Departamento de Ciências e Tecnologias da Informação

Industrialização do Processo de Desenvolvimento de Software Software Product Lines

Bruno Miguel Saraiva Duarte da Cal

Dissertação submetida como requisito parcial para obtenção do grau de

Mestre em Engenharia Informática

Orientador:

Doutor Henrique O'Neill, Professor Associado,
ISCTE-IUL

Junho de 2011

AGRADECIMENTOS

Gostava de iniciar esta secção, com um sincero agradecimento aos colaboradores, investigadores e professores envolvidos na instituição que me disponibilizou todas as condições para realizar este trabalho, a ADETTI. Em particular, gostava de deixar uma palavra especial aos colegas do grupo Information Systems Research, que ao longo do trabalho, contribuíram com opiniões e comentários extremamente úteis e construtivos.

Ao meu orientador, Professor Henrique O'neill, gostava de agradecer a sua constante disponibilidade e de manifestar o imenso prazer que tive em trabalhar com ele. Para além da aplicação neste trabalho, não tenho qualquer dúvida, que a perspectiva de investigação aplicada que me ensinou, constitui e constituirá um importante activo na minha vida profissional.

Gostava ainda de agradecer ao Professor Mauro Nunes e ao Professor André Santos, pelo apoio, ensinamentos e comentários que muito ajudaram a concretizar este trabalho.

Por último, gostava ainda de agradecer à minha família e amigos, os incentivos e compreensão que tiveram em alguns momentos mais críticos deste trabalho.

RESUMO

Desde os primeiros tempos da engenharia de software, investigadores (e praticantes) dedicam-se à análise do processo de desenvolvimento de software, avaliando os seus problemas e possíveis soluções. Vários factores devem ser envolvidos nesta análise, todavia será certamente seguro afirmar que a falta de reutilização de desenvolvimento e a baixa produtividade do processo de construção de uma aplicação, são duas preocupações recorrentes cuja resolução não se tem mostrado simples.

Na última década, este contexto tem motivado a comunidade de investigação a analisar o que designam por industrialização do processo de desenvolvimento de software. Esta vertente de investigação defende que várias técnicas industriais podem ser transpostas para o desenvolvimento de software e que a sua aplicação trará enormes benefícios à abordagem tradicional, nomeadamente, potenciará substanciais ganhos de produtividade, diminuição de custos e melhorias de qualidade dos sistemas desenvolvidos.

As Software Product Lines são uma materialização desta perspectiva industrializada do processo de desenvolvimento. O fundamento principal desta orientação consiste numa mudança de paradigma de desenvolvimento que substitui o modelo tradicional, focado na construção de uma aplicação específica para um determinado cliente, para a construção de uma infra-estrutura de produção que suporte o desenvolvimento de um leque de sistemas (família de produtos) que sirvam um conjunto de clientes. Segundo os defensores deste novo paradigma, esta visão multi-sistema, permitirá uma reutilização massiva e sistemática de vários componentes entre os elementos da família de produtos, que servirá de base para a optimização do processo de desenvolvimento associada aos ganhos de produtividade, diminuição de custos e melhorias de qualidade pretendidos.

Esta dissertação procura sistematizar os conceitos envolvidos nas Software Product Lines através de uma análise abrangente dos princípios teóricos subjacentes e da sua concretização prática através de técnicas e tecnologias existentes. Com base nesta sistematização de conhecimento, propomos um modelo de desenvolvimento baseado em famílias de produtos (SPLUP), cuja aplicabilidade demonstramos através da elaboração de um caso de estudo e construção de protótipo.

Complementarmente, foram submetidos dois artigos científicos a conferências científicas (CENTERIS 2011 e CAPSI 2011) que sistematizam as principais conclusões e contribuições do nosso trabalho.

Palavras chave

Industrialização; Linhas de produto de software; Ciclo de vida de desenvolvimento de software; Engenharia de software; Gestão de projecto; Desenvolvimento orientado por modelos; Automatização de desenvolvimento.

ABSTRACT

Since the early days of Software Engineering, researchers and practitioners have sought to improve the software development process. The lack of reuse and the low productivity rates are recurring concerns whose resolution has not proved to be simple.

Over the last decade, the research community and industry have been studying the industrialization of the software development process. The ideas underlying this perspective are supported by the proposition that several industrial techniques should be compliant with software development, and that its implementation will bring substantial benefits to the traditional approach, namely, considerable productivity gains, costs reductions and quality improvements to the developed systems.

Software Product Lines materialize this "industrialized" version of the software development. The main baseline of this approach, consists on the shift from the present model, in which a system is built from a set of needs stated by a single customer, to a model that aims to the construction of a generic production platform that supports the development of a set (family) of systems that covers a related set of customers' needs. According to the proponents of this theory, the multi-system approach will allow a massive and systematic component reutilization within the family members, which contributes to a considerable process optimization.

This thesis attempts to organize the concepts related to Software Product Lines through a comprehensive analysis of the underlying theoretical principles and its practical implementation through existing techniques and technologies. Based on this knowledge, we propose a family-oriented development model (SPLUP), whose applicability is demonstrated by a case study and prototype development.

In addition, two papers were submitted (CENTERIS 2011 and CAPSI 2011 conferences) to systematize the main conclusions and contributions of our work.

Keywords

Industrialization; Software product line; Software development lifecycle; Software engineering; Project management; Model driven development; Software development automation;

ÍNDICE

ÍNDICE DE TABELAS

ÍNDICE DE FIGURAS

1 INTRODUÇÃO

1.1 ENQUADRAMENTO

Por invulgar que possa parecer, a temática abordada neste trabalho, realizado em pleno século XXI, é perfeitamente enquadrada pelo artigo *Mass Produced Software Components* (McIlroy, 1968) de Doug McIlroy, publicado em 1968 na *1st Conference on Software Engineering* na Alemanha. As afirmações que utilizamos para introduzir o trabalho têm mais de 40 anos, o que nesta área é, no mínimo, bastante tempo, mas ainda mantêm uma validade impressionante.

A indústria de software não está industrializada

Actualmente, a indústria de *software* pode ser vista como um sector de uma indústria maior, as tecnologias de informação. Como sugere McIlroy, a industrialização do sector do *software* é questionável e as suas deficiências são colocadas em evidência quando comparadas com os restantes sectores, em particular com o sector do hardware. Ainda que a comparação deva ser ajustada pelas diferenças inerentes ao tipo de produto, a realidade é que dos actuais sectores TI, o *software* é aquele que apresenta processos de produção mais artesanais.

Inquestionavelmente, desenvolvemos software com técnicas pouco avançadas

Muitos progressos foram feitos nestes últimos 40 anos: institutos dedicam-se, em exclusivo, à normalização e estandardização de processos e tecnologias; normas de qualidade específicas para *software* são uma realidade; padrões e boas práticas são amplamente divulgados; a comunidade de investigação contribui activamente; etc...

Paradoxalmente, o ciclo de vida de desenvolvimento de aplicações personalizadas (tipicamente sistemas de informação organizacionais) - vulgo recolha de requisitos, análise e desenho, implementação, testes e instalação – permanece sem grandes alterações. Continuamos a abordar os projectos com desenvolvimentos dedicados não reaproveitáveis, continuamos a empregar demasiado esforço em tarefas repetitivas, continuamos a desenvolver aplicações pouco evolutivas.

Os progressos de técnicas e metodologias têm compensado a crescente exigência colocada pelos sistemas de informação. No entanto, é uma realidade para muitos autores (e praticantes) que chegámos a um ponto de estagnação insustentável. É necessária uma mudança de paradigma.

Já utilizámos vezes suficientes a abordagem "Que aspectos devemos construir?" para passarmos a pensar em "Que aspectos devemos reutilizar?"

A reutilização é um tema recorrente e, enquanto conceito, geralmente bem aceite. No entanto os resultados da sua aplicação ficam muitas vezes aquém das expectativas. Tipicamente reutilizamos partes de desenvolvimento, utilizamos funcionalidades de uma biblioteca ou "copiamos" um pedaço de código, aplicamos um padrão de desenho que conhecemos. Todavia, não passam de reutilizações *ad-hoc*. À semelhança do que se passou

noutras indústrias precisamos de sistematizar a reutilização se pretendemos alcançar o seu pleno potencial.

Este enquadramento, ou parte dele, constitui a base motivacional de vários métodos, técnicas ou tecnologias que, de algum modo, propõem novas formas de desenvolvimento de aplicações (por exemplo, desenvolvimento baseado em componentes, prototipagem rápida ou *Model-driven Architecture*). Infelizmente, a adopção destas propostas nem sempre resulta nos ganhos desejados. Uma razão possível para este resultado, é o carácter localizado dessas abordagens, em particular, a ausência de uma transformação do processo de desenvolvimento que acompanhe e suporte a sua utilização.

Cientes do problema e da necessidade de reestruturar o modelo tradicional, a comunidade científica e empresas desenvolveram e adoptaram nos últimos anos um processo de desenvolvimento inspirado em técnicas e processos industriais, o desenvolvimento baseado em linhas de produto (*Software Product Lines* – SPL). Este tipo de processo assume como objectivo central a reutilização em grande escala. Se atentarmos à história recente, este mote é coincidente com o processo de industrialização que fez com que vários sectores, maioritariamente artesanais, se transformassem nas indústrias modernas que hoje são (e.g. vestuário, alimentação ou mobiliário).

Seguindo o percurso histórico dessas indústrias conseguimos identificar alguns momentos importantes que permitiram que chegassem ao estado de maturidade de desenvolvimento que hoje apresentam. O desenvolvimento de produtos pela montagem de componentes reutilizáveis, a criação de linhas de montagem ou a especialização de trabalhadores em ferramentas e processos específicos para a produção visada, são alguns exemplos desses lugares comuns.

A aplicação destes ensinamentos à construção de aplicações informáticas não é linear. Se por um lado a nossa sociedade não é a mesma da época da industrialização, também o produto não apresenta as mesmas características materiais e tangíveis. Contudo, alguns sectores TI têm demonstrado que esta transposição é possível e bastante positiva. O exemplo mais divulgado desse sucesso é sector das aplicações *embedded* (sistemas específicos para um determinado equipamento). Os resultados obtidos neste sector são entusiasmantes, com ganhos de produtividade e qualidade até 10x ou a diminuição de custos na ordem dos 60% (Institute, 2010a).

Este trabalho analisa o desenvolvimento baseado em linhas de produto e as principais técnicas e tecnologias que podem apoiar a sua implementação. Esta análise permitirá atingir o principal objectivo deste trabalho, a definição de uma metodologia de desenvolvimento baseada em famílias de produto que seja suficientemente genérica para ser aplicada num largo espectro de domínios mas também específica o suficiente para fornecer um trilho concreto de concretização e não um conjunto abstracto de princípios.

1.2 PROBLEMA

A nossa sociedade está cada vez mais dependente dos sistemas de informação. Vivemos na era digital onde poucas coisas no nosso dia-a-dia não recorrem, directa ou indirectamente, a aplicações informáticas.

Esta dependência contrasta com os problemas do sector. Existe uma percentagem considerável de projectos de desenvolvimento de *software* que não terminam como o esperado, i.e. não cumprem o âmbito, orçamento ou calendário definido. A agravar a situação acrescenta-se o facto dos valores destes indicadores estarem praticamente estagnados há mais de 10 anos (Eveleens & Verhoef, 2010).

