

Sebrosa, Rui – Produção Magra na Indústria Gráfica [Consulta em 02-11-2011]. Disponível em WWW:
http://portaldasartesgraficas.com/artigos/rui_sebrosa_1.htm

Silva, J.P. Rodrigues da – Técnicas de ferramentas Lean V1-2008 [Consulta em 21-10-2011]. Disponível em WWW: http://pt.scribd.com/doc/3500513/Lean-Manufacturing-3Tecnicas-e-ferramentas

Silva, J.P. Rodrigues da – TPM – Formação Básica Pilar 1 [Consulta em 22-10-2011]. Disponível em WWW: http://pt.scribd.com/doc/2537982/5SO-Pilar-Basico-do-TPM

Sousa, Francisco & Almeida, Nuno – O Conceito Lean na indústria Gráfica (2008)[Consulta em 21-10-2011]. Disponível em WWW:
http://portaldasartesgraficas.com/artigos/isec4.htm

Tadashi, Odier – Estabilidade é a base para o sucesso da produção lean (2010)[Consulta em 21-10-2011]. Disponível em WWW:
http://giroconsultoria.blogspot.com/

Robinson, Harry – Using Poka-Yoke Techniques for Early Defect Detection. [Consulta em 21-10-2011]. Disponível em WWW: http://facultyweb.berry.edu/jgrout/pokasoft.html

Luna, Adriano Fernandes – A Aplicação do Just-In-Time na Manufatura Enxuta (Lean Manufacturing). [Consulta em 22-10-2011]. Disponível em WWW: http://www.administradores.com.br/informe-se/artigos/a-aplicacao-do-just-in-time-na-manufatura-enxuta-lean-manufacturing/21090/

McBride, David – Heijunka: Level the Load. [Consultado em 22-10-2011]. Disponível em WWW: http://www.emsstrategies.com/dm090804article.html

Suzaki, Kiyoshi – **Gestão de Operações LEAN**. 1ª ed. Mansores: LeanOp Press, 2010. ISBN: 978-989-20-2084-6.

Castiglioni, Jose Antonio de Matos – **Logística Operacional: Guia Prático**. 2ª ed. São Paulo: Editora Érica Ltda, 2009. ISBN 978-85-365-0181-9

Bösenberg, Dirk & Metzen, Heinz - **Como Aligeirar Estruturas e Custos –** Edição CETOP, 1999. ISBN 9789726413356

Courtois, A. & Martin-Bonnefous, C & Pillet, Maurice – **Gestão da Produção**. 5ª Edição Lidel. ISBN 9789727574698

Tuppas, Manufacturing Software. [Consultado em 02-11-2011]. Disponível em WWW: http://www.tuppas.com/Manufacturing-software/Manufacturing-software.htm

Sage, Sage ERP x3. [Consultado em 02-11-2011]. Disponível em WWW: http://www.sageerpx3.com/pt/

Alidata, SIA – ERP. [Consultado em 02-11-2011]. Disponível em WWW: http://www.alidata.pt/software/sia-erp/

Coolsoft, Gestão de Produção. [Consultado em 02-11-2011]. Disponível em WWW: http://www.coolsoft.com.pt/software/GestaoProducao.asp?ID=34

Weber Systems Inc., Lean ERP System. [Consultado em 02-11-2011]. Disponível em WWW: http://www.lean-manufacturing-inventory.com/pur_pricelist.aspx

Plex, Cloud ERP Manufacturing Software. [Consultado em 02-11-2011]. Disponível em WWW: http://www.plex.com/modules/software_areas.asp

QAD, Manufacturing. [Consultado em 02-11-2011]. Disponível em WWW: http://www.qad.com/erp/Solutions/QAD+Enterprise+Applications

Global Shop Solutions, One-System ERP™. [Consultado em 02-11-2011]. Disponível em WWW: http://globalshopsolutions.com/erp_software.cfm?id=9

Microsoft, Microsoft Dynamics ERP. [Consultado em 02-11-2011]. Disponível em WWW: http://www.microsoft.com/en-us/dynamics/erp-deployment.aspx

ARP Sistemas Informáticos, ERP GIN. [Consultado em 02-11-2011]. Disponível em WWW: http://www.apr.pt/gin/areas.aspx/-1/areas%20verticais/

Sistrade, Gestão da Produção. [Consultado em 02-11-2011]. Disponível em WWW:http://www.sistrade.pt/pt/Solucoes/MIS-ERP-Sistrade-gestao-producao.htm

IfThen, IfProd [Consultado em 02-11-2011]. Disponível em WWW: http://www.ifthensoftware.com/ProdutoX.aspx?ProdID=6

Hibbs, Curt & Jewett, Steve & Sullivan, Mike - **The Art of Lean Software Development**. O'Reilly, 2009. ISBN 978-0-596-51731-1

Poppendieck, Mary & Poppendieck, Tom – **Lean Software Development: An Agile Toolkit.** Addison Wesley, 2003. ISBN 978-0321150783

Poppendieck, Mary & Poppendieck, Tom – **Implementing Lean Software Development From Concept to Cash. Addison Wesley Professional, 2006.** ISBN 978-0-321-43738-9

Craig, the V-Model: the Development Process. [Consultado em 03-11-2011]. Disponível em WWW: http://www.betterprojects.net/2007/07/v-model-development-process.html

Boehm, 1988. Spiral model. [Consultado em 03-11-2011]. Disponível em WWW: http://en.wikipedia.org/wiki/File:Spiral_model_(Boehm,_1988).svg

Chapman, James R. - Software Development Methodology. [Consultado em 25-10-2011]. Disponível em WWW: http://www.hyperthot.com/pm_sdm.htm

Bowman, David. What is a Prototyping Methodology?. [Consultado em 03-11-2011]. Disponível em WWW: http://www.information-management-architect.com/prototyping-methodology.html

Agile Alliance, The Agile Manifesto. Consultado em 03-11-2011]. Disponível em WWW: http://www.agilealliance.org/the-alliance/the-agile-manifesto/

Spence, Ian & Bittner, Kurt. What is iterative development? [Consultado em 03-11-2011]. Disponível em WWW: http://www.ibm.com/developerworks/rational/library/may05/bittner-spence/

Apêndice I - IDBoject

```
public interface IDBObject
{
    long ID {get;set;}
    DateTime DataCriacao { get; set; }
    DateTime DataActualizacao { get; set; }
    long OperadorID { get; set; }
    int ChaveEstado { get; set; }
    int Interface { get; set; }
    long PostoID { get; set; }

}
```

Apêndice II – DBLinq

```
public abstract class DBLinq<T> where T : class,IDBObject
{
       public DBLinq(){}

        #region Propriedades
       public DateTime DataCriacaoValor
        {
              get
              {
                 return Entity.DataCriacao;
              }
        }

        public DateTime DataActualizacaoValor
        {
            get
            {
                return Entity.DataActualizacao;
            }
        }

        public Estado Estado
        {
            get
            {
                return new Estado(Entity.ChaveEstado);
            }
            set
            {
                Entity.ChaveEstado = (int)value.ChaveEstado;
            }
        }

        public Interface Interface
        {
            get
            {
                return new Interface(Entity.Interface);
            }
        }

        private Table<T> _table;
        public Table<T> Table
        {
            get
            {
                if (_table == null)
                    _table = DBConfig.DBInstance.GetTable<T>();
                return _table;
            }
        }
```

```
public long OperadorID
{
    get { return Entity.OperadorID; }
}

private T Entity
{
    get { return this as T; }
}

#endregion

#region Guardar
public ResultadoAccao OperacaoBD(TipoOperacaoBD operacao)
{
    switch (operacao)
    {
        case TipoOperacaoBD.NovoRegisto:
            return NovoRegisto();
        case TipoOperacaoBD.ActualizarRegisto:
            return ActualizarRegisto();
        case TipoOperacaoBD.BloquearRegisto:
            return BloquearRegisto();
        case TipoOperacaoBD.DesbloquearRegisto:
            return DesbloquearRegisto();
        case TipoOperacaoBD.ApagarRegisto:
            return ApagarRegisto();
        case TipoOperacaoBD.RemoverRegistoBD:
            return RemoverRegisto();
        default:
            break;
    }
    return new ResultadoAccao(false, "Erro desconhecido.");
}

private ResultadoAccao BloquearRegisto()
{
    try
    {
        if (DBConfig.Trans != null)
            Table.Context.Transaction = DBConfig.Trans;
        Entity.ChaveEstado = (int)Estado.Bloqueado.ChaveEstado;
        Entity.DataActualizacao = BDUtils.DataHoraServidor();
        if (DadosAplicacao.Operdador.ID == 0)
            Entity.OperadorID = -123;
        else
            Entity.OperadorID = DadosAplicacao.Operdador.ID;
        Table.Context.SubmitChanges();
        return ResultadoAccao.ResultadoBloquear(true);
    }
    catch
    {
         return new ResultadoAccao(false, "Erro a bloquear registo.");
    }
}
```

```
        private ResultadoAccao DesbloquearRegisto()
        {
            try
            {
                if (DBConfig.Trans != null)
                    Table.Context.Transaction = DBConfig.Trans;
                Entity.ChaveEstado = (int)Estado.Normal.ChaveEstado;
                Entity.DataActualizacao = BDUtils.DataHoraServidor();
                if (DadosAplicacao.Operdador.ID == 0)
                    Entity.OperadorID = -123;
                else
                    Entity.OperadorID = DadosAplicacao.Operdador.ID;
                Table.Context.SubmitChanges();
                return ResultadoAccao.ResultadoDesbloquear(true);
            }
            catch {
              return new ResultadoAccao(false, "Erro a desbloquear registo.");
            }

        }
        private ResultadoAccao ApagarRegisto()
        {
            try
            {
                if (DBConfig.Trans != null)
                    Table.Context.Transaction = DBConfig.Trans;
                Entity.ChaveEstado = (int)Estado.Apagado.ChaveEstado;
                Entity.DataActualizacao = BDUtils.DataHoraServidor();
                if (DadosAplicacao.Operdador.ID == 0)
                    Entity.OperadorID = -123;
                else
                    Entity.OperadorID = DadosAplicacao.Operdador.ID;
                Table.Context.SubmitChanges();
                return ResultadoAccao.ResultadoApagar(true);
            }
            catch
            {
                return new ResultadoAccao(false, "Erro a apagar registo.");
            }
         }

        private ResultadoAccao RemoverRegisto()
        {
            try
            {
                if (DBConfig.Trans != null)
                    Table.Context.Transaction = DBConfig.Trans;
                    Table.Context.ExecuteQuery<Object>("delete from " + typeof(T).Name +
                    "                           where id = " + Entity.ID);
                return ResultadoAccao.ResultadoApagar(true);
            }
            catch
            {
                return new ResultadoAccao(false, "Erro a remover registo.");
            }
        }
        #endregion

    }
```

Apêndice III – ResultadoAccao

```
public class ResultadoAccao
{
    public ResultadoAccao()
    {
        Sucesso = false;
        IDInserido = 0;
        Mensagem = "";
    }

    public ResultadoAccao(bool sucesso, string mensagem)
    {
        Sucesso = sucesso;
        IDInserido = 0;
        Mensagem = mensagem;
    }

    public ResultadoAccao(bool sucesso, string mensagem, long IDInserido)
    {
        Sucesso = sucesso;
        this.IDInserido = IDInserido;
        Mensagem = mensagem;
    }

    public bool Sucesso { get; set; }
    public long IDInserido { get; set; }
    public string Mensagem { get; set; }



    public static ResultadoAccao ResultadoGuardar(bool sucesso, long ID)
    {
        return new ResultadoAccao(sucesso, "Dados guardados com sucesso.", ID);
    }

    public static ResultadoAccao ResultadoActualizar(bool sucesso)
    {
        return new ResultadoAccao(sucesso, "Dados actualizados com sucesso.");
    }

    public static ResultadoAccao ResultadoApagar(bool sucesso)
    {
        return new ResultadoAccao(sucesso, "Dados apagados com sucesso.");
    }

    public static ResultadoAccao ResultadoBloquear(bool sucesso)
    {
        return new ResultadoAccao(sucesso, "Dados bloqueados com sucesso.");
    }

    public static ResultadoAccao ResultadoDesbloquear(bool sucesso)
    {
        return new ResultadoAccao(sucesso, "Dados desbloqueados com sucesso.");
    }
}
```