A baixa produtividade, os custos elevados e a reduzida qualidade das aplicações caracterizam, em muitos casos, o cenário de desenvolvimento de sistemas de informação. Estes factores resultam directamente da forma como abordamos o desenvolvimento. A referida crescente "dependência tecnológica", exige que se reflicta sobre estes factores, se questione o que os causa e se avalie as alternativas que temos para os resolver.

Torna-se então imperativo encontrar soluções para optimizar o processo de desenvolvimento de software por forma a melhorar os indicadores associados ao seu sucesso, e assim conseguir, por exemplo, diminuir tempos de entrega de soluções, promover uma melhoria da adequação problema-solução ou melhorar a qualidade e capacidade de evolução dos sistemas.

1.3 MOTIVAÇÕES

A motivação principal para a escolha do tema de dissertação foi a percepção de que o problema a abordar é real. Com alguma experiência no desenvolvimento de *software*, passamos por contratempos que resultam directamente das ineficiências do processo de desenvolvimento a que estamos habituados. Este "hábito" protelou mudanças mas a verdade é que perante a complexidade e exigência crescente do mundo e, consequentemente dos sistemas de informação, as limitações do processo de desenvolvimento tradicional são cada vez mais evidentes e impossíveis de sustentar.

A adopção de técnicas e práticas industriais ao processo de desenvolvimento de aplicações pareceu-nos um bom ponto de partida para endereçar o problema. Ainda que o carácter imaterial do produto - uma aplicação informática – limite a comparação e a adaptação de técnicas e que exista um verdadeiro processo criativo associado a cada projecto, a verdade é que a crescente industrialização do sector nos parece um passo inevitável.

Numa óptica académica, a atenção de que esta área tem sido alvo por parte de várias comunidades de investigação, que têm colaborado com um conjunto de abordagens e técnicas bastante consistentes e promissoras, motivou-nos a dar o nosso contributo.

1.4 OBJECTIVOS

O propósito central da dissertação é a apresentação de um modelo para um processo de desenvolvimento baseado em linhas de produtos de *software* (SPL). Este modelo deverá reflectir as boas práticas deste tipo de abordagem mas também fornecer instruções técnicas claras para guiar a necessária aplicação de diversas tecnologias. Esse intento leva-nos a considerar os seguintes objectivos para este trabalho:

1. Apresentação e análise dos conceitos, abordagem, técnicas e tecnologias potencialmente aplicáveis num processo SPL;
2. Elaboração de um modelo de desenvolvimento baseado em linhas de produtos com forte componente prática e técnica;
3. Construção de um caso de estudo para avaliação do modelo proposto;
4. Implementação de um protótipo (no âmbito do caso de estudo) que demonstre o funcionamento das tecnologias consideradas.

1.5 METODOLOGIA DE INVESTIGAÇÃO

A metodologia de investigação adoptada considera quatro fases essenciais:

- Recolha de informação que permita conhecer o problema em causa e os modelos e técnicas que sustentarão uma possível solução;
- Concepção de uma proposta de solução suportada numa análise crítica dos conceitos e abordagens estudadas;
- Avaliação da proposta através de um caso de estudo em que se inclui a implementação de um protótipo de infra-estrutura para desenvolvimento baseado em linhas de produto;
- Conclusões e recomendações sobre a adequação ao problema das abordagens estudadas e proposta de solução.

1.6 ORGANIZAÇÃO DA DISSERTAÇÃO

A dissertação está organizada em cinco secções:

- Introdução
  Enquadramento do problema e apresentação do trabalho
- Estado de arte
  Exposição dos principais conceitos e trabalho existente na área.
- Proposta de solução
  Apresentação de um modelo de desenvolvimento baseado em linhas de produtos
- Avaliação da proposta
  Elaboração de caso de estudo e implementação de protótipo para demonstração de conceito
- Conclusão e recomendações
  Apresentação de conclusões, limitações da proposta e recomendações

2 ESTADO DE ARTE

2.1 INDUSTRIALIZAÇÃO DO DESENVOLVIMENTO DE SOFTWARE

2.1.1 ENQUADRAMENTO

O *Standish Group*[1] produz anualmente o *CHAOS Report* com o objectivo de aferir qual o grau de sucesso/insucesso dos projectos de desenvolvimento de *software* em organizações espalhadas pelo mundo. Na Figura 2-i são apresentados alguns resultados sobre projectos executados nos últimos anos.

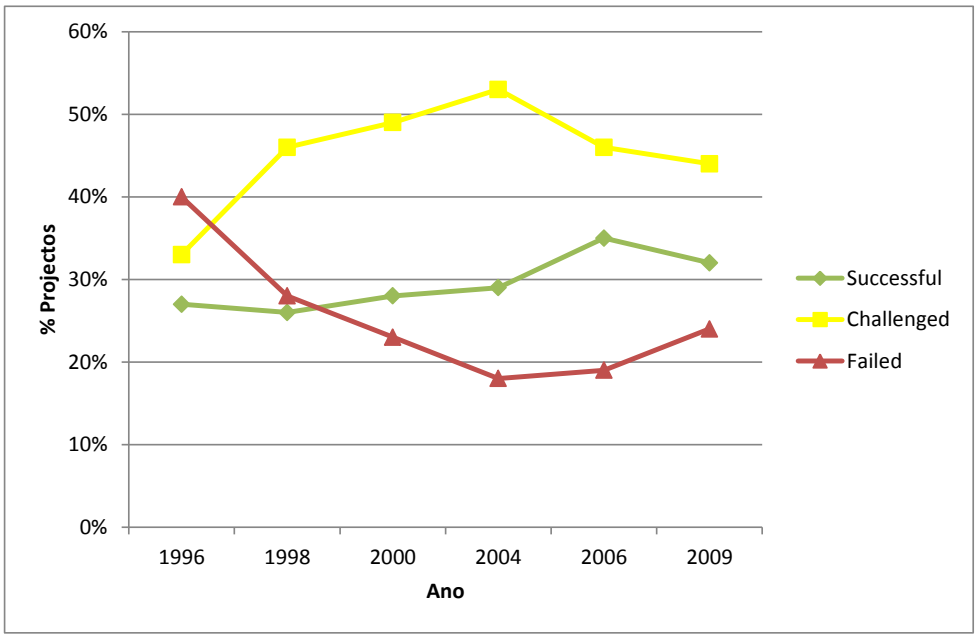

Figura 2-i - Evolução do sucesso dos projectos de Software. Adaptado de (Eveleens & Verhoef, 2010)

A categorização dos resultados é baseada em três vectores: tempo, orçamento e âmbito. Um projecto é considerado com sucesso (*Successful*) se tiver terminado dentro do prazo e orçamento previsto e com todas as funcionalidades esperadas. No lado oposto temos os projectos falhados (*Failed*), i.e. que foram cancelados em alguma altura do desenvolvimento. Por último, a categoria com resultados mais elevados (*Challenged*) que representa os projectos que terminam e ficam operacionais mas não no prazo ou orçamento esperado ou ainda se a implementação das funcionalidades for parcial.

Independentemente das críticas que se possam fazer ao método e critérios definidos no relatório (Eveleens & Verhoef, 2010), a verdade é que quem tem alguma experiência de participação em projectos de desenvolvimento sabe como é difícil cumprir os três vectores: tempo, orçamento e âmbito. A realidade mostra, que mesmo questionando os números, a taxa de sucesso estará longe do desejável. Reforçando esta ideia ressaltam do estudo as seguintes observações:

– Actualmente ainda existe uma baixa ocorrência de projectos com sucesso (apenas 32% em 2009);

[1] http://www.standishgroup.com

- Em 13 anos (de 1996 a 2009) o número de sucessos apenas cresceu 5%;
- De 2004 até aos dias de hoje, a percentagem de projectos falhados tem vindo a crescer.

Mas porque é tão difícil desenvolver sistemas no calendário, orçamento e âmbito definidos? Qual a razão de ganhos tão marginais no sucesso dos projectos apesar dos incontestáveis avanços tecnológicos e metodológicos?

Por um lado, parte da resposta a estas perguntas pode ser inferida directamente do contexto complexo em que o desenvolvimento desses sistemas ocorre, por outro, podemos identificar problemas na abordagem de construção que seguimos. É com base nestas duas perspectivas que sintetizamos na Tabela 2-i e na Tabela 2-ii as relações causa-impacto apresentadas por vários autores [e.g. (MELLOR, Scott, Uhl, & Weise, 2004), (Lenz & Wienands, 2006), (Pohl, Böckle, & Linden, 2005) ou (Greenfield, Short, Cook, & Kent, 2004)].

Conjuntura (causa)	Impacto nos sistemas de informação
As organizações estão expostas a uma economia global competitiva que as obriga a constantes mudanças.	O maior dinamismo resulta directamente no aumento do esforço de desenvolvimento e manutenção de SI.
O foco das organizações na satisfação das necessidades do cliente.	Os SI devem apresentar maiores níveis de qualidade e fiabilidade.
Uma organização não existe sozinha, interage directamente com os seus parceiros e clientes.	A capacidade de integração é hoje um requisito aplicável a (quase) todas as soluções aplicacionais actuais.
Passamos da era do software de produtividade para o software de automatização de processos de negócio.	A automatização é, em grande parte, uma responsabilidade (complexa) atribuída aos SI.

Tabela 2-i – Impacto do contexto complexo no desenvolvimento

Causa	Impacto nos sistemas de informação
Desenvolvimento dedicado, os projectos são estruturados para o desenvolvimento de um sistema, sem considerar a reutilização futura dos activos e conhecimentos adquiridos.	Baixa produtividade resultante da constante reinvenção de soluções.
O processo de desenvolvimento assenta, maioritariamente, num processo de construção artesanal.	Baixa produtividade resultante do (considerável) esforço empregue em tarefas repetitivas.
Baixo nível em que é definida a solução em comparação com o alto nível em que é especificado o problema.	Desfasamento entre a necessidade do cliente e a solução entregue.
Pouca participação do cliente no processo de desenvolvimento.	Desfasamento entre a necessidade do cliente e a solução entregue.
A documentação é desactualizada ou inexistente.	Dificuldade e custos da manutenção.

Tabela 2-ii – Impacto das ineficiências da abordagem tradicional de desenvolvimento

Da análise dos aspectos apresentados, salientamos as seguintes conclusões:

- Contextos, e por inerência, sistemas de informação cada vez mais complexos;
- Baixa produtividade no processo desenvolvimento (falta de reutilização, custos elevados);
- Pouca qualidade (desadequação da solução e dificuldade de manutenção).

Estes factores são, muito provavelmente, os maiores desafios que se colocam ao actual desenvolvimento de sistemas de informação. É sobre esta motivação que variadas vertentes de investigação se dedicam, analisando e detalhando o problema, fornecendo novas perspectivas sobre as suas causas, propondo soluções e novas abordagens ao desenvolvimento de SI.

2.1.2 INDUSTRIALIZAÇÃO DO DESENVOLVIMENTO DE SOFTWARE

Uma vertente da comunidade científica, e do próprio mercado TI, defende a industrialização como a via mais promissora para endereçar os problemas que actualmente se colocam ao desenvolvimento de *software*.

Mas o que significa realmente a industrialização do desenvolvimento de *software*? Tentando responder a esta questão, Greenfield refere que ninguém saberá até que este processo efectivamente aconteça mas, sugere que se tente antecipar algum cenário através da análise do processo de industrialização ocorrido noutras indústrias e da correspondente identificação de pontos mapeáveis ao desenvolvimento de SI (Greenfield, Short, Cook, & Kent, 2004). Assim, resumimos na Figura 2-ii os principais marcos no processo de industrialização associado ao início do século XX.

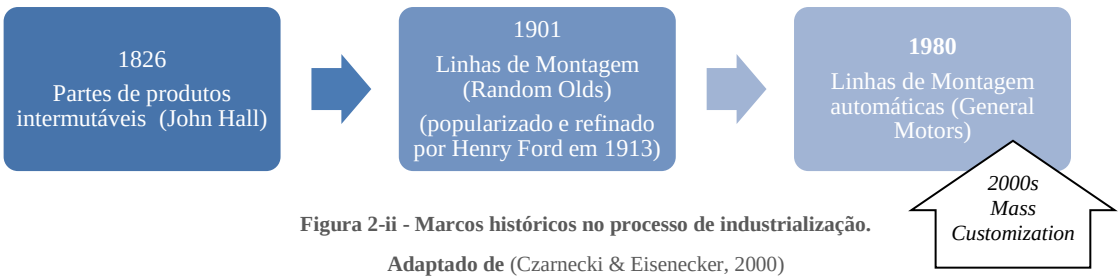

Figura 2-ii - Marcos históricos no processo de industrialização.

Adaptado de (Czarnecki & Eisenecker, 2000)

As partes de produtos intermutáveis e as linhas de montagem (automáticas) permitiram e estão na génese daquilo que chamamos produção em massa (*mass production*) surgido no início de século XX. Este tipo de produção foca a obtenção de economias de escala provenientes da construção de produtos estandardizados em grande quantidade. A verdade é que muito mudou desde então, e em pleno século XXI, a exigência do mercado é incomparavelmente maior e a satisfação das necessidades do cliente tornou-se uma preocupação central de qualquer indústria. Esta alteração tem alimentado uma nova filosofia de produção em que a concepção dos produtos e processos produtivos consideram opções de

personalização destinadas a aproximar as características dos produtos às necessidades dos potenciais clientes. Falamos do conceito de customização em massa (*mass customization*).

Esta experiência pode ser aproveitada e adaptada no processo de industrialização do desenvolvimento de *software*. Se é verdade que algumas práticas de produção em indústrias tradicionais não se aplicam (como a replicação, que no caso do software é simples), a verdade é que, olhando para os marcos históricos do processo de industrialização, podemos identificar pontos ajustáveis.

Partes de produtos intermutáveis

A modularização é uma tendência antiga na concepção de aplicações, desde o aparecimento de funções, passando pelos objectos e bibliotecas e mais recentemente os componentes e serviços, em que a segmentação é uma estratégia essencial para lidar com a complexidade envolvida.

Num contexto de sistemas de informação, as partes de produtos intermutáveis podem ser associadas a qualquer uma das técnicas de modularização referidas, e.g. funções, objectos bibliotecas, componentes ou serviços.

Linhas de montagem (automáticas)

O processo tradicional não aborda o desenvolvimento de sistemas numa perspectiva de montagem mas de construção. Um sistema é desenvolvido, na maioria das vezes de raiz, em função de um conjunto de requisitos.

Apesar de este ser o cenário actual, o processo de desenvolvimento industrializado poderá passar pelo equivalente a uma tradicional linha de montagem, ou seja, a produção de uma aplicação poderá ser realizada pela assemblagem, automática ou manual, de um conjunto de componentes reutilizáveis com recurso a um ambiente de desenvolvimento devidamente configurado para o efeito (Pohl, Böckle, & Linden, 2005).

Apesar da actual maturidade tecnológica permitir a transição de abordagem, a verdade é que a montagem de sistemas, utilizando activos reutilizáveis, é uma visão alternativa do desenvolvimento ainda com pouca expressão no panorama actual.

Produção em massa e customização em massa

Tal como a *mass production*, também o conceito de *mass customization*, deve ser adaptado ao contexto de produção de *software*. O primeiro fundamenta-se nas economias de escala obtidas pela produção em massa de produtos idênticos e o segundo na necessidade de adaptar/personalizar esses produtos às necessidades específicas de cada cliente. No caso do *software*, a *mass production* deve ter uma interpretação contextualizada pelo (desprezável) custo de replicação. Pelo contrário, a *mass customization* tem um impacto essencial no sucesso da aplicação das práticas industriais ao processo de desenvolvimento. Não é de todo interessante migrar para processos de desenvolvimento industrializados se isso implicar a perda da capacidade de personalização que é base para o alinhamento do SI com o contexto.

Há contudo visões alternativas à personalização "aberta" que tradicionalmente caracteriza os projectos actuais. Tal como na compra de um automóvel se escolhe uma cor de um catálogo (e não uma qualquer), no domínio dos sistemas de informação, a capacidade de personalização também poderá ser limitada. À primeira vista, considerando o largo espectro de contextos que os sistemas de informação estão sujeitos *versus* a limitação de personalização imposta pela estandardização, temos um impasse. Todavia, voltando à analogia dos automóveis, para assegurar a satisfação dos clientes descontentes com o catálogo de cores, muitas marcas disponibilizam um sistema de encomenda (excepcional) de viaturas de qualquer cor, aumentando naturalmente o preço cobrado e o tempo de entrega. Podemos idealizar um mecanismo semelhante no desenvolvimento de *software*.

Os pontos que acabámos de expor são os principais lugares comuns que podem ser estabelecidos entre um tradicional processo industrial e uma visão de um processo de desenvolvimento de sistemas de informação industrializado. Este enquadramento de perspectivas está na base da abordagem alvo deste trabalho, as *Software Produt Lines*.

2.2 SOFTWARE PRODUCT LINES – CONCEITOS E PRÁTICAS

2.2.1 DEFINIÇÃO

A reutilização é uma preocupação desde os primórdios do desenvolvimento de aplicações sendo o artigo de McIlroy (referido na introdução deste trabalho) várias vezes apontado como a primeira manifestação dessa posição (Jha & Brien, 2009). A Figura 2-iii apresenta os principais marcos na evolução da aplicação de técnicas de reutilização ao desenvolvimento.

Figura 2-iii – Evolução da aplicação de técnicas de reutilização. Adaptado de (Northrop, 2008)

As *Software Product Lines* são enquadradas como o próximo passo na evolução das abordagens pró reutilização (Northrop, 2008).

Uma definição para uma *Software Product Line* (SPL) é apresentada por Clements e Northrop, na obra *Software Product Lines: practices and patterns* (Clements & Northrop, 2001). Para os autores, uma SPL consiste num conjunto de sistemas (software) que compartilham características que satisfazem as necessidades específicas de um determinado segmento de mercado ou missão, e que são desenvolvidos a partir de um conjunto comum de activos (software e outros) de uma forma pré-definida.

Procurando uma clareza de conceito, Northrop complementa a definição especificando o que as SPL não são (Northrop, 2003):

- Desenvolvimento tradicional com reutilização;
- Reutilização fortuita e em pequena escala (e.g. bibliotecas, funções ou código);
- Desenvolvimento unicamente baseado em componentes ou orientado a serviços;
- Apenas versões diferentes do mesmo produto;
- Uma arquitectura configurável;
- Conjunto de *standards*.

A ideia fundamental assenta na mudança de paradigma de desenvolvimento de sistemas únicos "não reutilizáveis" para o fabrico de famílias de produtos, aproveitando as economias resultantes da reutilização sistemática, e de grande escala, de aspectos comuns que existem entre sistemas do mesmo domínio.

2.2.2 PRINCÍPIOS GERAIS

Na definição de SPL apresentada, é possível ler "necessidades específicas de um determinado segmento de mercado ou missão". Esta restrição de âmbito é necessária para limitar o **espaço do problema** e por consequência confinar o **espaço da solução** (Bragança, 2007).

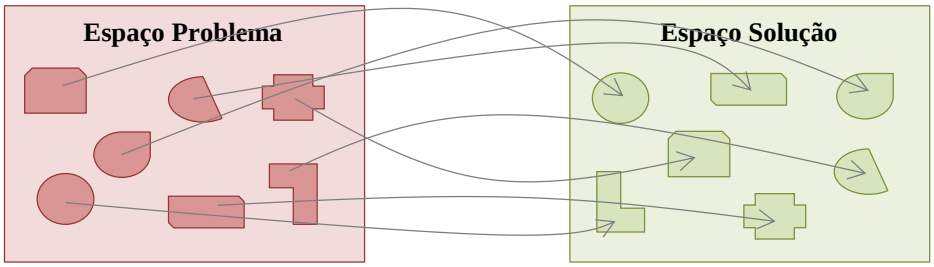

Figura 2-iv - Espaço de problema e espaço de solução. Adaptado de (Lenz & Wienands, 2006)

O objectivo central das SPL é a reutilização massiva, i.e. a construção de sistemas de informação com uma larga percentagem de trabalho reaproveitado. Utilizamos o termo "trabalho" pois quando nos referimos a reutilização em larga escala não podemos pensar em reutilizar apenas código, devemos estender este conceito para tudo o que está envolvido no desenvolvimento, como por exemplo, artefactos relacionados com análises de negócio, modelos de mercado, especificações de sistemas, casos de testes aplicacionais ou conhecimento e experiência. Em teoria, todos os artefactos envolvidos no desenvolvimento podem ser vistos como potenciais alvos de reutilização.

Para considerar reutilização de larga escala, não é realista pensar no universo como problema alvo. A diversidade é de tal forma elevada que a probabilidade de conseguir reutilizar algo seria mínima. Por isso mesmo, as SPL destinam-se a um domínio restrito de problemas, ou, usando a terminologia correcta, dirigem-se a um determinado espaço de problema.

Se alinharmos o raciocínio com a abordagem tradicional, percebemos que também fazemos algo semelhante quando iniciamos o processo com uma análise de negócio destinada à captação da informação necessária para a compreensão do problema que o sistema deve endereçar. Apesar da semelhança aparente, existe uma importante distinção. No desenvolvimento tradicional, a análise de negócio é focada apenas no subconjunto, muito restrito, das características do problema que dizem respeito ao sistema que estamos a construir. Numa óptica de linhas de produto, o âmbito desta análise não é suficiente pois pretende-se desenvolver um conjunto de aplicações - com base comum mas com características próprias - e não apenas uma única solução. Nesta abordagem, o estudo do domínio é delimitado, não pelo contexto de construção de uma aplicação, mas pelas características definidas para os membros da família de produtos, i.e. pelos limites do domínio alvo.

Na Figura 2-i vemos que, delimitando um problema/domínio, conseguimos restringir um espaço de solução e definir as relações entre eles (Lenz & Wienands, 2006). Na prática, falamos de um mundo em que existem conceitos de negócio que expressam os problemas com um correspondente universo técnico/tecnológico onde reside a solução.

Esta simples conceptualização está na base de argumentação de uma das técnicas que as linhas de produto fazem extensiva utilização: o desenvolvimento baseado em modelos. Fazendo a ponte com definição de espaços de problema e solução que acabámos de expor, consideremos que conseguimos definir um mecanismo que nos permite expressar com a

riqueza necessária e com um alto grau de formalidade, i.e. com o rigor necessário para ser interpretável por uma máquina, as características dos problemas que o domínio escolhido abrange. Se assim for, e se assumirmos que, para cada característica do problema, conhecemos a respectiva solução, podemos pensar em automatizações de desenvolvimento em função da definição do problema, i.e. na geração automática de (partes) de aplicações. O mecanismo como expressamos formalmente estas especificações é o desenvolvimento de linguagens e modelos específicos do domínio que estamos a considerar. Este tema será aprofundado na secção 2.3.3.

Com base no exposto, como seria expectável, o estudo do domínio toma um lugar central nas SPL. Esta análise é considerada num processo mais abrangente denominado por **engenharia de domínio** ou desenvolvimento para reutilização (*development for reuse)*.