Apêndice IV – BusinessBase

```
public abstract class BusinessBase<T, K>: IBusinessBase where K : DBLinq<K>, IDBObject
{

        public BusinessBase()
        {
            _table = (K)Activator.CreateInstance(typeof(K), new object[] {});
            initValues();
        }

        public BusinessBase(long ID)
        {
            K item = (K)Activator.CreateInstance(typeof(K), new object[] { });
            _table = item.Table.FirstOrDefault(p => p.ID == ID && p.ChaveEstado
                      !=  (int)Estado.Apagado.ChaveEstado);
            initValues();
        }

        public BusinessBase(K Tabela)
        {
            _table = Tabela;
            initValues();
        }

       public virtual void initValues() {}

        private K _table;
        public T converte(K elemento)
        {
            if(elemento == null)
                return (T)Activator.CreateInstance(typeof(T));
            return (T)Activator.CreateInstance(typeof(T), new object[] { elemento });
        }

        /// <summary>
        /// Objecto que representa a tabela em BD
        /// </summary>
        public K Tabela
        {
            get
            {
                if(_table == null)
                    _table = (K)Activator.CreateInstance(typeof(K));
                return _table;
            }
            set
            {
                if (value == null) throw new Exception(
                    "Este parâmetro não pode ser null.");
                _table = value;
            }
         }
```

```
        /// <summary>
        /// Converte uma lista de elementos da data layer para os correspondentes da
BusinessLayer
        /// </summary>
        /// <param name="list"></param>
        /// <returns></returns>
        public List<T> converter(IEnumerable<K> list)
        {
            List<T> _rtn = new List<T>();
            try
            {
                foreach (K d in list)
                    _rtn.Add(converte(d));
            }
            catch
            { }
            return _rtn;
        }

        /// <summary>
        /// Lista todos os elementos à excepção dos marcados como apagados
        /// </summary>
        /// <returns></returns>
        public List<T> ListarTodos()
        {
            return converter(this.Tabela.Table.Where(p => p.ChaveEstado !=
                    (int)ChaveEstado.Apagado).OrderBy(p => p.ID).ToList());
        }
       /// <summary>
        /// Recebe como argumento uma expressão elementos da data layer e
        /// retorna uma lista de elementos da business layer
        /// </summary>
        /// <param name="expression"></param>
        /// <returns></returns>
        public List<T> Listar(Expression<Func<K,bool>> expression)
        {
            return converter(this.Tabela.Table.Where(expression));
        }

        /// <summary>
        /// Obtêm o primeiro elemento que cumpra a expressão
        /// </summary>
        /// <param name="expression"></param>
        /// <returns></returns>
        public T ObterPrimeiro(Expression<Func<K, bool>> expression)
        {
            return converte(this.Tabela.Table.FirstOrDefault(expression));
        }

        /// <summary>
        /// Recebe um comando SQL para listar a tabela correspondente ao elemento da
        /// business layer
        /// </summary>
        /// <param name="command"></param>
        /// <returns></returns>
        public List<T> ListarCustomQuery(String command)
        {
            return converter(this.Tabela.Table.Context.ExecuteQuery<K>(command));
        }
```

```
        /// <summary>
        /// Retorna o elemento com o ID pretendido
        /// </summary>
        /// <param name="ID"></param>
        /// <returns></returns>
        public T GetByID(long ID)
        {
            K item = (K)Activator.CreateInstance(typeof(K), new object[] {});
            item = item.Table.FirstOrDefault(p => p.ID == ID && p.ChaveEstado != 
(int)Estado.Apagado.ChaveEstado);
            return converte(item);
        }

        /// <summary>
        /// [get] Data em que o registo sofreu alteração
        /// </summary>
        public DateTime DataActualizacao
        {
            get { return _table.DataActualizacaoValor; }
        }

        /// <summary>
        /// [get] Data em que o registo foi criado em base de dados
        /// </summary>
        public DateTime DataCriacao
        {
            get { return _table.DataCriacaoValor; }
        }

        /// <summary>
        /// [get] ID do registo
        /// </summary>
        public long ID
        {
            get { return _table.ID; }
        }

        /// <summary>
        /// [get;set] Estado do registo
        /// </summary>
        public Estado Estado
        {
            get
            {
                return _table.Estado;
            }
            set
            {
                _table.Estado = value;
            }
        }

        /// <summary>
        /// [get] Imagem do estado
        /// </summary>
        public Bitmap EstadoImagem
        {
            get { return _table.Estado.Imagem; }
        }
```

```
        /// <summary>
        /// [get] ID do operador que por ultimo criou/alterou o registo
        /// </summary>
        public long OperadorID
        {
            get { return _table.OperadorID; }
        }
        /// <summary>
        /// [get] ID do posto onde por ultimo se criou/alterou o registo
        /// </summary>
        public long PostoID
        {
            get { return _table.PostoID; }
        }
        /// <summary>
        /// [get] Interface com a qual por ultimo se criou/alterou o registo
        /// </summary>
        public Interface Interface
        {
            get { return _table.Interface; }
        }
        /// <summary>
        /// Guarda os dados ou o resultado de determinada acção sobre o
        /// </summary>
        /// <param name="operacao"></param>
        /// <returns>ResultadoAccao</returns>
        public virtual ResultadoAccao OperacaoBD(TipoOperacaoBD operacao)
        {
            ResultadoAccao res = VerificacaoDados();
            if(!res.Sucesso) return res;
            return _table.OperacaoBD(operacao);
        }

        /// <summary>
        /// Faz a verificação pretendida dos valores a guardar
        /// </summary>
        /// <returns>ResultadoAccao</returns>
        public abstract ResultadoAccao VerificacaoDados();

        /// <summary>
        /// Apaga o registo pretendido da base de dados
        /// </summary>
        /// <returns></returns>
        public virtual ResultadoAccao RemoverRegistoBD()
        {
            return OperacaoBD(TipoOperacaoBD.RemoverRegistoBD);
        }

        public override bool Equals(object obj)
        {
            IBusinessBase item = null;
            try
            {
                item = (IBusinessBase)obj;
                return this.ID == item.ID;
            }
            catch { }
            return false;
        }
    }
}
```