Os objectivos centrais da engenharia de domínio passam pela definição da linha de produtos (LP), i.e. pela determinação das características dos membros da família com base no domínio definido, e pela definição e implementação de todos os elementos reutilizáveis necessários para a construção desses mesmos sistemas. Dito de forma informal, a engenharia de domínio determina o domínio de actuação, os sistemas que a LP deve ser capaz de produzir e ainda toda a infra-estrutura e componentes necessários para a sua operacionalização.

O momento chave que estabelece formalmente o contexto para o qual a linha de produtos deve estar preparada responder e quais as características dos membros da famílias que estão consideradas é a fase de definição de **âmbito da LP**.

Ao contrário de muitas indústrias, a produção de aplicações informáticas "iguais" é simples. A replicação é, neste contexto, trivial. Ao mesmo tempo, a construção de famílias de produtos sem qualquer capacidade de personalização dos seus membros, i.e. aplicações iguais, seria inútil. Desse modo, aquando da definição do âmbito da SPL, é necessário definir quais as características comuns a todos os membros mas também que características poderão variar. Esta necessidade introduz outra responsabilidade importante da engenharia de domínio a **gestão da variabilidade**.

Ao invés de atribuir a responsabilidade da gestão dos pontos comuns e variáveis ao desenvolvimento, deixando que para cada produto se defina o que pode ou não variar, a abordagem SPL defende que esta elasticidade deve ser estruturada numa óptica de família e não de produto isolado e que por isso deve ser considerada atempadamente.

A especificação do domínio que suporta a SPL não é o destino mas sim o caminho, i.e., a instanciação do processo e da infra-estrutura implementada é que resulta no produto desejado (Pohl, Böckle, & Linden, 2005). Para que tal aconteça é necessário um processo específico, com ciclo de vida com algum grau de independência da engenharia de domínio, que agregue e faça a gestão das actividades necessárias para que o produto final corresponda ao desejado – **engenharia de aplicação** ou desenvolvimento com reutilização (*development with reuse*).

É com base nos dois processos distintos - engenharia de domínio e engenharia de aplicação – destinados à construção de famílias de sistemas, que assenta a mudança de paradigma de desenvolvimento promovido pelas SPL.

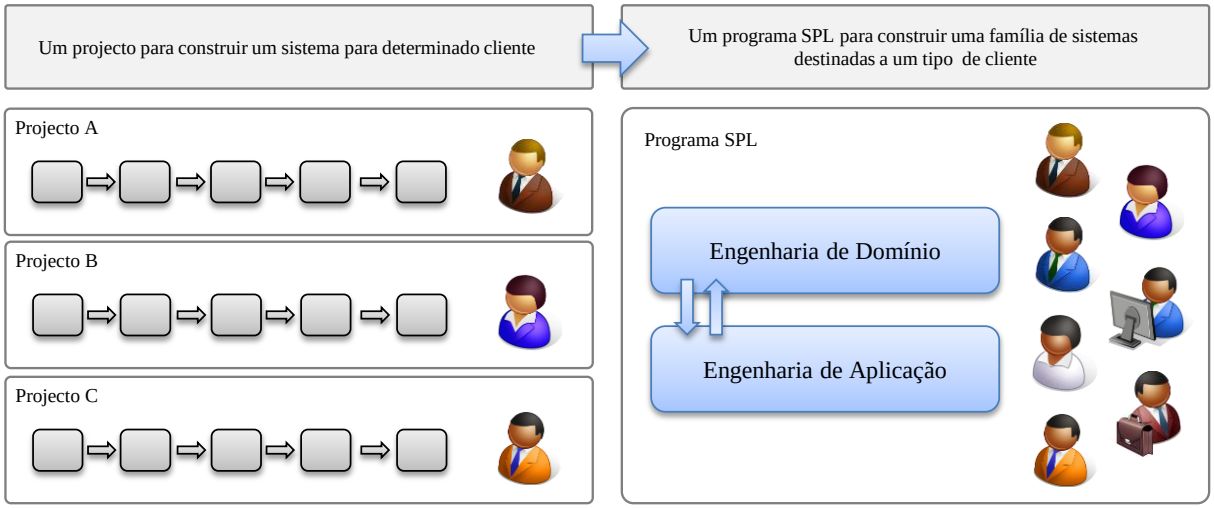

Figura 2-v – Visão SPL

Aparte desta relação engenharia de domínio / engenharia de aplicação, não existe um modelo de desenvolvimento universal que especifique as fases de cada processo de engenharia. Todavia, existem variadas propostas de modelos de desenvolvimento de alto nível baseados em conhecimento adquirido em implementações de programas SPL.

Não podemos alhearmo-nos do contexto das SPL. Falamos de uma filosofia de desenvolvimento e não de um conjunto pré-definido de passos para construir aplicações. Não é, aliás, a única alternativa a colocar-se neste nível de abstracção. Algumas semelhanças de posição podem ser estabelecidas com outros tipos de abordagens como o *Component-Based Development* ou o *Rapid Application Development* (RAD) (Greenfield, Short, Cook, & Kent, 2004).

Na próxima secção apresentamos algumas propostas para apoio ao desenvolvimento orientado a famílias de sistemas. Todas se baseiam em experiências provenientes de implementações reais mas, ainda que se apresentem num formato independente de domínio, nenhuma delas se destina a fornecer um plano prescrito de passos para a execução de programas idênticos.

2.2.3 FRAMEWORKS SPL

Nesta secção apresentamos algumas *frameworks* que se baseiam nos princípios genéricos do desenvolvimento baseado em linhas de produtos. Antes de mais, é importante esclarecer o que entendemos por *framework* neste contexto. Uma *framework* SPL é um conjunto de informação organizado sobre o desenvolvimento baseado em linhas de produto. Não contempla, obrigatoriamente, uma sequência de passos pré-determinados para execução de programas SPL mas agrega conhecimento importante para a sua implementação.

A engenharia de domínio é o núcleo de um programa SPL. Todavia, o enquadramento SPL não é uma condição *sine qua non* para a sua adopção. Czarnecki e Bragança (Czarnecki & Eisenecker, 2000) (Bragança, 2007) identificam várias metodologias que não condicionam a sua aplicação a contextos orientados a famílias de sistemas:

- DRACO
- Organization Domain Modeling (ODM)
- Feature-Oriented Domain Analysis (FODA)
- Context Analysis
- Capture
- Domain Analysis and Reuse Environment (DARE)
- Domain-Specific Software Architecture (DSSA)
- Reuse-Driven Software Engineering Business (RSEB)
- Algebraic Approach

Cada uma destas práticas tem as suas particularidades mas todas têm um ponto em comum: focam a criação de um ambiente de reutilização estruturado. No que respeita à importância dada à reutilização, a identificação com as SPL é total, por isso, muitas destas metodologias serviram de inspiração para modelos de processo incluídos em programas SPL.

O âmbito deste trabalho limita a abrangência da nossa análise. Em resultado, cobriremos exclusivamente *frameworks* SPL, ou seja, propostas que cubram o ciclo de vida completo de um programa orientado à construção de famílias de produtos (engenharia de domínio e engenharia de aplicação). Mesmo optando por esta restrição, são inúmeras as alternativas. Assim, seleccionámos as propostas que considerámos mais relevantes para o modelo que propomos no próximo capítulo. Nessa circunstância, restringimos a nossa exposição a duas *frameworks*: a Framework for Software Product Line Practice (FSPLP) e as Software Factories.

2.2.4 FRAMEWORK FOR SOFTWARE PRODUCT LINE PRACTICE

Em 1999, o Software Engineering Institute (SEI) da Universidade Carnegie Mellon publicou a primeira versão da Framework que propõem para o desenvolvimento de SPL. Actualmente, a *Framework for Software Product Line Practice* (FSPLP) está na versão 5.0 (Institute, 2010a). Esta proposta é reconhecida pela comunidade como uma das mais completas, em particular porque considera o alinhamento entre o desenvolvimento da SPL e a estratégia da organização.

A FSPLP inspira-se em organizações que implementaram, com ou sem sucesso, estratégias de desenvolvimento orientado a linhas de produtos, tentando obter dessa amostra os ensinamentos que permitiram ou inviabilizaram resultados positivos. Boeing, Bosh, Ericsson, Hewlett Packard, Nokia, Philips e Toshiba são alguns exemplos dessas organizações (Institute, 2010b).

O canal privilegiado de publicação e actualização da *framework* é o sítio Web http://www.sei.cmu.edu/productlines/. De seguida, apresentamos os princípios gerais descritos nessa documentação.

2.2.4.1 Fundamentos

Embora as organizações variem bastante, de acordo com o domínio em que actuam, existem actividades e práticas comuns. A FSPLP estrutura esses aspectos em dois grupos distintos: as actividades essenciais e as áreas de prática.

As **actividades essenciais** correspondem aos macros processos que envolvem uma iniciativa SPL. Existem 3 actividades essenciais relacionadas entre si:

- O desenvolvimento dos activos core
- O desenvolvimento dos produtos
- A gestão técnica e organizacional do projecto

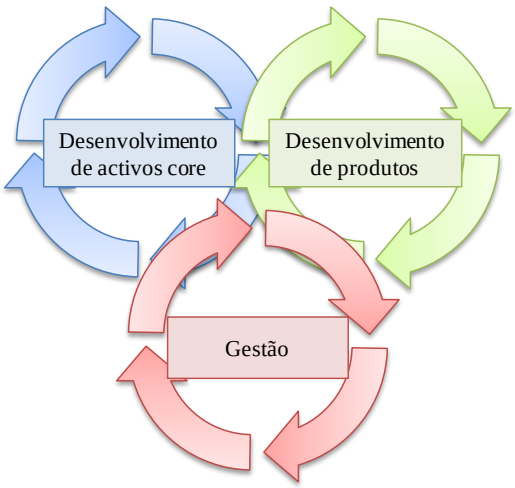

Figura 2-vi – Actividades essenciais FSPLP

As primeiras duas actividades têm um mapeamento directo com o que anteriormente apelidámos de engenharia de domínio e aplicação (respectivamente). A consideração da gestão como actividade essencial é uma particularidade interessante deste proposta e surge da constatação da importância de relacionar e integrar as restantes actividades, por exemplo, para gerir a alocação de recursos a cada processo ou para coordenar feedback entre as equipas de cada processo.

Em vez de definir um modelo de processo para a execução de cada actividade essencial, a FSPLP define áreas de prática. De acordo com os autores, uma área de prática (AP) é "uma área de trabalho, ou conjunto de actividades, que uma organização deve dominar para garantir a execução correcta das actividades essenciais"

A FSPLP define 29 áreas de prática, no entanto adverte para que a divisão é meramente indicativa e a redundância em algumas é inevitável pois o conhecimento não é "separável". Para facilitar a identificação do âmbito, as áreas estão organizadas em 3 grupos:

- Engenharia de Software: concentra as áreas fundamentais para aplicar a tecnologia apropriada para criar e evoluir os activos core e produtos;
- Gestão Técnica: agrega as áreas necessárias para gerir a criação e evolução dos activos core e produtos;
- Gestão Organizacional: agrupa as áreas essenciais para orquestrar o esforço de construção da SPL.

2.2.4.2 Actividades essenciais

2.2.4.2.1 Desenvolvimento dos activos core

Este ciclo de desenvolvimento visa a definição e implementação dos activos reutilizáveis e artefactos necessários para realizar o processo produtivo. Tal como apresentado na Figura 2-vii, a *framework* aponta quatro formas de condicionantes à execução da actividade e três importantes tipos de resultados.