Apêndice V – ProcessoCalculo

```
public class ProcessoCalculo
{
    public ProcessoCalculo(Processo processo, Quantidade QuantidadeAProduzir)
    {
        _processo = processo;
        _listaPais = new List<ProcessoCalculo>();
        _listaFilhos = new List<ProcessoCalculo>();
        _produto = null;
        _totalAProduzir = QuantidadeAProduzir;
        _custoTotal = new Dinheiro();
        _custoProdutos = new Dinheiro();
        _custoEquipamentos = new Dinheiro();
        _custoColaboradores = new Dinheiro();
        _listaNecessidades = null;

    }

    #region Propriedades
    private Processo _processo;
    public Processo Processo
    {
        get
        { return _processo;  }
        set
        {_processo = value; }
    }

    private Produto _produto;
    public Produto Produto
    {
        get
        {
            return _processo.Produto;
        }
        set
        {
            _processo.Produto = value;
        }
    }

   /// <summary>
   /// Lista de processos que este processo vai fornecer
   /// </summary>
   private List<ProcessoCalculo> _listaPais;
   public List<ProcessoCalculo> Pais
   {
      get
      {
          return _listaPais;
      }
   }
```

```
/// <summary>
/// Lista de processos que fornecem este processo
/// </summary>
private List<ProcessoCalculo> _listaFilhos;
public List<ProcessoCalculo> Filhos
{
    get
    {
        return _listaFilhos;
    }
}

private Dinheiro _custoTotal;
public Dinheiro CustoTotal
{
    get
    {
        return _custoTotal;
    }
}

private Dinheiro _custoProdutos;
public Dinheiro CustoProdutos
{
    get
    {
        return _custoProdutos;
    }
}

private Dinheiro _custoEquipamentos;
public Dinheiro CustoEquipamentos
{
    get
    {
        return _custoEquipamentos;
    }
}

private Dinheiro _custoColaboradores;
public Dinheiro CustoColaboradores
{
    get
    {
        return _custoColaboradores;
    }
}
```

```
 #region Dados MRP
 private List<OrdemTrabalho> _agendamentos;
 public List<OrdemTrabalho> Agendamentos
 {
    get
    {
       if (_agendamentos == null)
              _agendamentos = new List<OrdemTrabalho>();
       return _agendamentos;
    }
    set
    {
       _agendamentos = value;
    }
 }

 public TimeSpan TempoTotal
 {
       get
       {
            return new TimeSpan(this.Processo.TempoCicloReal.Ticks *
              (decimal.ToInt64(this.TotalAProduzir.Valor)));
       }
}


 private Quantidade _totalAProduzir;
 public Quantidade TotalAProduzir
 {
       get
       {
              return _totalAProduzir;
       }
        }
 #endregion
 #endregion


 /// <summary>
 /// Adiciona os processos antecessores e dependentes
 /// </summary>
 /// <param name="processo"></param>
 public void AdicionaPais(ProcessoCalculo processo)
 {
       if (!_listaPais.Contains(processo))
       {
              _listaPais.Add(processo);
       }
 }
```

```
/// <summary>
/// Adiciona os processos sucessores e que esta dependente
/// </summary>
/// <param name="processo"></param>
public void AdicionaFilhos(ProcessoCalculo processo)
{
      if (!_listaFilhos.Contains(processo))
                _listaFilhos.Add(processo);
}


/// <summary>
/// Incrementa a quantidade a produzir
/// </summary>
/// <param name="produto"></param>
/// <returns></returns>
public void adicionaProducao(Quantidade quantidade)
{
      _totalAProduzir += quantidade;
}


private List<ProdutoNecessidade> _listaNecessidades;
public Quantidade NecessidadeTotalPorProduto(Produto produto)
{
      if (_listaNecessidades == null)
                _listaNecessidades = this.Processo.NecessidadesProdutos(TotalAProduzir);

      foreach (ProdutoNecessidade pn in _listaNecessidades)
      {
            if (pn.Produto != null && produto != null && pn.Produto.ID == produto.ID)
                  return pn.Quantidade;
       }
       return new Quantidade();
}


public List<CustosSimulacao> ListaCustos()
{
      List<CustosSimulacao> _list = new List<CustosSimulacao>();
      _list.Add(new CustosSimulacao(
                  "Colaboradores", this.CustoColaboradores.ToString()));
      _list.Add(new CustosSimulacao(
                  "Equipamentos", this.CustoEquipamentos.ToString()));
      _list.Add(new CustosSimulacao(
                  "Produtos", this.CustoProdutos.ToString()));
      _list.Add(new CustosSimulacao(
                  "Total", this.CustoTotal.ToString()));
      return _list;
}
```

```
public void setCustos()
{
      _custoProdutos = setCustosProdutos();
      _custoEquipamentos = this.Processo.CalculoCustosEquipamentos(TotalAProduzir);
      _custoColaboradores = this.Processo.CalculoCustosColaboradores(TotalAProduzir);
      _custoTotal = _custoProdutos + _custoEquipamentos + _custoColaboradores;
}

private Dinheiro setCustosProdutos()
{
      Dinheiro d = new Dinheiro();
      foreach (ProcessoCalculo ps in _listaFilhos)
      {
            Dinheiro d_ps = ps.CustoTotal;
            Quantidade q1 = ps.TotalAProduzir;
            Quantidade q2 = q1 / this.NecessidadeTotalPorProduto(ps.Produto);
            d += new Dinheiro(decimal.Divide(d_ps.Valor, q2.Valor));
      }

      return this.Processo.GetCustosProdutos(TotalAProduzir) + d;
}

public void setAgendamentos()
{
      Quantidade _stockInicial = new Quantidade();
      if (_listaPais.Count < 1 && this.Agendamentos.Count < 1)
      {
            OrdemTrabalho oTrabalho = new OrdemTrabalho();
            oTrabalho.Processo = this.Processo;
            oTrabalho.Fim = new DateTime(2100, 12, 31, 0, 0, 0);
            oTrabalho.Inicio = oTrabalho.Fim.Add(-
                  this.Processo.TempoParaProduzir(this.TotalAProduzir));
            oTrabalho.Tarefa = Tarefa.Laborar;
            oTrabalho.Quantidade = this.TotalAProduzir;
            this.Agendamentos.Add(oTrabalho);
            if (this.Processo.TempoCicloReal > new TimeSpan(0))
            {
                  OrdemTrabalho setup = new OrdemTrabalho();
                  setup.Fim = oTrabalho.Inicio;
                  setup.Inicio = setup.Fim.Add(-this.Processo.TempoCicloReal);
                  setup.Tarefa = Tarefa.Setup;
                  this.Agendamentos.Add(setup);
            }
      }

      foreach (ProcessoCalculo ps in _listaPais)
      {
            if (ps.Agendamentos.Count < 1)
                  ps.setAgendamentos();
            foreach (OrdemTrabalho pa in ps.Agendamentos)
            {
                  OrdemTrabalho oTrabalho = new OrdemTrabalho();
                  oTrabalho.Processo = this.Processo;
```

```
                    if (pa.Tarefa == Tarefa.Laborar)
                    {
                        oTrabalho.Fim = pa.Fim.Add(-ps.Processo.TempoCicloReal);
                        oTrabalho.Quantidade = pa.Quantidade;
                        oTrabalho.Inicio = oTrabalho.Fim.Add(-
                           this.Processo.TempoParaProduzir(oTrabalho.Quantidade));
                        oTrabalho.Tarefa = Tarefa.Laborar;
                        if (this.Agendamentos.Count > 0)
                        {
                            for (int i = 0; i < this.Agendamentos.Count; i++)
                            {
                                if ((oTrabalho.Inicio <= this.Agendamentos[i].Inicio &&
                                     oTrabalho.Fim <= this.Agendamentos[i].Fim) ||
                                    (oTrabalho.Inicio <= this.Agendamentos[i].Inicio &&
                                     oTrabalho.Fim >= this.Agendamentos[i].Fim) ||
                                    (oTrabalho.Inicio >= this.Agendamentos[i].Inicio &&
                                     oTrabalho.Fim >= this.Agendamentos[i].Fim))
                                {

                                    if (oTrabalho.Fim < this.Agendamentos[i].Fim)
                                        oTrabalho.Fim = pa.Fim;
                                    oTrabalho.Quantidade +=
                                         this.Agendamentos[i].Quantidade;
                                    oTrabalho.Inicio = oTrabalho.Fim.Add(-
                                  this.Processo.TempoParaProduzir(oTrabalho.Quantidade));
                                    this.Agendamentos.RemoveAt(i);
                                    i = -1;
                                }
                            }
                        }
                        this.Agendamentos.Add(oTrabalho);
                    }
                }

                if (this.Processo.TempoCicloReal > new TimeSpan(0))
                {
                        DateTime iniciotemp = new DateTime(2100, 12, 31);
                        foreach (OrdemTrabalho ppa in this.Agendamentos)
                            if (iniciotemp > ppa.Inicio) iniciotemp = ppa.Inicio;
                        OrdemTrabalho setup = new OrdemTrabalho();
                        setup.Fim = iniciotemp;
                        setup.Inicio = setup.Fim.Add(-this.Processo.TempoCicloReal);
                        setup.Tarefa = Tarefa.Setup;
                        this.Agendamentos.Add(setup);
                }
            }
        }
        private void adicionarAgenda(OrdemTrabalho agenda)
        {
            if (this.Agendamentos.Count < 1)
            {
                this.Agendamentos.Add(agenda);
                return;
            }
            bool mudou = true;
            while (mudou)
            {
                foreach (OrdemTrabalho pa in this.Agendamentos)
                {
```

```
private void adicionarAgenda(OrdemTrabalho agenda)
{
      if (this.Agendamentos.Count < 1)
      {
            this.Agendamentos.Add(agenda);
                  return;
      }
      bool mudou = true;
      while (mudou)
      {
            foreach (OrdemTrabalho pa in this.Agendamentos)
            {
                  if (agenda.Fim > pa.Inicio)
                  {
                        TimeSpan periodo1 = agenda.Fim - agenda.Inicio;
                        agenda.Fim = pa.Inicio;
                        agenda.Inicio = agenda.Inicio.Add(-periodo1);
                  }
            }
       }

}


public override string ToString()
{
      if (_processo == null) return "";
            return _processo.ToString();
}
```

Apêndice VI – KanbanCalculoQte

```
public class KanbanCalculoQte : BusinessBase<KanbanCalculoQte,
        dbKanbanCalculoQte>, IKanbanCalculoQte
    {
        public KanbanCalculoQte() :base() {}
        public KanbanCalculoQte(long ID) : base(ID) {}
        public KanbanCalculoQte(dbKanbanCalculoQte tabela) : base(tabela) { }

        public override void initValues()
        {
            base.initValues();
            _processo = null;
            _supermercado = new ParametroGenerico<KanbanSupermercado>();
            _estatisticas = null;
        }

        #region Propriedades
        private EstatisticaDadosProducao _estatisticas;
        private EstatisticaDadosProducao Estatisticas
        {
            get
            {
                if (_estatisticas == null)
                {
                    _estatisticas = new EstatisticaDadosProducao();
                    _estatisticas =  _estatisticas.ObterPrimeiro(P => P.ChaveEstado ==
(int)ChaveEstado.Normal &&
                    P.refProduto == this.Processo.Produto.ID &&
                    P.refProcesso == this.Processo.ID);
                }
                return _estatisticas;

            }
        }



        private Processo _processo;
        public Processo Processo
        {
            get
            {
                if (_processo == null)
                    _processo = new Processo().ObterPrimeiro(P => P.ChaveEstado == 0 &&
                        P.refKanbanCalculoQte == this.ID);
                return _processo;
            }
            set
            {
                _processo = value;
            }
        }
```

```
private ParametroGenerico<KanbanSupermercado> _supermercado;
        public KanbanSupermercado Supermercado
        {
            get
            {
                return _supermercado.getItem(Tabela.refSupermercado);
            }
            set
            {
                Tabela.refSupermercado = _supermercado.setItem(value);
            }
        }


        /// <summary>
        /// Referência Produto
        /// </summary>
        public string Referencia
        {
            get
            {
                if (this.Processo != null && this.Processo.Produto != null)
                    return this.Processo.Produto.Referencia;
                return "";
            }
        }

        /// <summary>
        /// Descrição do Produto
        /// </summary>
        public string Descricao
        {
            get
            {
                if (this.Processo.Produto != null)
                    return this.Processo.Produto.Descricao;
                return "";
            }
        }

        /// <summary>
        /// Máquina a Utilizar no fabrico desta Referência
        /// </summary>
        public string NomeMaquinaAT
        {
            get
            {
                if (this.Processo != null && this.Processo.Produto != null)
                    return this.Processo.Produto.Referencia;
                return "";
            }
        }
```

```
        /// <summary>
        /// % da Produção total que é feita nesta Máquina
        /// </summary>
        public Percentagem PerProdTotal
        {
            get
            {
                return new Percentagem(Tabela.PerProdTotal);
            }
            set
            {
                Tabela.PerProdTotal = value.Valor;
            }
        }

        /// <summary>
        /// % da Produção que Falta Alocar
        /// </summary>
        public Percentagem PerProdPorAlocar
        {
            get
            {
                return new Percentagem(Tabela.PerProdPorAlocar);
            }
            set
            {
                Tabela.PerProdPorAlocar = value.Valor;
            }
        }

        /// <summary>
        /// Procura Média Semanal
        /// </summary>
        public int ProcuraMediaSemanal
        {
            get
            {
                return Tabela.ProcuraMediaSemanal;
            }
            set
            {
                Tabela.ProcuraMediaSemanal = value;
            }
        }

        /// <summary>
        /// Desvio padrão/média
        /// </summary>
        public Percentagem DesvioPadraoMedia
        {
            get
            {
                return new Percentagem(Tabela.DesvioPadraoMedia);
            }
            set
            {
                Tabela.DesvioPadraoMedia = value.Valor;
            }
        }
```

```
        /// <summary>
        /// Desvio padrão [Semana]
        /// </summary>
        public int DesvioPadraoSemana
        {
            get
            {
                try
                {
                    return decimal.ToInt32(this.DesvioPadraoMedia.Valor *
ProcuraMediaSemanal);
                }
                catch
                {
                    return 0;
                }
            }
        }


        /// <summary>
        /// Máximo semanal
        /// </summary>
        public int MaximoSemanal
        {
            get
            {
                return this.ProcuraMediaSemanal + this.DesvioPadraoSemana;
            }

        }

        /// <summary>
        /// Procura média diária com % de sucata e rework
        /// </summary>
        public int ProcMediaDiariaComRejeitado
        {
            get
            {
                if(this.Processo == null || this.Processo.Produto == null) return 0;
                try
                {
                    return decimal.ToInt32(decimal.Ceiling(
                        (decimal.Divide(decimal.Divide((decimal)this.ProcuraMediaSemanal,
(decimal)Turno.DiasTrabalhoSemana) *
                         (100 + this.Estatisticas.PercentagemSucata +
this.Estatisticas.PercentagemRework), 100))));

                }
                catch
                {
                    return 0;
                }
            }
        }
```

```
        /// <summary>
        /// Desvio padrão
        /// </summary>
        public int DesvioPadrao
        {
            get
            {
                return decimal.ToInt32(decimal.Multiply
                    (this.DesvioPadraoMedia.Valor,(decimal)ProcMediaDiariaComRejeitado));
            }
        }
        /// <summary>
        /// Número de referências fabricadas na máquina/Bancada
        /// </summary>
        public int NumReferenciasMaquina
        {
            get
            {
                int i = 0;
                foreach (KanbanCalculoQte kcq in
this.ListaPorEquipamentosEAreasTrabalho())
                {
                    if (this.ID != kcq.ID && this.Processo.Maquina ==
kcq.Processo.Maquina &&
                        this.Processo.Bancada == kcq.Processo.Bancada)
                        i++;
                }
                return i;
            }
        }
        /// <summary>
        /// Tempo ciclo (minutos) - SAP/Medido
        /// </summary>
        public decimal TempoCicloMinutosSap
        {
            get
            {
                return (decimal)this.Processo.TempoCicloSAP.TotalMinutes;
            }
        }
        /// <summary>
        /// Tempo de Ciclo Real [afectado do OEE]
        /// </summary>
        public decimal TempoCicloRealMinutos
        {
            get
            {
                return (decimal)this.Processo.TempoCicloReal.TotalMinutes;
            }
        }

        public TimeSpan TempoCicloReal
        {
            get
            {
                long ticks = (long)this.TempoCicloRealMinutos * 60000;
                return new TimeSpan(ticks);
            }
        }
```

```
        /// <summary>
        /// Nº Médio de Peças por turno
        /// </summary>
        public int NumMedioPecasTurno
        {
            get
            {
                if (this.TempoCicloRealMinutos == 0)
                    return 0;
                return decimal.ToInt32(
                    decimal.Ceiling(
                        decimal.Divide(this.NumHorasTurno * 60,
this.TempoCicloRealMinutos)));
            }
        }

        /// <summary>
        /// OEE
        /// </summary>
        public Percentagem OEE
        {
            get
            {
                return this.Processo.OEE;
            }
        }

        /// <summary>
        /// % Sucata
        /// </summary>
        public Percentagem Sucata
        {
            get
            {
                return new Percentagem(this.Estatisticas.PercentagemSucata);
            }
        }

        /// <summary>
        /// % Rework Constante
        /// </summary>
        public Percentagem ReworkConstante
        {
            get
            {
                return new Percentagem(this.Estatisticas.PercentagemRework);
            }
        }

        /// <summary>
        /// Carga diária (h)
        /// </summary>
        public decimal CargaDiariaHoras
        {
            get
            {
                return decimal.Multiply((decimal)this.ProcMediaDiariaComRejeitado,
                    this.TempoCicloRealMinutos);
            }
        }
```

```
/// <summary>
        /// Carga diária acumul. (h)
        /// </summary>
        public decimal CargaDiariaAcumuladaHoras
        {
            get
            {
                List<KanbanCalculoQte> _list = ListaPorEquipamentosEAreasTrabalho();
                decimal soma = 0;
                foreach (KanbanCalculoQte k in _list)
                {
                    if(k.ID != this.ID)
                    {
                        soma += k.CargaDiariaHoras;
                    }
                }
                return soma;
            }
        }

        /// <summary>
        /// Número de dias Trabalhados numa semana
        /// </summary>
        public int NumDiasTrabSemana
        {
            get
            {
                return Turno.DiasTrabalhoSemana;
            }
        }

        /// <summary>
        /// Número de Turnos Disponiveis
        /// </summary>
        public int NumTurnosDisponiveis
        {
            get
            {
                return Turno.NumTurnosDisponiveis;
            }
        }

        /// <summary>
        /// Número de Horas por Turno
        /// </summary>
        public decimal NumHorasTurno
        {
            get
            {
                return Turno.NumHorasTurnoDisponiveis;
            }
        }
```

```
        /// <summary>
        /// Carga diária acumul. (máq.) - %
        /// </summary>
        public Percentagem CargaDiariaAcumuladaMaq
        {
            get
            {
                if (decimal.Multiply((decimal)this.NumTurnosDisponiveis,
                        (decimal)this.NumHorasTurno) == 0)
                    return new Percentagem(0);
                decimal d = decimal.Divide(
                    this.CargaDiariaAcumuladaHoras,
                        decimal.Multiply((decimal)this.NumTurnosDisponiveis,
                        (decimal)this.NumHorasTurno)) * 100;
                return new Percentagem(d);
            }
        }


        /// <summary>
        /// Numero máximo de suportes
        /// </summary>
        public int NumMaxSuportes
        {
            get
            {
                return Tabela.NSuportesTotal;
            }
            set
            {
                Tabela.NSuportesTotal = value;
            }
        }

        /// <summary>
        /// Numero máximo de suportes
        /// </summary>
        public int NumSuportesUsar
        {
            get
            {
                return Tabela.NSuportesUsar;
            }
            set
            {
                Tabela.NSuportesUsar = value;
            }
        }

        /// <summary>
        /// Processo
        /// </summary>
        string IKanbanCalculoQte.ProcessoNome
        {
            get
            {
                return this.Processo.Nome;
            }
        }
```

```
        /// <summary>
        /// % Ratio Peça/Máquina
        /// </summary>
        public Percentagem RatioPecaMaquina
        {
            get
            {
                if (this.CargaDiariaAcumuladaHoras == 0)
                    return new Percentagem(0);
                return new Percentagem(
                    decimal.Divide(this.CargaDiariaHoras, this.CargaDiariaAcumuladaHoras)
* 100);
            }
        }

        /// <summary>
        /// Horas Possiveis ou Trabalhadas
        /// </summary>
        public decimal HorasPossiveisTrabalhadas
        {
            get
            {
                if (this.CargaDiariaAcumuladaHoras >
                    decimal.Multiply(this.NumTurnosDisponiveis, this.NumHorasTurno))
                    return this.RatioPecaMaquina.Valor * this.NumTurnosDisponiveis *
this.NumHorasTurno / 100;

                return this.CargaDiariaHoras;
            }
        }

        /// <summary>
        /// Peças Produzidas
        /// </summary>
        public int PecasProduzidas
        {
            get
            {
                if (this.TempoCicloRealMinutos == 0)
                    return 0;
                decimal d =
                    decimal.Ceiling(
                        decimal.Divide(this.HorasPossiveisTrabalhadas * 60,
this.TempoCicloRealMinutos));
                return decimal.ToInt32(d);
            }
        }

        /// <summary>
        /// Perda de produção
        /// </summary>
        public int PecasNecessarias
        {
            get
            {
                return this.ProcMediaDiariaComRejeitado;
            }
        }
```

```
        /// <summary>
        /// Potencial de produção
        /// </summary>
        public int PerdaProducao
        {
            get
            {
                if (this.PecasNecessarias > this.PecasProduzidas)
                    return this.PecasNecessarias - this.PecasProduzidas;
                return 0;
            }
        }

        /// <summary>
        /// Best product
        /// </summary>
        public int PotencialProducao
        {
            get
            {
                if(this.TempoCicloRealMinutos == 0)
                    return 0;
                decimal d1 = 0;
                if (this.CargaDiariaAcumuladaHoras <= this.NumTurnosDisponiveis *
this.NumHorasTurno)
                    d1 = (this.RatioPecaMaquina.Valor / 100) *
                        ((this.NumTurnosDisponiveis * this.NumHorasTurno) -
this.CargaDiariaAcumuladaHoras);
                return decimal.ToInt32(
                    decimal.Ceiling((d1 * 60) / this.TempoCicloRealMinutos));

            }
        }

        /// <summary>
        /// Best product
        /// </summary>
        public int BestProduct
        {
            get
            {
                if (this.TempoCicloRealMinutos == 0)
                    return 0;
                decimal d1 = 0;
                if (this.CargaDiariaAcumuladaHoras <= this.NumTurnosDisponiveis *
this.NumHorasTurno)
                    d1 = (this.NumTurnosDisponiveis * this.NumHorasTurno) -
this.CargaDiariaAcumuladaHoras;
                return decimal.ToInt32(
                    decimal.Ceiling((d1 * 60) / this.TempoCicloRealMinutos));
            }
        }
```

```
/// <summary>
        /// Número de dias de procura produzidos [1 = produção diária] sempre que entra
em produção
        /// </summary>
        public decimal NumDiasProcuraProduzidos
        {
            get
            {
                return Tabela.NDiasProcProd;
            }
            set
            {
                Tabela.NDiasProcProd = value;
            }
        }

        /// <summary>
        /// Quantidade por  Embalagem
        /// </summary>
        public int QuantidadeEmbalagem
        {
            get
            {
                if (this.Supermercado != null &&
                        this.Supermercado.EmbalagemCapacidade != null)
                    return decimal.ToInt32(
                        this.Supermercado.EmbalagemCapacidade.Capacidade.Valor);
                return 0;
            }
        }

        /// <summary>
        /// Unidade de agrupamento
        /// </summary>
        public int UnidadeAgrupamento
        {
            get
            {
                if (this.Supermercado != null)
                    return this.Supermercado.UnidadesAgrupamento;
                return 0;
            }
        }

        /// <summary>
        /// Quantidade por  Embalagem [com unidade de agrupamento]
        /// </summary>
        public int QuantidadeEmbalagemAgrupamento
        {
            get
            {
                return this.QuantidadeEmbalagem * this.UnidadeAgrupamento;
            }
        }
```

```
        /// <summary>
        /// Tempo de setup (h)
        /// </summary>
        public decimal TempoSetupHoras
        {
            get
            {
                if (this.Processo != null)
                    return (decimal)this.Processo.TempoSetup.TotalHours;
                return 0;
            }

        }

        /// <summary>
        /// Lote de produção (peças)
        /// </summary>
        public int LoteProducao
        {
            get
            {
                return decimal.ToInt32(

decimal.Ceiling(decimal.Multiply((decimal)this.ProcMediaDiariaComRejeitado,
                    this.NumDiasProcuraProduzidos)));
            }

        }

        /// <summary>
        /// Lote de produção (h)
        /// </summary>
        public decimal LoteProducaoHoras
        {
            get
            {
                return decimal.Divide(
                    decimal.Multiply((decimal)this.LoteProducao,
this.TempoCicloRealMinutos), 60);
            }

        }

        /// <summary>
        /// Lote de produção calculado na unidade de Embalagem
        /// </summary>
        public int LoteProducaoCalcUnidadeEmbalagem
        {
            get
            {
                if ((decimal)this.QuantidadeEmbalagemAgrupamento == 0)
                    return 0;
                return decimal.ToInt32(
                    decimal.Ceiling(
                        decimal.Divide((decimal)this.LoteProducao,
                            (decimal)this.QuantidadeEmbalagemAgrupamento)));
            }

        }
```

```
        /// <summary>
        /// Peças produzidas a mais
        /// </summary>
        public int PecasProduzidasMais
        {
            get
            {
                return this.LoteProducaoCalcUnidadeEmbalagem * this.QuantidadeEmbalagem -
                    this.LoteProducao;
            }
        }

        /// <summary>
        /// Tempo chegada 1ª caixa (h)
        /// </summary>
        public TimeSpan TempoChegada1Caixa
        {
            get
            {
                return new TimeSpan(Tabela.TChegadaPrimeiraCaixa);
            }
            set
            {
                Tabela.TChegadaPrimeiraCaixa = value.Ticks;
            }
        }



        /// <summary>
        /// Tempo de reposição (h)
        /// </summary>
        public decimal TempoReposicaoHoras
        {
            get
            {
                return this.LoteProducaoHoras + this.TempoSetupHoras +
(decimal)this.TempoChegada1Caixa.TotalHours;
            }
        }

        /// <summary>
        /// Tempo de reposição máximo exceptuando a refª em análise, para esta máquina
        /// </summary>
        public decimal TempoReposicaoMaxExcepRefHoras
        {
            get
            {
                decimal rtn = 0;
                foreach (KanbanCalculoQte kct in
this.ListaPorEquipamentosEAreasTrabalho())
                {
                    if (kct.ID != this.ID && kct.TempoReposicaoHoras > rtn)
                        rtn = kct.TempoReposicaoHoras;
                }
                return rtn;
            }
        }
```

```
/// <summary>
        /// Tempo takt (minutos)
        /// </summary>
        public decimal TempoTaktMinutos
        {
            get
            {
                if (this.ProcMediaDiariaComRejeitado == 0)
                    return 0;
                return
decimal.ToInt32(decimal.Divide(decimal.Multiply((decimal)this.NumTurnosDisponiveis,
                    this.NumHorasTurno * 60), this.ProcMediaDiariaComRejeitado));
            }
        }

        /// <summary>
        /// Ponto de lançamento (peças)
        /// </summary>
        public int PontoLancamento
        {
            get
            {
                if (this.TempoTaktMinutos == 0)
                    return 0;
                decimal d =(this.TempoReposicaoMaxExcepRefHoras * 60) /
this.TempoTaktMinutos;
                return decimal.ToInt32(decimal.Ceiling(d));
            }
        }

        /// <summary>
        /// Ponto de lançamento só para a reposição, em quantidade por embalagem
        /// </summary>
        public int PontoLançamentoQuantEmbalagem
        {
            get
            {
                if (this.QuantidadeEmbalagemAgrupamento == 0)
                    return 0;
                return decimal.ToInt32(
                        decimal.Ceiling(((decimal)this.PontoLancamento) /
                            ((decimal)this.QuantidadeEmbalagemAgrupamento)));
            }
        }

        /// <summary>
        /// Stock (pto lançto + lote)
        /// </summary>
        public int StockPtoLancLote
        {
            get
            {
                return this.PontoLançamentoQuantEmbalagem +
this.LoteProducaoCalcUnidadeEmbalagem;
            }
        }
```

```
        /// <summary>
        /// Stock 2S
        /// </summary>
        public decimal Stock2S
        {
            get
            {
                if (this.QuantidadeEmbalagem == 0)
                    return 0;
               return (2*(decimal)this.DesvioPadrao *
(decimal)this.NumDiasProcuraProduzidos) /
                    (decimal)this.QuantidadeEmbalagem;
            }
        }

        /// <summary>
        /// Stock Segurança
        /// </summary>
        public decimal StockSeguranca
        {
            get
            {
                if (this.QuantidadeEmbalagem == 0)
                    return 0;
               return ((this.ProcMediaDiariaComRejeitado + 2 * this.DesvioPadrao) *
                    this.NumDiasProcuraProduzidos * (this.Sucata.Valor/100)) /
                        ((decimal)this.QuantidadeEmbalagem);

            }
        }

        /// <summary>
        /// (Ponto de lançamento para a reposição) + DP + Stock segurança
        /// </summary>
        public decimal PontoLançamentoReposicaoDPStockSeg
        {
            get
            {
                return decimal.ToInt32(decimal.Ceiling(this.Stock2S + this.StockSeguranca
+
                    (decimal)this.PontoLançamentoQuantEmbalagem));
            }
        }

        /// <summary>
        /// Lote de Produção + Quantidade de Reposição + DP + Stock segurança
        /// </summary>
        public decimal LoteProduçãoQuantReposDPStockSeg
        {
            get
            {
                return (decimal)this.LoteProducaoCalcUnidadeEmbalagem +
                    this.PontoLançamentoReposicaoDPStockSeg;
            }
        }
```

```
        /// <summary>
        /// Stock médio
        /// </summary>
        public decimal StockMedio
        {
            get
            {
                return
decimal.Divide((decimal)(this.LoteProducao+2*this.DesvioPadrao+this.PontoLancamento)*
                    (1+(this.Sucata.Valor/100)),2);
            }
        }

        /// <summary>
        /// Stock médio [com unidade de agrupamento]
        /// </summary>
        public decimal StockMedioComUniAgrupamento
        {
            get
            {
                if (this.QuantidadeEmbalagemAgrupamento == 0)
                    return 0;
                return this.StockMedio / (decimal)this.QuantidadeEmbalagemAgrupamento;
            }
        }

        /// <summary>
        /// Quantidade por palete no supermercado/próximo processo
        /// </summary>
        public int QuantPaleteSupermercadoProxProcesso
        {
            get
            {
                return Tabela.QuantPaleteProxProc;
            }
            set
            {
                Tabela.QuantPaleteProxProc = value;
            }
        }

        /// <summary>
        /// Quantidade máxima expressa em paletes
        /// </summary>
        public int QuantidadeMaximaExpressaPaletes
        {
            get
            {
                decimal d = (this.LoteProduçãoQuantReposDPStockSeg * this.LoteProducao) /
                                (decimal)this.QuantPaleteSupermercadoProxProcesso;
                return decimal.ToInt32(decimal.Ceiling(d));
            }
        }
```

```
        /// <summary>
        /// Dias de procura cobertos pela quantidade em armazém
        /// </summary>
        public decimal DiasProcuraCobertosQuantidadeArmazem
        {
            get
            {
                if (this.ProcMediaDiariaComRejeitado == 0)
                    return 0;
                return
                    (decimal)(this.QuantidadeMaximaExpressaPaletes *
this.QuantPaleteSupermercadoProxProcesso) /
                        (decimal)this.ProcMediaDiariaComRejeitado;
            }
        }

        /// <summary>
        /// Zona Verde Calculada
        /// </summary>
        public int ZonaVerdeCalculada
        {
            get
            {
                return this.LoteProducaoCalcUnidadeEmbalagem;
            }
        }

        /// <summary>
        /// Zona Amarela Calculada
        /// </summary>
        public int ZonaAmarelaCalculada
        {
            get
            {
                int i1 =
decimal.ToInt32(decimal.Ceiling(this.LoteProduçãoQuantReposDPStockSeg));
                return i1 - ZonaVerdeCalculada - ZonaVermelhaCalculada;
            }
        }

        /// <summary>
        /// Zona Vermelha Calculada
        /// </summary>
        public int ZonaVermelhaCalculada
        {
            get
            {
                return decimal.ToInt32(decimal.Ceiling(
                    this.Stock2S + this.StockSeguranca));
            }
        }
```

```
/// <summary>
        /// Zona Verde Corrigida
        /// </summary>
        public int ZonaVerdeCorrigida
        {
            get
            {
                return Tabela.ZVerdeCorrigida;
            }
            set
            {
                Tabela.ZVerdeCorrigida = value;
            }
        }

        /// <summary>
        /// Zona Amarela Corrigida
        /// </summary>
        public int ZonaAmarelaCorrigida
        {
            get
            {
                return Tabela.ZAmarelaCorrigida;
            }
            set
            {
                Tabela.ZAmarelaCorrigida = value;
            }
        }

        /// <summary>
        /// Zona Vermelha Corrigida
        /// </summary>
        public int ZonaVermelhaCorrigida
        {
            get
            {
                return Tabela.ZVermelhaCorrigida;
            }
            set
            {
                Tabela.ZVermelhaCorrigida = value;
            }
        }

        public bool EmSistemaKanban
        {
            get
            {
                return Tabela.EmSistemaKanban;
            }
            set
            {
                Tabela.EmSistemaKanban = value;
            }
        }
```

```
public bool Simulacao
        {
            get
            {
                return Tabela.Simulacao;
            }
            set
            {
                Tabela.Simulacao = value;
            }
        }

        /// <summary>
        /// Total de Kanbans
        /// </summary>
        public int TotalKanbans
        {
            get
            {
                return this.ZonaVerdeCorrigida + this.ZonaAmarelaCorrigida +
this.ZonaVermelhaCorrigida;
            }
        }

        /// <summary>
        /// Kanbans por cavidade
        /// </summary>
        public int KanbansCavidade
        {
            get
            {
                if (this.TotalKanbans > 0)
                    return this.QuantPaleteSupermercadoProxProcesso /
this.QuantidadeEmbalagemAgrupamento;

                return 0;
            }
         }

        /// <summary>
        /// Número Total de Cavidades
        /// </summary>
        public int NumTotalCavidades
        {
            get
            {
                if (this.TotalKanbans > 0)
                    return decimal.ToInt32(
                        decimal.Ceiling(
                            (decimal)this.TotalKanbans / (decimal)this.KanbansCavidade));
                return 0;
            }
        }
```

```
        public Quantidade ProcuraMinimaPecas
        {
            get
            {
               return new Quantidade(decimal.Ceiling((decimal)this.PecasProduzidas *
                    this.NumDiasProcuraProduzidos),Unidades.Nenhuma);
            }
        }
        #endregion


        private List<KanbanCalculoQte> ListaPorEquipamentosEAreasTrabalho()
        {
            BDKanbansDataContext context = new BDKanbansDataContext();
            return
this.converter(context.dbKanbanCalculoQte_ListarPorEquipamentosEAreasTrabalho());
        }

        public override ResultadoAccao VerificacaoDados()
        {
            return new ResultadoAccao(true, "");
        }

        public override ResultadoAccao OperacaoBD(TipoOperacaoBD operacao)
        {

            ResultadoAccao res = base.OperacaoBD(operacao);
            if (res.Sucesso)
            {
                _processo.KanbanCalculo = this;
                res = _processo.OperacaoBD(TipoOperacaoBD.ActualizarRegisto);
            }
            return res;
        }
    }
```