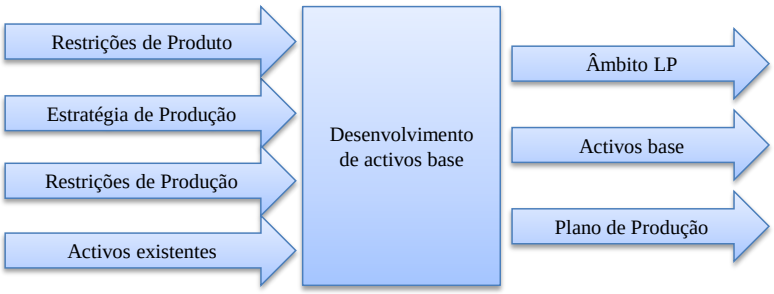

Figura 2-vii - Condicionantes e resultados do Desenvolvimento de activos core

Como não é realista estruturar o desenvolvimento de uma linha de produtos num vácuo, a *framework* define à partida quatro aspectos que contextualizam e devem ser considerados pela actividade de desenvolvimento de activos core:

- **As restrições do processo produtivo**
  A produção deve estar enquadrada com as necessidades do mercado e características internas da organização. Exemplos: Definição de *time-to-market* máximo ou a determinação de *standards* a aplicar ao processo de desenvolvimento (e.g. CMMI).
- **As restrições do produto**
  Os produtos que constituirão a linha de produtos devem respeitar um conjunto de requisitos (restrições aplicáveis a todos os produtos a produzir). Exemplos: Definição das restrições gerais da linha de produção ou determinação dos padrões gerais de qualidade a aplicar aos produtos gerados.
- **A estratégia de produção**
  A estratégia de produção deve reflectir os aspectos tácticos e técnicos que contextualizam a construção da SPL, em particular deve garantir que as restrições de produção e produto são consideradas. Exemplos: Definições da abordagem de produção dos activos reutilizáveis - se são construídos de raiz, se são extraídos e adaptados a partir de produtos existentes ou ainda se a opção recai sobre uma estratégia mista que combine construção de raiz com adaptação de soluções existentes.
- **Activos já existentes**
  A experiência que a organização ganhou ao desenvolver produtos pela via tradicional não deve ser descartada na adopção de uma abordagem SPL. Pelo contrário, esses elementos são aspectos importantes a considerar durante a construção dos activos reutilizáveis pois várias economias e eficiências podem ser obtidas. A organização deve avaliar o seu passado e responder a perguntas como: *Existe algum padrão de desenho, com comprovado sucesso no domínio, que possa ser um bom candidato a fazer parte da SPL? Existem algoritmos, bibliotecas ou ferramentas que a organização tenha aplicado com sucesso? E para evitar erros passados, existe o mesmo tipo de tentativa sem sucesso?*

O desenvolvimento da actividade, devidamente contextualizada pelos aspectos que acabámos de enumerar, envolverá várias áreas de prática. O resultado da actividade deverá materializar-se de três formas distintas:

- **O âmbito da SPL**
  O âmbito de uma SPL consiste na definição das características dos produtos que estão incluídos na linha de produtos. Esta descrição deve estabelecer claramente as potencialidades e limitações da SPL (indicação dos pontos comuns entre elementos da família, dos aspectos variáveis disponíveis e ainda dos pontos de extensão que permitem expandir as funcionalidades existentes)

- **Activos base**
  Todos os elementos necessários à produção. Nesta categoria incluem-se componentes de software (desenvolvidos internamente ou comprados), arquitecturas, padrões de desenho, ferramentas, geradores, planos de testes, especificações de requisitos, ou artefactos de gestão como planeamentos ou orçamentos. Cada activo deve estabelecer um plano de utilização que assegure a sua correcta aplicação.
- **Plano de produção**
  Estabelece como os produtos são construídos a partir dos activos base. Engloba a definição do processo produtivo propriamente dito, isto é, define a articulação entre os processos associados a cada activo base, bem como a descrição detalhada da gestão e execução do projecto.

2.2.4.2.2 Desenvolvimento de produtos

Esta actividade depende directamente dos resultados da actividade de desenvolvimento de activos base, nomeadamente do âmbito da linha de produtos, dos activos reutilizáveis e do plano de produção.

Como referimos anteriormente, cada activo reutilizável deve ter associado um rigoroso plano que estipule os parâmetros da sua utilização. É este o método que a FSPLP define para que, em tempo de construção de aplicação, se assegure que a forma de reutilização estabelecida no desenvolvimento do activo é cumprida. Estes planos são depois integrados de acordo com o plano de produção principal, o plano de produção da linha de montagem.

O plano de produção principal é genérico. Fornece um conjunto de passos para construir uma aplicação que pertença à família de sistemas considerado. No entanto, o grau de variação entre membros da família pode ser elevado o que resulta em grandes variações nos planos de produção dos produtos respectivos. Para lidar eficazmente com a configuração do plano de produção e garantir que o produto corresponde às expectativas, a FSPLP propõe que se elabore uma caracterização para cada produto (*Product Description*). Para além desta função de apoio à configuração do plano de produção, esta caracterização poderá também ser utilizada para verificar eventuais desalinhamentos com o âmbito da linha de produtos (e.g. validar se há alguma característica do produto que não esteja considerada).

Outro importante aspecto associado às caracterizações dos produtos é a importante contribuição que trazem para a manutenção da linha de produtos. A linha de produtos deve evoluir, isto é, o conjunto de elementos base deve ser enriquecido ao longo do tempo para permitir que os produtos que são construídos através dela evoluam também. As características específicas de cada produto são importantes motivações para a construção de novos elementos base, bem como o feedback dos processos existentes é crucial para o seu melhoramento.

2.2.4.2.3 Gestão

Nos casos de estudo em que se baseou para definir a FSPLP, o SEI identificou uma importante relação entre a saudável coordenação e gestão das actividades essenciais e o

sucesso dos programas SPL. À actividade de gestão cabe a responsabilidade de criar condições para que essa articulação decorra da melhor forma.

Podemos distinguir dois tipos de gestão: uma gestão de projecto (mais técnica) e uma gestão organizacional. A primeira assegura que a operacionalização dos processos definidos ocorre da melhor forma, a segunda certifica o alinhamento da SPL com a estratégia global da organização.

Esta actividade é, muito provavelmente, a área cujo contributo da *framework* é mais relevante. Neste campo não existe proposta que apresente semelhante grau de detalhe. A FSPLP define 20 áreas de prática (ver próximo ponto) com a estruturação de um extenso e detalhado conhecimento sobre aspectos de gestão (técnico e organizacional) que a maior parte das abordagens define superficialmente ou nem sequer aborda. Se considerarmos que se trata de um programa estrutural, de elevado investimento inicial, com mudanças consideráveis no modelo de desenvolvimento e com um horizonte temporal alargado, facilmente aceitamos a importância de incorporarmos, em qualquer programa SPL, um processo consistente de gestão.

2.2.4.3 Áreas de prática

Uma área de prática é apresentada na *framework* como uma unidade de trabalho ou campo de conhecimento. Pretende representar um campo de actuação (essencial) para qual a organização deve estar preparada para responder de forma adequada. Nesse sentido, representam a operacionalização das actividades principais e por isso mesmo podem acompanhar, de forma mais próxima e realista, os trabalhos a realizar. A Tabela 2-iii resume a forma como estão organizadas.

Engenharia de Software	Gestão Técnica	Gestão Organizacional
Definição da arquitectura	Gestão de configurações	Construir um caso de negócio
Avaliação da arquitectura	Análise Desenvolver, Comprar, Extrair, Subcontratar	Gestão do relacionamento com o cliente
Desenvolvimento de componentes	Medição e Monitorização	Desenvolver uma estratégia de aquisição
Exploração de activos existentes	Disciplina de processo	Financiamento
Engenharia de requisitos	Definição do âmbito	Lançamento e institucionalização
Integração de componentes de software	Planeamento técnico	Análise de mercado
Testes	Gestão do risco (técnico)	Definição da operacionalização
Compreender domínios relevantes	Suporte de ferramentas	Planeamento organizacional
Utilização de software disponível externamente		Gestão do risco (perspectiva organizacional)
		Estruturação da organização
		Previsão de alterações tecnológicas
		Treino

Tabela 2-iii – Áreas de Prática FSPLP

Para cada área de prática, são abordados aspectos como as particularidades da orientação a linhas de produtos em comparação com a abordagem tradicional, os maiores riscos identificados, ferramentas e técnicas existentes, metodologias ou até obras de referência para cada assunto. A informação disponível para cada área torna impossível a sua apresentação nesta análise. Contudo, considerando o grande contributo para o modelo que propomos no próximo capítulo, apresentamos no Anexo – Áreas de Prática FSPLP um breve resumo de cada área com o objectivo de fornecer ao leitor um panorama geral do seu objecto.

2.2.5 SOFTWARE FACTORIES

O termo Software Factory não é novo nem foi sempre usado com o mesmo significado (Cusumano, 1989). Recentemente a designação foi adoptada por Jack Greenfield e Keith Short na obra "Software Factories: Assembling applications with patterns, models, frameworks and Tools" para intitular uma *framework* de desenvolvimento baseado em linhas de produtos.

Uma Software Factory (SF) é apresentada pelos autores como uma SPL que configura ferramentas, processos e conteúdo usando um SF *Template*, baseado num SF *Schema*, para automatizar o desenvolvimento e manutenção de produtos através da adaptação, montagem e configuração de componentes (Greenfield, Short, Cook, & Kent, 2004).

A definição identifica, desde logo, vários aspectos essenciais das SF:

- É uma SPL com dois artefactos essenciais: SF *Schema* e SF *Template*;
- Assenta em automatização de desenvolvimento;
- Baseia-se em componentes e linhas de montagem;
- Recorre a ferramentas, processos e activos desenvolvidos/configurados especificamente para o contexto em causa (específicos de domínio).

A *framework* não pretende prescrever uma metodologia de desenvolvimento mas antes apresentar um conjunto de princípios e fluxos de trabalho de alto nível para serem instanciados de acordo com o contexto em causa, e.g. de acordo com o domínio da LP, com as tecnologias adoptadas, com os activos existentes ou com a estrutura da organização.

2.2.5.1 Modelo

As Software Factories dividem o desenvolvimento em dois processos distintos: o desenvolvimento da infra-estrutura de produção (software factory) e o desenvolvimento de produtos.

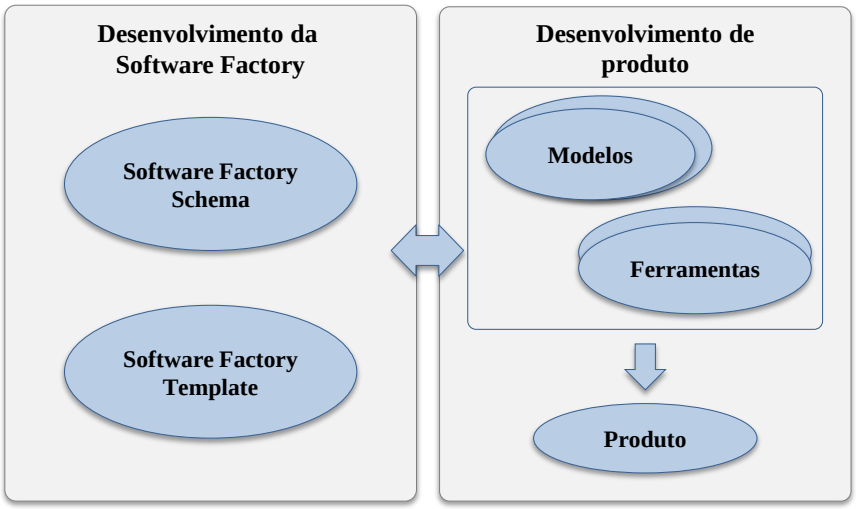

Figura 2-viii – Modelo alto-nível das Software Factories. Adaptado de (Lenz & Wienands, 2006)

Conforme apresentado na Figura 2-viii, o processo de desenvolvimento da Software Factory assenta em dois elementos essenciais: o SF *Schema* e o SF *Template*.