Apêndice VII – Desenvolvimento Lean

1. Evolução do Desenvolvimento de Software

O desenvolvimento, tal como o conhecemos nos dias de hoje, é muito diferente do que se assistia até há bem pouco tempo. Este facto deve-se essencialmente a dois factores, que "forçaram" a busca de melhores metodologias, para dar resposta a um mercado cada vez maior:

- ➔ A taxa de sucesso é muito baixa, existindo muitos desperdícios e insatisfação por parte do consumidor.
- ➔ O aumento exponencial e a diversificação de produtos baseados em software.

➢ CHAOS Report

Num estudo efectuado pela primeira vez em 1994, feito pelo Standish Group, foram examinados mais de 8000 projectos de desenvolvimentos de software, chegando-se a resultados assustadores.

Foram incluídas, no estudo, empresas de pequena, média e grande dimensão de diversos segmentos de mercado (banca, segurança, saúde, etc.)

Os projectos eram classificados em três categorias:

Sucedido - O projecto foi completado dentro do prazo previsto e do orçamentado, fornecendo todas as ferramentas inicialmente especificadas.

Desafiado – O Projecto foi completado, mas passou o tempo estimado e o orçamento, ou não implementou o que se pretendia e estava especificado.

Falhado – O projecto acabou cancelado durante o seu desenvolvimento.

Ao longo dos anos o estudo tem sido actualizado, mostrando uma melhoria significativa.

	1994	1996	1998	2000	2002	2004
Sucedido	16%	27%	26%	28%	34%	29%
Desafiado	53%	33%	46%	49%	51%	53%
Falhado	31%	40%	28%	23%	15%	18%

Tabela 28 – Chaos Report 2004 **(Hibbs, 2003)**

Como é possível analisar, a taxa de sucesso tem vindo a aumentar a um ritmo baixo.

Alguns peritos não concordam com a forma como os projectos são classificados, visto que muitos dos projectos que estavam como "desafiados" acabaram, no final, por ser produtos com sucesso. Ainda assim a margem para melhorar é grande. **(Hibbs, 2003)**

Problemas com o Desenvolvimento de Software

Como é demonstrado pelo CHAOS Report, existem problemas que afectam, de forma muito séria, a produção de software. Falhas na orçamentação, nos prazos de entrega do produto final e na satisfação das necessidades dos clientes, são os mais comuns. Constituem, também, o reflexo visível da incapacidade de dar resposta séria e capaz aos mercados de muitas empresas de software.

Um dos factores a que se atribui esta elevada taxa de insucesso, é a adopção e uso extensivo do Método da Cascata, nas décadas de 80/90.
Winston Royce, em 1970, descreveu este método num artigo intitulado "Gerir o Desenvolvimento de Grandes Sistemas de Software". Este documento é comummente citado como sendo o estudo que valida o uso do modelo da Cascata. Contudo, já na altura, o autor alertava que "convidava" ao falhanço, recomendando um método iterativo.

1.1. Método da Cascata – (*Waterfall Method*)

O Método da Cascata é uma transposição directa da metodologia aplicada ao desenvolvimento e produção de hardware. Neste, o desenvolvimento divide-se num conjunto de fases distintas, que são executadas sequencialmente:

REQUISITOS?

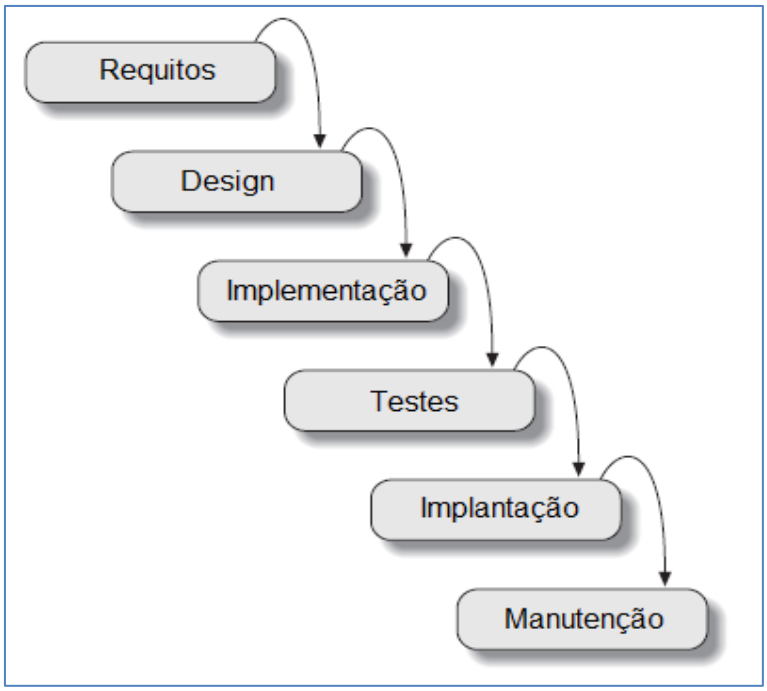


Imagem 60 – Método em Cascata **(Hibbs, 2003)**

O desenvolvimento é visto como uma Cascata, passando de fase em fase e sem possibilidade de voltar atrás.

O problema da Cascata é que assenta em pressupostos muito falíveis, o que, na prática, torna esta metodologia uma ferramenta que raramente funciona:

➔ **É possível identificar todos os requisitos pretendidos**
Só na implementação das funcionalidades iniciais é que vão surgindo situações e/ou funcionalidades necessárias, que não eram "visíveis" no levantamento de requisitos.

➔ **Depois de identificados os requisitos, estes não irão mudar**
Existe um grande intervalo de tempo entre o início de um projecto e a sua possível entrega ao cliente, sendo natural que tenha de proceder-se a alterações aos requisitos iniciais, para que satisfaçam as necessidades do modelo e negócio actual.

➔ **É possível fazer previsões mais ou menos precisas**
Para além da mudança de requisitos, normalmente, as estimativas iniciais de tempo são baixas para garantir o negócio, ou seja, logo à partida, irreais.

➔ **O desenvolvimento é um processo mecânico de passagem do desenho para código**

O desenvolvimento de software não se assemelha a uma linha de produção de qualquer fábrica, onde se podem fazer previsões mais ou menos exactas. Na verdade, existe arte no oficio de programação, o que a torna menos previsível do que aquilo que os gestores gostariam.

A Cascata teria desaparecido lentamente se, nos anos oitenta, não tivesse sido adoptada, pelo departamento de defesa dos Estados Unidos, como standard de desenvolvimento e produção de software. Em 1994 foi substituída por um método com suporte iterativo, mas o mal estava feito e a adopção da metodologia já se tinha generalizado.

Nos anos noventa surgiram métodos alternativos denominados Métodos Leves ("*LightWeight Methods*") e já no início de 2000 surgiram os Métodos Ágeis ("*Agile Methods*"), que começaram a mudar o panorama, mas ainda se estão a dar os primeiros passos e há muito caminho a desbravar.

1.2. Modelo em V

É considerado uma extensão do modelo da cascata e também é usado no desenvolvimento de software. Os passos e a ordem por que está organizado são os mesmos que se verificam no método Cascata, mas há uma grande diferença. Em vez de se percorrer a "cascata" de forma linear, existe um vértice na fase de escrita de código. A razão para isto, reside no facto de se ter verificado existir uma relação entre as fases de desenho e as fases de testes.

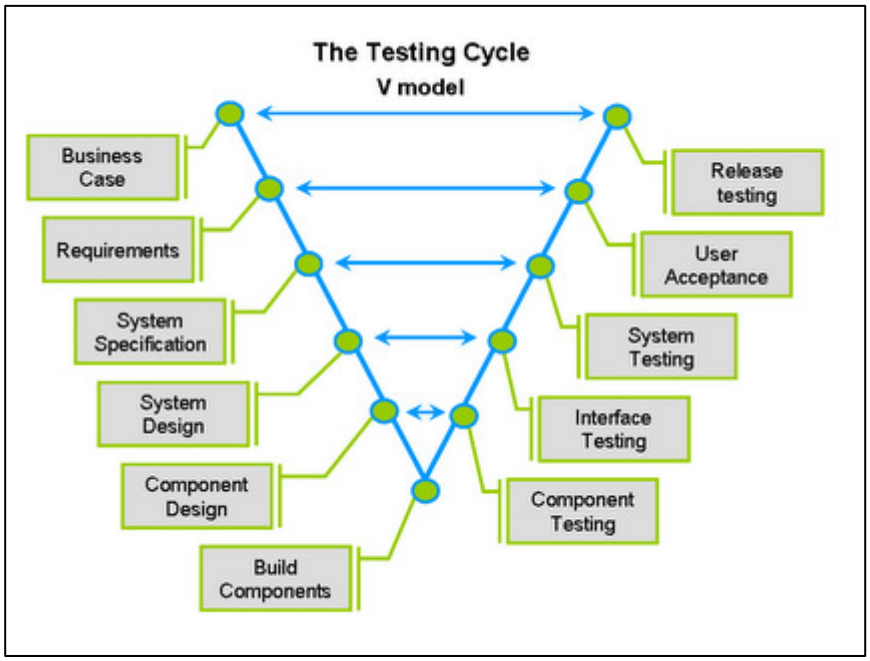


Imagem 61 – Modelo em V **(Craig, 2007)**

A importância dos testes e a "participação" destes no desenvolvimento só foi possível com melhoria e novas técnicas na fase de requisitos.

Outra diferença encontra-se na última etapa. Comparativamente com o método Cascata, deixa de ser a manutenção e passa a ser uma etapa de validação de requisitos, ou seja, verifica-se se estes estão correctos e bem implementados.

No caso de serem precisas alterações e novos requisitos, é necessário percorrer outro ciclo, formando, deste modo, uma sequência de ciclos de desenvolvimento.

1.3. Modelo em Espiral

Este modelo foi definido por Barry Boehm, em 1986, no artigo "*A Spiral Model of Software Development and Enhancement*". Representa a tentativa de combinar o que de melhor há nos conceitos *top-down* e *bottom-up,* integrando características do modelo cascata *(bottom-up*) e do modelo de prototipagem (*top-down*)

Para melhor visualizar, dois eixos perpendiculares definem 4 regiões, que representam quatro tarefas:

- Tarefa de planeamento – Para definir recursos, responsabilidades e calendarizar.
- Tarefa de Definição de Objectives – Para definir os requisitos e limitações para o produto, bem como definir possíveis alternativas.
- Tarefa de Análise de Risco – Para avaliar riscos técnicos e de gestão.
- Tarefa de Engenharia – Para desenhar e implementar um ou mais protótipos.

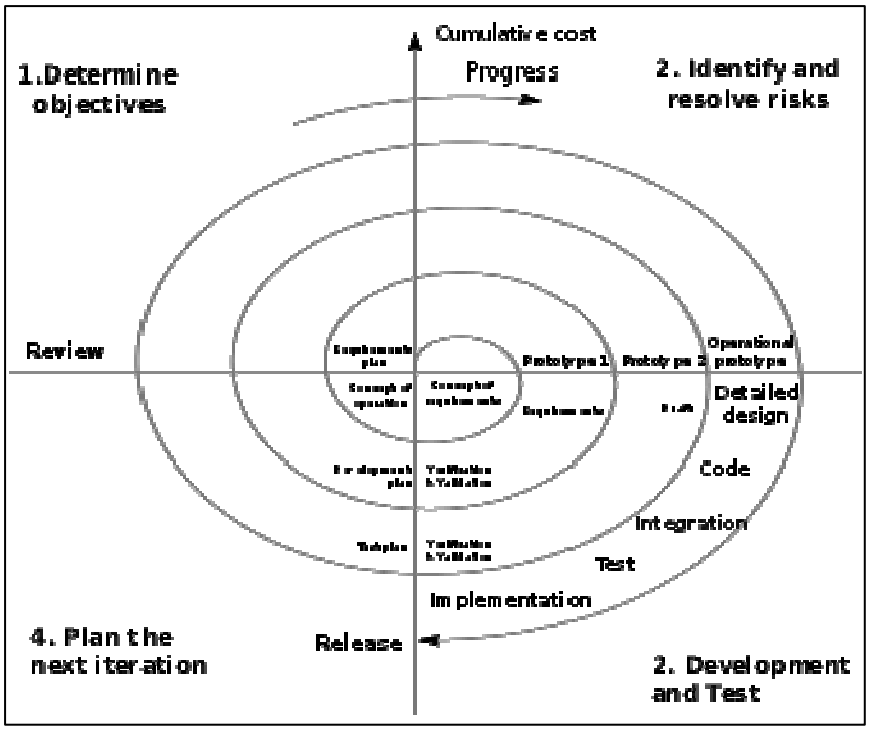


Imagem 62 – Modelo em Espiral **(Boehm, 1988)**

Este modelo tem uma característica que o diferencia. A tarefa de análise de riscos está inserida no processo e é a fase mais importante em todo o desenvolvimento. É nesta etapa que se identificam e se resolvem eventuais riscos no desenvolvimento do projecto.

De forma generalizada, podemos descrever os passos desta metodologia:

➔ Definição dos requisitos, da forma mais detalhada possível.

➔ Este passo é o mais importante. Foi adicionado, especificamente, para identificar e resolver possíveis riscos do projecto. É elaborado um desenho inicial para o projecto, são identificados os possíveis riscos e é feita uma análise às possíveis alternativas, que permitam resolver os mesmos, ou levar a uma melhor solução do ponto de vista financeiro. Nos casos em que a análise de alternativas e de riscos seja mais difícil de determinar, poderá ser desenvolvido um protótipo.

➔ É desenvolvido um primeiro protótipo a partir do desenho inicial. Ainda que seja uma versão "verde" do projecto, já possui as características e funcionalidades do que se pretende para produto final.

➔ Faz-se uma avaliação do primeiro protótipo em termos de ricos, pontos mais fortes e mais fracos. Depois, segue-se o procedimento já usado: definição de requisitos, desenhar e desenvolver o segundo protótipo.

1.4. Prototipagem

Na base desta metodologia, está a obtenção de um protótipo o mais cedo possível, de forma a entender melhor os requisitos e para que o cliente possa validar o implementado. Depois, poderá surgir uma redefinição do protótipo, criando um ciclo até à obtenção do produto final. Esta interacção do cliente durante o ciclo de desenvolvimento, permite, a quem desenvolve, entender melhor o que é pretendido e, ao cliente, avaliar se o que está em desenvolvimento vai de encontro ao esperado e/ou se há mais requisitos que tenham de ser implementados.

Este modelo de desenvolvimento é atractivo, quando se está a lidar com sistemas grandes e complexos, para os quais nada existe que possa servir de base à delineação dos requisitos, deixando que seja o próprio cliente a determinar o caminho a seguir e o que deve ou não ser implementado.

Embora seja um modelo convidativo, há factores de risco que podem pôr em causa o projecto. Após a implementação do primeiro protótipo, a fase de redefinição pode levar à correcção do sistema. Estas correcções poderão elevar, em demasia, a complexidade, pois há a possibilidade das modificações irem muito além do que inicialmente foi esperado.

1.5. Desenvolvimento Iterativo e Incremental

É a base das actuais metodologias, as quais possuem um processo de desenvolvimento de software cíclico. Começa com uma planificação inicial e, após N ciclos iterativos, acaba com a instalação da solução final.
A ideia base é desenvolver um sistema, usando ciclos repetitivos com pequenos incrementos. Desta maneira, é possível corrigir e evitar erros cometidos em ciclos anteriores, assim como ter a percepção de novos requisitos, ou soluções alternativas ao inicialmente planeado.