O SF Schema é um modelo interpretável por humanos e máquinas que descreve os activos, e respectivos fluxos de trabalho, necessários para construir uma família de produtos (Greenfield, Short, Cook, & Kent, 2004).

O SF Schema descreve elementos, não envolve a concretização dessa descrição, i.e. não implementa os elementos especificados. Esta materialização pode envolver a implementação de linguagens específicas de domínio, a criação de extensões para IDE, a construção de arquitecturas aplicacionais, bem como o *packaging* e *deployment* desses elementos num ambiente integrado de desenvolvimento de aplicações. SF Template é a designação atribuída a este conjunto de elementos, implementados de acordo com o SF Schema.

Tanto o SF Schema como o SF Template serão utilizados no processo de desenvolvimento do produto. Colocada de forma abstracta, a ideia é bastante simples: os elementos disponibilizados pelo SF Template, se devidamente acoplados como definido no SF Schema, constituirão a solução. Essa "montagem" será feita com base em modelos e ferramentas especializadas.

Analisemos com maior detalhe estes elementos e a articulação entre os dois processos.

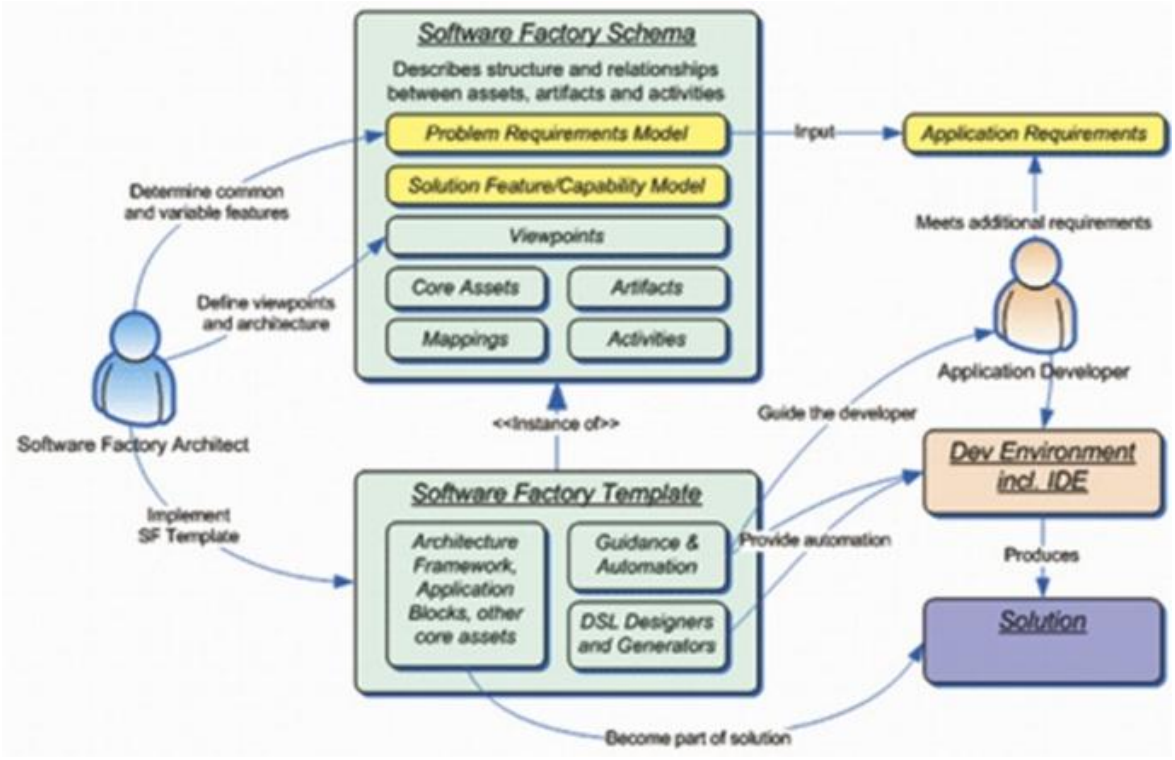

Figura 2-ix – SF Schema e SF Template. Retirado de (Lenz & Wienands, 2006)

A Figura 2-ix apresenta uma visão mais detalhada da *framework*. Comecemos por analisar as responsabilidades atribuídas ao processo de construção da Software Factory, na figura, representado pelo *Software Factory Architect*:

- Determinar as características comuns e variáveis
  Na construção da LP, o primeiro passo é a definição do seu âmbito, o que corresponde à definição dos aspectos comuns e variáveis dos produtos. Esta informação é registada sob duas perspectivas – problema e solução – e utilizando técnicas de modelação com níveis de abstracção diferentes – características (aspectos funcionais de alto nível) e requisitos.
- Definir arquitectura e *viewpoints*
  A arquitectura é a infra-estrutura aplicacional que servirá para acomodar a variabilidade definida para a LP. Tipicamente, a complexidade associada a estrutura desse tipo implica tomar várias perspectivas (*viewpoints*) na sua descrição.
- Implementar o SF Template
  O SF Template implementa o SF Schema pelo que os seus elementos dependem das descrições que aí estiverem definidas. Ainda que exista essa dependência do SF Schema, a figura identifica três tipos de elementos: activos core (*architecture framework*, *Application blocks*, etc), editores de Domain-Specific Languages (DSL) e geradores, e mecanismos de *Guidance* e Automatização.

Esta segmentação do SF Template introduz uma das principais contribuições desta framework, uma visão integrada de tecnologia que numa perspectiva muito concreta diz respeito à articulação de três tipos de técnicas/tecnologias:

- Frameworks aplicacionais e componentes (na figura referidos como *Architecture frameworks* e *Application blocks*): num contexto Software Factories, são utilizados para criar a plataforma base da linha de produtos e a variabilidade definida;
- Domain-Specific Languages (DSL): consistem em linguagens criadas para modelação das aplicações (e correspondentes ferramentas de manipulação – Designers) com base na framework aplicacional e componentes existentes;
- Técnicas de automatização (na figura referidos *Generators* e *Guidance and Automation*): incluem geradores que permitem a integração DSL-frameworks aplicacionais e extensões/adaptações de ferramentas de desenvolvimento com a vista ao apoio e automatização de passos da construção da aplicação.

A aplicabilidade destas técnicas não se restringe às SPL nem tão pouco às Software Factories, sendo que vários processos de desenvolvimento fazem uso das suas potencialidades. Esta independência de contexto levou-nos a separar a sua análise da apresentação das *frameworks*. A forma lacónica com que acabámos de apresentar as técnicas/tecnologias deverá servir apenas para contextualização do leitor pois os detalhes serão encontrados na próxima secção.

Para culminar a análise do modelo das Software Factories, tomemos agora a perspectiva da construção de produto representada na Figura 2-ix pelo *Application Developer*.

- O primeiro ponto a notar é que será este elemento que definirá os requisitos da aplicação. Estes requisitos poderão, ou não, estar previamente considerados na linha de produtos. Se estiverem, não haverá desenvolvimento adicional e os recursos disponibilizados pela LP serão suficientes para produzir a aplicação, se não existirem será necessário desenvolvê-los.
- O segundo ponto a assinalar é que o trabalho do Application Developer será realizado através de um ambiente de desenvolvimento adaptado com base nos mecanismos de *guidance* e automatização, editores DSL e geradores disponibilizados pelo Software Template.
- O terceiro e último aspecto que gostaríamos de realçar é o facto da aplicação gerada ser constituída pela *architecture framework* (base aplicacional) e outros activos core disponibilizados pelo SF Template.

Não há nada nos princípios das Software Factories que imponha que iniciativas que nela se baseiem tenham de optar por determinado contexto tecnológico. Todavia, há uma forte compatibilidade entre os elementos definidos pela *framework* e as ferramentas disponibilizadas pela Microsoft[2] (Greenfield, Short, Studio, Tools, et al., 2004).

[2] http://www.microsoft.com

2.3 SOFTWARE PRODUCT LINES – TÉCNICAS E TECNOLOGIAS

2.3.1 ENQUADRAMENTO

A secção anterior focou-se no aspecto conceptual do desenvolvimento baseado em famílias de produtos. Os próximos pontos serão dedicados à apresentação das técnicas e tecnologias mais relevantes e frequentes na operacionalização – com foco particular na implementação dos sistemas - desse tipo de abordagem.

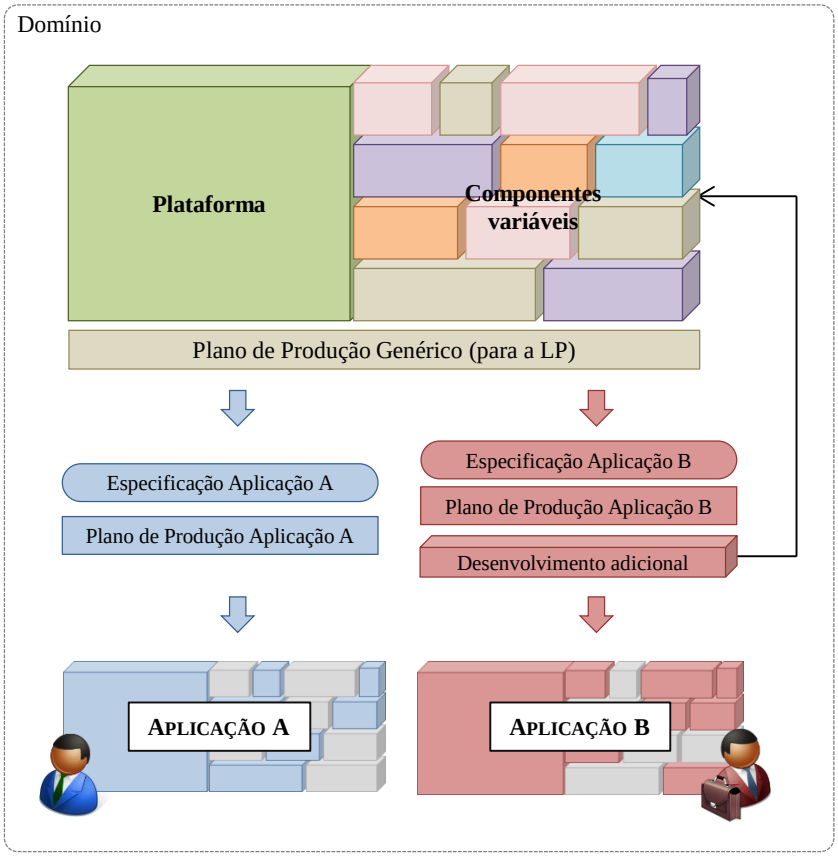

Figura 2-x – Cenário de produção baseado em linhas de produtos

A Figura 2-x apresenta um cenário típico de SPL em funcionamento. Sucintamente, trata-se de uma linha de produtos que é instanciada para dois clientes de acordo com as suas necessidades específicas (enquadráveis no domínio da LP). Cada aplicação é construída com base no plano de produção genérico da LP, devidamente configurado de acordo com a especificação respectiva. As características da aplicação A são completamente cobertas pela linha de produtos ao contrário da Aplicação B em que a especificidade do problema resultou na necessidade adicional de desenvolver um novo componente.