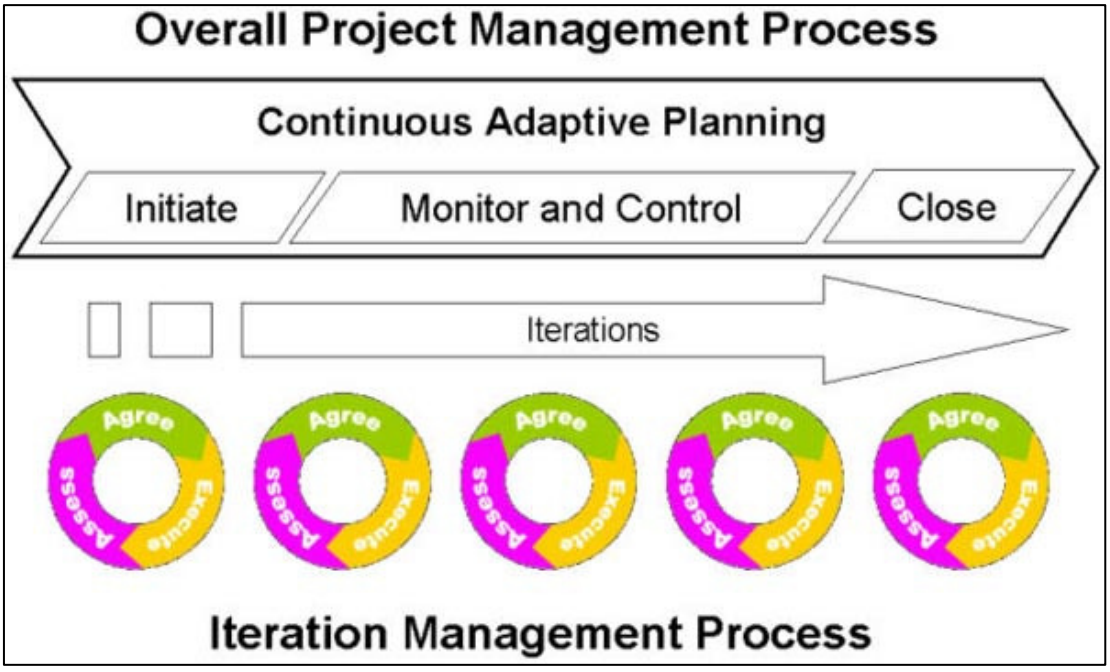


Imagem 63 - Desenvolvimento iterativo e incremental **(Spence, 2005)**

A abordagem é muito semelhante ao já visto no ponto anterior. O primeiro passo consiste em estabelecer um plano inicial, sendo criada uma primeira versão do projecto, á semelhança do que acontece no modelo de prototipagem e com os mesmos propósitos.

Uma das diferenças para o modelo de prototipagem, reside no facto de ser realizado um planeamento das tarefas que devem ser implementadas, o qual servirá de guia para a fase iterativa. Depois, será necessário corrigir o plano inicial, melhorar e mesmo incrementar, dependendo das observações e obstáculos se encontrarem na implementação e teste do protótipo.

Outra diferença é que esta metodologia é incremental, ou seja, embora no plano inicial se descreva o objectivo final, a cada ciclo é feita a implementação de alguns elementos. Este factor permite evitar que haja necessidade de alterar muita da lógica já criada, para implementar funcionalidades não visíveis no plano inicial, ter a percepção, em fase embrionária, de erros de implementação ou tarefas que não têm razão de existir, bem como outros aspectos que, de uma ou de outra forma, iriam ter efeitos negativos na utilização do sistema.

1.6. Desenvolvimento de Software Ágil

Nos anos 90, havia uma insatisfação crescente com a prevalência de metodologias pesadas, como, por exemplo, a cascata. Estas não respondiam a problemas persistentes, nomeadamente, elevada taxa de projectos falhados, pouca qualidade de software, elevados custos de desenvolvimento e rigidez quanto à mudança de requisitos, durante o processo de desenvolvimento.

A necessidade de alterar este cenário fez com que começassem a aparecer metodologias, que ficariam conhecidas como metodologias "peso leve" (*Lightweight*). ***Scrum*** (1995) ou ***Extreme Programming*** (1996) são bons exemplos do esforço inicial para mudar a situação do desenvolvimento de software.

Cada uma das ferramentas tinha uma combinação de diferentes ideias: *ideias mais antigas, ideias antigas transfiguradas e novas ideias*. Todas tinham um conjunto de pontos-chave em comum:

- ➔ Enfatizar a colaboração próxima entre a equipa de desenvolvimento e os peritos de negócio;
- ➔ Comunicação cara a cara, mais eficaz que documentação escrita;
- ➔ Entrega frequente de um produto, cuja implementação adiciona valor;
- ➔ Equipas unidas e auto-organizadas;
- ➔ Escrita do código, de modo a que a "mistura" provocada pelos requisitos não seja uma crise."

Todas as ferramentas eram muito idênticas. Em Fevereiro de 2001, reuniu-se, durante dois dias, um grupo de notáveis na área de desenvolvimento de software, evento que ficou conhecido como reunião Snowbird (Art of software development).

Desta reunião saíram três aspectos significantes:

- ➔ **O uso do termo Ágil**. Muitos dos mentores das metodologias não gostavam do nome "peso leve", pois dava a ideia de superficial, o que poderia induzir a algo mais "amador";

- ➔ **O Manifesto Ágil**. Descreve os quatro pilares, em que assentam as filosofias de todas as metodologias ágeis (8.x);

- ➔ **A Aliança** (http://www.agilealliance.org/). Uma organização, sem fins lucrativos, que existe para adicionar, actualizar e disseminar informação acerca de processos ágeis.

1.7. Manifesto Ágil (Agile Alliance, 2011)

"O manifesto Ágil" foi escrito em Fevereiro de 2001, numa cimeira de dezassete praticantes de diversas metodologias de programação e com ideologias independentes. Os participantes, embora não tenham concordado em muitos pontos, encontraram consenso em torno de quatro ideias chave:

- ➔ **Indivíduos e interacções, acima de processos e ferramentas;**
- ➔ **Software funcional, acima de documentação compreensível;**
- ➔ **Colaboração dos clientes, acima de negociação de contrato;**
- ➔ **Resposta a mudanças, acima do cumprimento de um plano.**

Isto significa que, não obstante a importância dos itens da direita, os itens da esquerda têm ainda muito mais valor.

Analisemos agora melhor o que estas 4 directrizes significam:

➔ **Indivíduos e interacções, acima de processos e ferramentas**:

Deve valorizar-se a maneira como as equipas de desenvolvimento interagem entre si e o modo como cada um desenvolve o seu trabalho, em detrimento de tentar sistematizar, recorrendo a ferramentas e implementando processos, ainda que importantes.

A sistematização em produção tem o objectivo de facilitar a gestão, mas, em software, deve ser sempre um elemento facilitador e não um obstáculo aos membros da equipa de desenvolvimento.

➔ **Software funcional, acima de documentação compreensível**

É importante que exista documentação compreensível para servir de guia, mas, para o cliente, é muito mais importante haver um software funcional. Assim, será primordial investir tempo em desenvolvimento, até porque, do ponto de vista do utilizador, a aprendizagem é realizada, maioritariamente, a interagir com o sistema e não a ler documentos explicativos das potencialidades do software.

➔ **Colaboração dos clientes, acima de negociação de contrato**

Um contrato é sempre necessário para estabelecer regras, direitos e deveres das partes envolvidas, mas o maior investimento deve ir no sentido de englobar o cliente no processo de desenvolvimento.

Para quem se encontra a desenvolver o sistema, é muito importante perceber o que é que o cliente espera do produto, a maneira como interage com o sistema e também "educar" o cliente a atingir o que pretende, mostrando-lhe caminhos alternativos.

➔ **Responder a mudanças, acima de seguir um plano.**

Quando se inicia a produção de um sistema é impossível fazer um plano com tudo o que é necessário.
Há diversos factores que podem provocar mudanças, desde a especificação inicial já não ser adequada à realidade de negócio, até modificações nas tecnologias e/ou alterações na vontade do cliente.

Assim, é necessário que a prioridade seja conseguir responder a mudanças, em prejuízo de seguir o previamente planeado.

Contudo, o estabelecimento de um plano é uma ferramenta essencial para que o desenvolvimento de software seja feito na direcção correcta. Este tem de ser maleável e permitir mudanças sem que se torne desadequado e, em consequência, irrelevante para o desenvolvimento do sistema.

Para ser possível alcançar os quatro "mandamentos" acima descritos, ficaram estabelecidos princípios que devem ser seguidos:

- A maior prioridade é, *desde as primeiras etapas do projecto,* satisfazer o cliente através da entrega rápida e contínua de software com valor;

- Aceitar alterações de requisitos, mesmo numa fase tardia do ciclo de desenvolvimento. Os processos ágeis potenciam a mudança, em benefício da vantagem competitiva do cliente;

- Fornecer, frequentemente, software funcional. Os períodos de entrega devem ser de poucas semanas a poucos meses, dando preferência a períodos mais curtos;

- Durante o decorrer do projecto, o cliente e a equipa de desenvolvimento devem trabalhar juntos, diariamente;

- Desenvolver projectos com base em indivíduos motivados, dando-lhes o ambiente e o apoio de que necessitam, confiando que irão cumprir os objectivos;

- O método mais eficiente e eficaz de passar informação para dentro de uma equipa de desenvolvimento, é através de conversa pessoal e directa;

- A principal medida de progresso é a entrega de software funcional;

- Os processos ágeis promovem o desenvolvimento sustentável. Os promotores, a equipa e os utilizadores deverão ser capazes de manter, indefinidamente, um ritmo constante;

- A atenção permanente à excelência técnica e um bom desenho da solução, aumentam a agilidade;

- Simplicidade – *a arte de maximizar a quantidade de trabalho que não é feito* – é essencial*;*

- As melhores arquitecturas, requisitos e desenhos surgem de equipas auto-organizadas;

- A equipa reflecte regularmente sobre o modo de se tornar mais eficaz, fazendo as adaptações e os ajustes e necessários.

2. Desenvolvimento de Software Lean (LSD – Lean Software Development)

"Lean e Ágil não serão dois nomes para a mesma coisa?

Esta é uma pergunta e um equívoco, frequentes. A resposta curta é: não, não são a mesma coisa.

Mas, certamente, deseja-se uma resposta pormenorizada.
É fácil ver o porquê do equívoco. Lean e Ágil partilham os mesmos objectivos, nomeadamente, aumentar, em simultâneo, a produtividade do desenvolvimento de software e a qualidade deste. Para compor ainda mais o equívoco, várias metodologias ágeis suportam os princípios Lean.

Ágil tem uma perspectiva diferente do Lean e, de uma forma geral, uma visão mais estreita. Reciprocamente a visão do Lean é mais larga, preferindo olhar para todo o contexto de negócio, em que se enquadra o desenvolvimento de software. O Lean vê as metodologias de desenvolvimento de software Ágil, como válidas para suportar as suas práticas de desenvolvimento de software.
" **(Hibbs, 2007)**

O conceito de desenvolvimento de software Lean foi introduzido, pela primeira vez, por Mary e Tom Poppendieck, no livro *"Lean Software Development: An Agile Toolkit*" em 2003.

Neste livro, os autores transpõem os princípios do TPS e da produção Lean para o desenvolvimento de software, apresentando um conjunto de 22 ferramentas.

Em 2006, os mesmos autores apresentaram um novo livro intitulado "*Implementing Lean Software Development*", no qual agruparam as ferramentas em 7 princípios e práticas. É sobre estas que, hoje em dia, assenta o LSD.

2.1. Princípio 1: Eliminar o desperdício (muda)

O principal objectivo do Lean é eliminar o desperdício. É necessário, portanto, saber ver o desperdício, consoante a área de negócio em que se encontra a ser aplicado.