O exemplo escolhido, ainda que muito simples, contém a informação necessária para ser utilizado como guia na introdução das técnicas e tecnologias que se segue.

2.3.2 VARIABILIDADE EM LINHAS DE PRODUTO

2.3.2.1 Características (*features*)

Um simples ficheiro de configuração - que utilizamos para alterar o comportamento de uma aplicação de acordo com as particularidades do contexto - é um exemplo de variabilidade no desenvolvimento tradicional. Estes ficheiros incluem um, ou vários aspectos, que foram isolados para poderem ser facilmente alterados porque, antecipadamente, foram identificados como variáveis.

Numa linha de produtos, a variabilidade toma proporções muito maiores. É com base na correcta gestão da variabilidade que se vão produzir sistemas adequados às necessidades de um conjunto potencial de clientes, através de um processo de desenvolvimento centrado em reutilização sistemática. Graças a essa relevância, a identificação dos pontos comuns e variáveis entre os membros da família de produtos é um dos passos mais importantes na engenharia de domínio. Neste campo, uma técnica bastante utilizada é a modelação de características (Sochos, Philippow, & Riebisch, 2004).

Antes de entrar na modelação de características importa clarificar o que é uma característica numa abordagem de linhas de produtos. Bosch (Bosch, 2000) apresenta a seguinte definição: *uma característica é uma unidade lógica de comportamento que é especificada por um conjunto de requisitos (funcionais ou não funcionais).* Desta definição, podemos desde logo inferir uma relação entre característica e requisito. Adicionemos à análise a definição de Requisito no IEEE Standard 610.12 (IEEE, 1990): *um requisito é uma condição ou capacidade que deve garantida por um sistema ou componente de sistema*.

Ambos reflectem aspectos do sistema, a principal distinção está no nível de abstracção em que se colocam. Um requisito, tal como a definição do IEEE assume, tem uma directa relação com o sistema – traduz, por exemplo, uma funcionalidade da aplicação -, uma característica também expõe um aspecto do sistema mas num nível de abstracção completamente independente da sua implementação que permita a sua correcta interpretação por todos os *stakeholders*. Por outras palavras, características são uma forma de abstrair requisitos (Svahnberg, 2000).

Figura 2-xi - Característica vs Requisito

A diferença de nível de detalhe entre característica e requisito, traduz-se na relação "muitos para muitos" apresentada na Figura 2-xi. Uma característica pode, e por norma é, mapeada em vários requisitos e um requisito geralmente traduz a materialização de várias características.

Segundo a Czarnecki (Czarnecki & Eisenecker, 2000) analisar a variação da linha de produtos numa camada de abstracção adicional, independente de implementação, ajuda a evitar dois sérios problemas na definição da LP:

- Falhar a inclusão de características relevantes e pontos de variação importantes:
- Incluir aspectos pouco relevantes, nunca ou raramente utilizados, contribuindo exclusivamente para o peso do desenvolvimento e manutenção.

Existem várias propostas de tipificações para características [(Czarnecki & Eisenecker, 2000), (Svahnberg, 2000) ou (Ye & H. Liu, 2005)]. Embora existam algumas variantes, três grandes categorias estão presentes em todas elas:

- Características obrigatórias: características que devem ser incluídas em todos os produtos da família;
- Características variantes: conjunto de características alternativas com obrigação de inclusão no produto de pelo menos uma.
- Características opcionais: características cuja inclusão no produto é facultativa;

Existem ainda alguns autores [e.g. (Tirilä, 2003)] que englobam nesta tipificação as características específicas de cada produto.

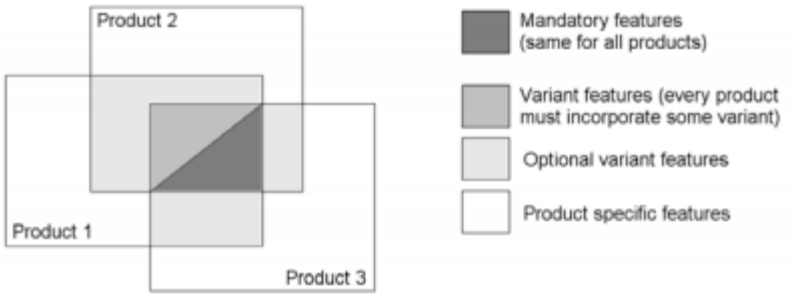

Figura 2-xii – Tipos de características. Retirado de (Tirilä, 2003)

Outro aspecto importante na definição da variabilidade é a identificação do momento em que a decisão de variação, i.e. a decisão de incluir ou não a característica num produto, deve ser tomada (*binding time*) (Chakravarthy, Regehr, & Eide, 2008). No que à implementação diz respeito, deve ser estabelecida a seguinte diferenciação (Sven & Saake, 2011):

- *Static binding*: incorporação de características em tempo de compilação ou em fase de pré-processamento do código;
- *Dynamic binding*: incorporação de características na pré-execução ou execução do programa (runtime)

Ambas opções têm vantagens e desvantagens. O método dinâmico é mais flexível (porque evita recompilação de código) mas introduz uma maior penalização na performance da aplicação. Como a variação em LP atinge configurações bastante complexas, existem situações em que não é possível antecipar a necessidade de incluir características para

momentos anteriores à execução da aplicação. Nestas circunstâncias, o método dinâmico é o único aplicável.

2.3.2.1.1 Modelação de características

Modelação de características é a actividade de identificar características de produtos num determinado domínio e organizá-las num modelo de características (feature model) (K. Lee, Kang, & J. Lee, 2002). Um modelo de características é constituído por diagramas de características e informação adicional (e.g. detalhe descritivo, *binding time*, ou restrições sobre as características).

Um diagrama de características é um modelo gráfico de uma hierarquia de características que captura sua estruturação e relações conceptuais (K. Lee, Kang, & J. Lee, 2002). Embora seja possível modelar características através de UML (e.g. (Vrani & Snirc, 2006)) esta não é a forma mais usual de o fazer pois, por um lado, não têm suporte nativo, por outro, está demasiado preso a conceitos técnicos como herança ou composição.

Hoje, a notação mais aceite para modelação de características é a Cardinality-Based Feature Modeling de Czarnecki-Eisenecker (Ípka, 2005). Esta linguagem muito simples é uma extensão da notação utilizada na *framework* de engenharia de domínio Feature-Oriented Domain Analisys do Software Engineering Institue (Carnegie Mellon).

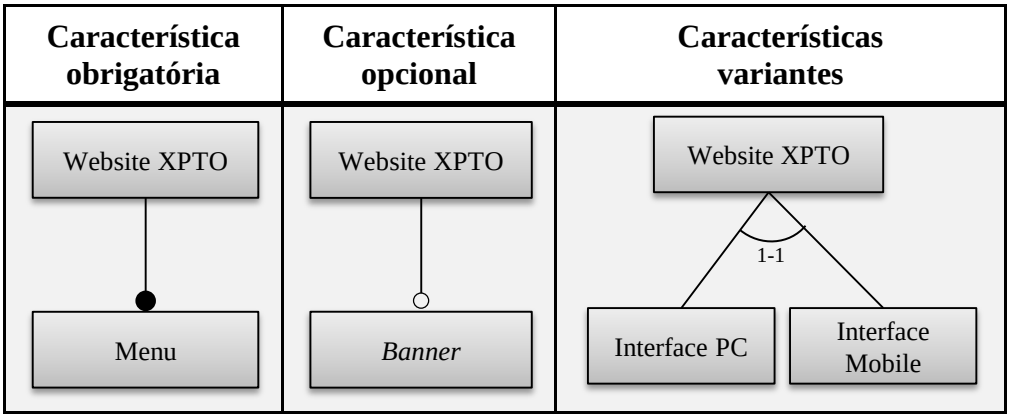

Tabela 2-iv – Notação Cardinality-Based Feature Modeling

A leitura é bastante intuitiva. O diagrama toma a forma de árvore em que a raiz é a *macro-característica* cujas características estamos a caracterizar. Estas *macro-características* são chamadas de conceitos. Como tínhamos assinalado anteriormente uma característica (ou melhor, conceito) pode ter características obrigatórias (ramo com circulo preenchido a preto), características opcionais (ramo com circulo sem preenchimento) ou características variantes. As características variantes são constituídas por alternativas agrupadas por um arco e uma indicação de cardinalidade. Esta referência permitirá aferir se é possível adicionar todas as características do grupo (e.g. 0-*) ou apenas um determinado subgrupo (e.g. 2-3).

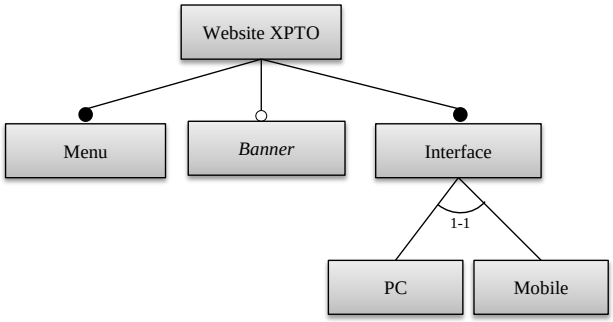

Figura 2-xiii - Exemplo de diagrama de características

A Figura 2-xiii representa o modelo de características do website XPTO. Um website baseado neste modelo de variabilidade terá, obrigatoriamente que incluir um menu, poderá ou não incluir um *banner* publicitário e possuirá inevitavelmente uma interface que poderá ser para PC ou para equipamentos móveis.

2.3.2.2 Arquitectura de Referência

As características permitem-nos definir a variabilidade de uma linha de produtos num formato abstracto, livre de implementação. Essas características são então mapeadas para elementos menos abstractos, requisitos. Naturalmente, que esse mapeamento e posterior desenho e implementação terá, não só de concretizar a funcionalidade desejada mas também assegurar a capacidade de variação idealizada numa infra-estrutura de produção apropriada.

Parte desse esforço passa pela definição e implementação de uma plataforma comum a todos os membros da família e pela disponibilização de um conjunto de componentes que, acoplados à plataforma base, assegurem a variabilidade planeada. Estes elementos, plataforma e componentes variáveis são identificados por Pohl *et al.* (Pohl, Böckle, & Linden, 2005) como "Arquitectura de Referência" da Linha de Produtos.

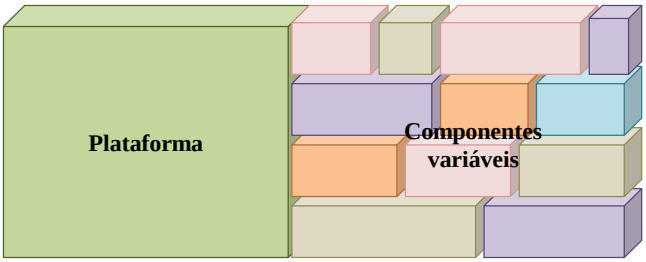

Figura 2-xiv – Arquitectura de referência da linha de produtos

A Figura 2-xiv reflecte a arquitectura da uma linha de produtos apresentada no exemplo. Idealmente, uma arquitectura de referência agrupa todos os componentes que farão parte do sistema final, uma aplicação seria apenas uma composição desses elementos previamente criados. A realidade é que esta abordagem de composição nem sempre é viável pois, em

muitos casos, não é possível antecipar a totalidade dos requisitos de todos os membros da família. Nestes casos, o desenvolvimento adicional é inevitável (caso da "Aplicação B" no exemplo).