No primeiro livro de Mary e Tom, a primeira ferramenta faz uma correspondência entre os desperdícios da produção Lean e os desperdícios no desenvolvimento de software:

Processamento → Processos desnecessários:

Nada de diferente. Etapas que não adicionem valor, quer seja em produção automóvel, quer seja em desenvolvimento de software, são puro desperdício. Desde a elaboração de manuais, que ninguém irá ler, até procedimentos que apenas tornam difíceis as tarefas mais simples.

Defeitos → Defeitos:

O único aspecto diferente é o modo como se lida com o defeito.
Em produção, investiga-se a causa e altera-se o processo, para que o defeito não se repita. Previne-se o mesmo, usando mecanismos de avaliação da qualidade das peças (passa - não passa).

Em desenvolvimento de software, são usados testes automáticos para não deixar passar os erros e, quando se detecta que um erro passou, adiciona-se mais um teste aos já existentes, para que a situação não volte a repetir-se.

Deslocamento → mudança de tarefa

Mudar de tarefa representa mudar o contexto do problema, que é preciso resolver. Ainda que num cenário, em que as ferramentas usadas são as mesmas (linguagem, ide, base de dados, etc.), demora-se algum tempo até conseguir a concentração necessária ao desenvolvimento do novo trabalho.

No caso de se retomar uma tarefa que se havia interrompido, tem de passar-se pela fase de perceber/ recordar o que estava a ser feito.

Todos estes factores geram interrupções no desenvolvimento e, consequentemente, desperdício de tempo.

Sock → Trabalho parcialmente feito:

Aqui o stock é representado por lotes de trabalho, que está só parcialmente feito, verificando-se a existência de requisitos não implementados na sua totalidade, testes por efectuar, bugs por corrigir, documentação por realizar, etc. O Lean procura uma corrente contínua, desde o pedido do cliente até à obtenção do produto final e, no caso do software, a implementação do mesmo.

Espera → Atrasos:

No desenvolvimento de software é frequente aparecerem dúvidas de natureza diversa.

Quando o programador se depara com uma dúvida e consegue obter resposta rápida para a mesma, o fluxo de desenvolvimento é contínuo.

No entanto, se a não consegue (resposta rápida), pode mudar para outra tarefa ou tentar adivinhar a solução, o que, em qualquer dos casos, significa desperdício. É precisamente isso que o Lean pretende eliminar.

Transporte → Passar a:

O deslocamento, no caso do desenvolvimento de software, é representado pela passagem do projecto entre os diversos intervenientes. Aqui, o desperdício está associado à perda de informação e de conhecimento adquirido.

Há perda de conhecimento, quando o projecto muda de mãos, quer seja entre elementos do desenvolvimento (em geral, os programadores têm diferentes métodos de escrita de código e, por muito que documentem, o segundo programador leva algum tempo a entender toda a sequencia e intenção do seu antecessor), quer seja entre processos (exemplo: quando um projecto chega à equipa de programação, perde-se uma boa percentagem do conhecimento inicial, adquirido nas primeiras conversas com o cliente).

Em conclusão, sempre que possível, deve evitar-se a passagem do projecto.

Superprodução → Funcionalidades Extra:

A superprodução é um dos maiores problemas, senão mesmo o maior, do desenvolvimento de software. Funcionalidades extra representam mais linhas de código a serem escritas, mais código para testar, manter e documentar. Durante o desenvolvimento é comum ouvirem-se expressões do género: “Já agora…”, “É só mais…” e, muitas vezes, estas expressões representam implementações que o cliente não precisa, nem vem a precisar.

A regra dos 80/20 aplica-se à maioria dos softwares: 80% daquilo que o cliente pretende, é fornecido por 20% das funcionalidades do produto, o que significa que 80% das funcionalidades, raramente ou nunca são usadas.

Na conferência XP de 2002, o Grupo Standish relatou que 45% das funcionalidades nunca eram usadas e que apenas 20% eram usadas frequentemente. Aliás, o cenário era ainda pior no relatório CHAOS, já descrito no ponto 1, no qual se diz que 64% das funcionalidades nunca ou raramente eram usadas.

Nos seus livros, Mary e Tom chamam a atenção para o perigo que representa a ideia de que, com uma especificação antecipada, se consegue reduzir o desperdício.

O cliente, quando confrontado com a necessidade de, logo à partida, indicar tudo o que pretende, sob pena de ter que repetir pedidos de desenvolvimento, cabendo-lhe os respectivos encargos, irá "despejar" tudo o que julga que pode ser útil e irá esquecer coisas que são importantes. Ou seja, vai ser desenvolvido um software que contempla funcionalidades sem interesse e que vai necessitar de alterações, para implementação de outras (funcionalidades), que foram esquecidas, pelo que não fazem parte do contrato inicial.

1.2 Princípio 2: desenvolver com qualidade

Como já foi visto em pontos anteriores, existem duas alturas para tratar os erros: durante a produção, prevenindo os mesmos e, depois, no departamento de qualidade, na detecção de defeitos. Se os houver, são analisados, para verificar as respectivas causas e inserir novas ferramentas, que impeçam que voltem a acontecer.

A melhor forma de verificar que existem defeitos é a existência de trabalho parcialmente feito e de correcção de erros. Do ponto de vista do Lean, a existência de "stock" é desperdício e por isso é de evitar.

Já existem muitas ferramentas/metodologias, que permitem que erros no código não passem despercebidos. Para além de ferramentas integradas no próprio IDE, o "desenvolvimento guiado por testes" (*test-driven development TDD*) é uma técnica muito eficaz para a sua prevenção.

No TDD, o programador introduz no sistema, tanto o código, como o teste correspondente e só adiciona novo código, para nova tarefa, quando o anterior passar no teste. No final do dia o programador corre os testes de forma mais completa e, no final de cada semana, os testes são reunidos para que possam ser corridos de forma abrangente.

1.3. Princípio 3: Criar conhecimento

"Um dos aspectos intrigantes da metodologia cascata é a ideia de que o conhecimento existe na forma de requisitos e é anterior à escrita de código**.** O desenvolvimento de software é um processo de criação de conhecimento. Enquanto o conceito de arquitectura do sistema, no seu todo, vai ser idealizado antes da escrita de código, a validação daquela (arquitectura) vem, precisamente, na codificação, ainda que o desenho tenha sido muito detalhado. Um desenho antecipado não pode antever a complexidade encontrada durante a implementação, nem consegue ter em conta o feedback, que vem com a construção do software. Pior, realizar cedo um design detalhado torna-se avesso ao feedback dos intervenientes e dos clientes. Um processo de desenvolvimento focado em criar conhecimento, esperará que o desenho se desenvolva durante a escrita do código e não irá desperdiçar tempo, limitando a criação de conhecimento a uma fase muito prematura." **[30](Poppendieck, 2006)**

O uso de comentários e metodologias, que permitam obter mais rapidamente feedback, são exemplos de como é possível adquirir e documentar o conhecimento, não só para o projecto em curso, mas também para projectos futuros.

1.4 Princípio 4: Adiar o compromisso

O normal no desenvolvimento de software é tentar codificar o sistema, para que, no futuro, todos os processos possam ser modificados. No entanto, há situações e pontos de decisão, que marcam o caminho que tem de ser seguido e que representam situações irreversíveis, ou situações reversíveis, mas que exigem alterações de custos elevados.

Para tomar decisões irreversíveis ou de retrocesso a custo muito elevado, deve-se esperar, responsavelmente, até ao último momento (a escolha do momento certo deve resultar de uma avaliação e de uma estimativa do máximo de tempo que se poderá gastar para tomar a decisão), de modo a conseguir-se angariar a maior quantidade possível de informação. Por exemplo: verificar, através da realização de testes, se determinada solução responde ao que é pretendido e, só em caso positivo, englobar a mesma no sistema que se encontra em desenvolvimento.

> "Equipas de emergência são treinadas para lidar com situações imprevisíveis, desafiantes e, muitas vezes, perigosas. Os seus elementos são ensinados a avaliar as situações e a determinar o tempo de que dispõem até à altura certa para tomar decisões críticas. Assim, estabelecendo um intervalo de tempo, são treinados a esperar, até ao limite, para pôr em acção a decisão tomada, pois é a altura em que têm a maior informação possível.
>
> Nós devíamos aplicar a mesma lógica às decisões irreversíveis que temos de tomar durante o curso do desenvolvimento de software…" **(Poppendieck, 2006)**

1.5. Princípio 5: Entrega rápida

O uso de metodologias que admitem uma entrega rápida, feita com a implementação de pequenos conjuntos de funcionalidades, de per si, permite ter vantagens no processo de desenvolvimento de software.

O desenvolvimento com ciclos pequenos dá a possibilidade de obter um feedback rápido, por parte dos intervenientes e dos clientes. Permite perceber melhor o que está bem implementado, o que precisa de ser mudado e mesmo o que é supérfluo, não só no que já foi desenvolvido, mas também nos requisitos a ser desenvolvidos.

1.6. Princípio 6: Respeitar as pessoas

"Quando Joel Spolsky finalizou o curso e era um novo empregado da Microsoft, foi-lhe atribuída a tarefa de desenvolver uma estratégia destinada a linguagem macro, para o Excel. Ele gastou algum tempo para chegar à conclusão do que é que os clientes poderiam querer e, no final, elaborou as especificações.

Para sua surpresa, a situação chegou aos ouvidos de um grupo de arquitectura de aplicações, que quis conhecer as especificações. Joel pensou que lhe quisessem dar bons conselhos, mas constatou que os elementos do grupo sabiam menos que ele, muito embora se tratasse de quatro doutorados e um chefe de alto perfil (um amigo de Bill Gates, que era qualquer coisa como empregado número seis).

Apesar de terem apenas uma ideia superficial de como é que os clientes poderiam usar as macros, o grupo entendia que era da sua competência determinar o modo de implementação das mesmas (macros). Os gestores de Joel esclareceram que a decisão era dele e que a sua equipa também o apoiou. Porém, o grupo de arquitectura de aplicações não ficou agradado com a situação e o seu chefe convocou uma grande reunião, para denunciar que Joel estava a perverter a estratégia das macros.

Se a Microsoft fosse uma companhia vulgar, poder-se-ia adivinhar o fim desta história. O grupo de arquitectura de aplicações "ganharia" e o Joel teria de submeter-se ou sair, mas não foi isso que aconteceu. Um dia após a reunião, um vice-presidente sénior encontrou-se com Joel na cafetaria e perguntou-lhe como estava a situação, relativamente ao grupo de arquitectura de aplicações. Claro que Joel respondeu que estava tudo bem.

No entanto, no dia seguinte ouviu dizer que o grupo havia sido desfeito." **(Poppendieck, 2006)**

Na produção Lean e no TPS uma das metodologias é, precisamente, criar o sentimento de compromisso no trabalhador, ouvindo-o, aceitando e experimentando as suas sugestões e, no caso das mesmas resultarem, adoptá-las no processo de produção. Esta história exemplifica esse princípio.

O respeito pelas pessoas significa confiar que elas sabem como e qual a melhor maneira de desenvolver o seu trabalho, permitindo-lhes que demonstrem erros nos processos actuais e que dêem sugestões para melhorias.

1.7. Princípio 7: Optimizar o todo

Este princípio é, nada mais, nada menos, o que já está no conceito base da produção Lean. Melhorar apenas determinadas etapas do processo de desenvolvimento e de forma "desgarrada", melhora alguma coisa, mas os resultados perdem-se e acabam por não ter grande significado. O processo tem de ser visto como um todo, desde o pedido do cliente até à instalação do sistema.

A forma mais eficiente de o fazer é utilizar, por exemplo, as ferramentas que se usam na produção VSM, que permite ver o fluxo de desperdícios nos e entre processos, não só no contexto de produção, mas também no contexto do negócio.