Como seria espectável, a construção destes activos não é trivial. A plataforma deve disponibilizar os mecanismos para acoplar os componentes variáveis pré-definidos mas também exibir capacidade de extensão para acomodar componentes desenvolvidos para requisitos específicos. Para além da complexidade da plataforma, construir componentes reutilizáveis envolve um esforço adicional de abstracção para lidar com a complexidade de concepção e implementação associada.

Nos pontos seguintes destacamos algumas técnicas e tecnologias que existem actualmente para apoiar este tipo de cenário de implementação.

2.3.2.2.1 Object-Oriented Framework

A construção de *Object-Oriented Frameworks* (OOF) é uma das técnicas mais utilizadas para construir plataformas aplicacionais para famílias de sistemas. Johnson utiliza a seguinte definição: *uma OOF consiste num conjunto de classes que incorporam um conjunto de abstracções de soluções para uma família de problemas relacionados* (Johnson, 1991). Uma definição alternativa, mas igualmente válida, descreve uma OOF como *uma aplicação reutilizável, "semi-completa", que pode ser especializada para produzir aplicações personalizadas* (Fayad, 1997).

A realidade alinha-se directamente com a definição de Fayad. Na generalidade dos casos, uma OOF é uma aplicação construída com um conjunto de pontos (*hot spots*), pontos esses que são estendidos/adaptados de acordo com as características da aplicação que se quer produzir.

Não será demasiado forçado inferir das definições de OOF a seguinte afirmação: uma OOF é constituída por um conjunto de classes organizadas num formato reutilizável. Este tipo de afirmação, ainda que correcta, pode subentender que uma OOF é semelhante a uma biblioteca de classes (*library*). De facto, são conceitos diferentes.

Uma OOF diverge directamente de uma biblioteca no que respeita ao controlo de fluxo de execução. Utilizar uma biblioteca no desenvolvimento de uma aplicação significa adicionar ao código específico que se está desenvolver, um conjunto de classes e métodos que podem ser invocados quando necessários. A invocação é sempre controlada pela aplicação.

Pelo contrário, numa OOF, os métodos definidos pelo utilizador – código de adaptação da *framework* - vão, em inúmeras situações, ser chamados exclusivamente pela *framework* e não pelo código da aplicação (Johnson, 1991). Esta técnica é apelidada de Inversão de Controlo, e tal como a grande maioria dos mecanismos utilizados na construção de *frameworks*, é apoiada em dois elementos fundamentais: técnicas de orientação a objectos (O-O) e padrões de desenho.

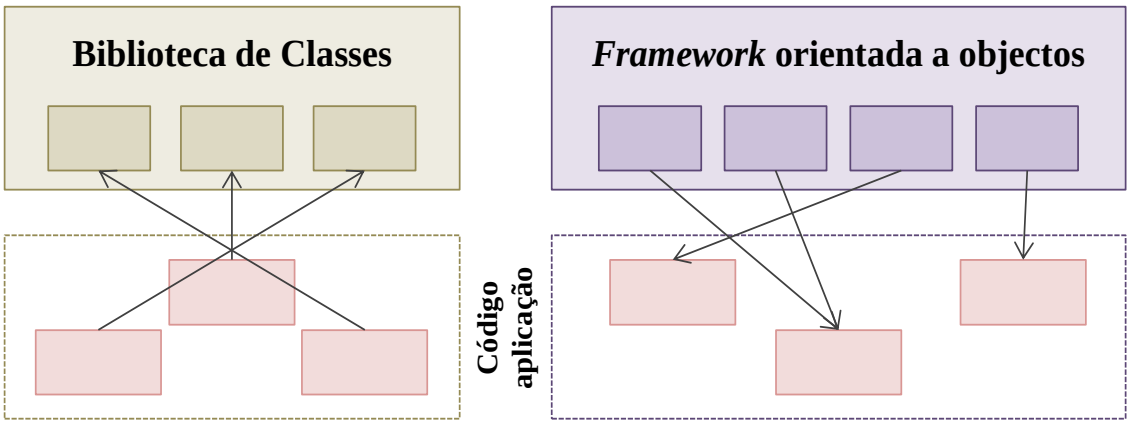

Figura 2-xv – Bibliotecas vs Frameworks. Adaptado de (Landin, Niklasson, Bosson, & Technology, 1995)

Cumulativamente, a aplicação dessas técnicas O-O e padrões de desenho variam de acordo com a estratégia de construção/consumo da *framework* adoptada (Bosch, Molin, Mattsson, & Bengtsson, 1997):

- *White Box* OOF: A construção baseada em herança e polimorfismo. Personalização mais aberta, o consumidor da OOF adiciona comportamento através da extensão de classes da *framework*;
- *Black Box* OOF: A construção baseada em composição. Personalização é mais fechada, as opções de comportamento estão pré-definidas. O papel do consumidor da OOF é configurar e interligar os componentes disponíveis.

Embora do ponto de vista teórico a diferenciação seja relevante, na prática, são muitos os casos em que uma OOF é simultaneamente *White Box* e *Black Box*.

2.3.2.2.1.1 Desenvolvimento OOF

Inúmeras técnicas podem ser aplicadas ao desenvolvimento de *frameworks*. Muitas fazem uso extensivo de mecanismos – e.g. encapsulamento, polimorfismo ou herança – disponibilizados por linguagens orientadas a objectos.

Classes abstractas e classes concretas

As classes abstractas são utilizadas para capturar as abstracções chave do domínio e suas interacções. Pretendem fornecer um desenho reutilizável, deixando para o consumidor da *framework* (*developer* da aplicação) a responsabilidade de atribuir a funcionalidade/comportamento concreto desses elementos (Froehlich, Hoover, L. Liu, & Sorenson, 1995). É também comum a *framework* ter associado uma biblioteca de componentes com classes concretas que instanciam essas abstracções. Em casos em que a funcionalidade desejada pertença a essa biblioteca, o *developer* da aplicação deve utilizá-las na instanciação da *framework*. Em casos em que isso não aconteça, o developer deve concretizar essas classes abstractas no código da aplicação.

```
namespace Framework {
   abstract class Cliente {
      abstract string RegimeFiscal();
   }
}
```

```
namespace BibliotecaFramework {
   class Particular : Cliente {
      override string RegimeFiscal() {
         return "IRS";
      }
   }
}
```

```
namespace AplicacaoA {
   class App {
      // consumindo framework
      void main() {
         // usando a classe concreta da biblioteca da framework (pré-definida)
         BibliotecaFramework.Particular cp = new BibliotecaFramework.Particular();

         …

         // usando estensão da framework específica desta aplicação
         Empresarial emp = new Empresarial();
         …
      }
   }

   // estendendo classe abstracta da framework
   class Empresarial : Framework.Cliente {
      public override string RegimeFiscal() {
         return "IRC";
      }
   }
}
```

Figura 2-xvi – Pseudo-código exemplo classes abstractas e concretas

Hook methods

Hook methods são métodos "especiais" da framework que não estão implementados. A implementação de acordo com o comportamento específico da aplicação fica a cargo do *developer* de aplicação.

```
namespace Framework {
   abstract class Operacoes {
      abstract void adicionaOperacoes(List<IOperacao> operacoes);
   }
}
```

```
namespace AplicacaoA {
   class App {
      // consumindo framework
      override void adicionaOperacoes(List<IOperacao> operacoes) {
         OperacaoConcreta oc = new OperacaoConcreta();
         Operacoes.add(oc);
      }
   }
}
```

Figura 2-xvii – Pseudo-código *hook methods*

Herança

Herança envolve especializar métodos de uma classe abstracta (ou interface) ou adicionar funcionalidades a uma classe abstracta (Froehlich, Hoover, L. Liu, & Sorenson, 1995).

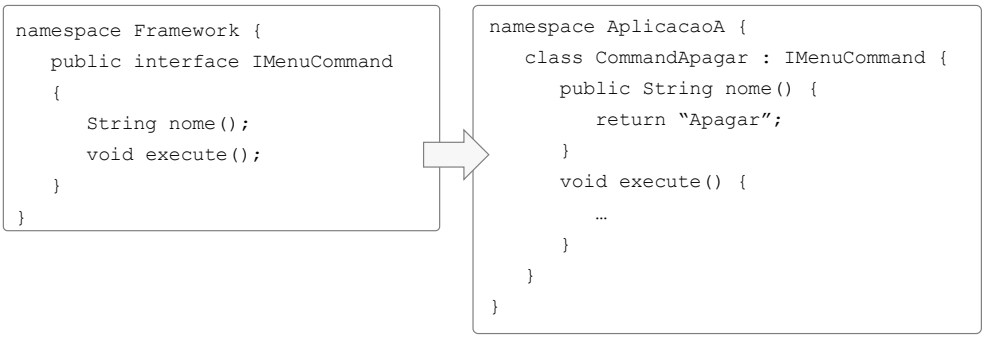

Figura 2-xviii – Pseudo-código herança

Composição

A composição é um mecanismo de variabilidade fechado, i.e., o consumo da *framework* está limitado pelas opções de configuração disponibilizadas pela *framework*. É menos flexível do que mecanismos abertos, e.g. herança, mas tem a vantagem de ser mais fácil de consumir porque iliba o *developer* de aplicação de conhecer a estrutura interna da *framework*.

Figura 2-xix – Pseudo-código composição

Padrões de desenho

Padrões de desenho estão na base de qualquer *framework* (Landin, Niklasson, Bosson, & Technology, 1995). Frequentemente, os mecanismos de variabilidade apresentados são aplicados em conjunto com padrões de desenho (e.g. hot spots e padrão Template Method).

Os padrões de desenho trazem potencial qualidade a qualquer aplicação ou *framework* por estabelecerem soluções testadas e conhecidas para problemas recorrentes. Adicionalmente, para além de apoiarem a implementação da variabilidade, nas *frameworks*, a aplicação de padrões de desenho permite assegurar que as boas práticas que os incorporam estarão presentes em todas as aplicações geradas com base na *framework*.

2.3.2.2.1.2 Desafios do desenvolvimento de OOF

Embora as OOF sejam das técnicas/tecnologias de reutilização que mais consenso reúne sobre o impacto positivo que trazem para ambientes de reutilização massiva, como seria de esperar, não apresentam só vantagens. Os autores (Steyaert, 1997) e (Fayad, 1997) reuniram alguns dos desafios que geralmente se colocam ao desenvolvimento de frameworks:

- Elevado esforço de desenvolvimento e manutenção
  Tempo de desenvolvimento elevado; é necessário uma equipa de programadores seniores e com largo espectro de competências; A *framework* rapidamente ganha grande complexidade de manutenção.
- Elevado esforço de aprendizagem para consumir *frameworks*
  A complexidade das OOF não se resume à sua construção, o próprio consumo de *frameworks*, i.e. a instanciação dos pontos de variação, pode tornar-se bastante exigente.
- Dificuldade de integração de *frameworks*
  Em projectos de escala média ou grande, uma *framework* é muitas vezes insuficiente para incorporar a funcionalidade e variabilidade desejada. Nestes casos é necessário segmentar o desenvolvimento em várias *frameworks* e posteriormente proceder à sua complexa integração.
- Requisitos em constante mudança
  O horizonte temporal alargado de desenvolvimento de uma *framework* aumenta a sua exposição a uma eventual mudança de domínio ou tecnológica durante o projecto.
- *Debugging* das aplicações mais complexo
  O *debugging* de uma aplicação baseada numa *framework* é complicado por duas razões: o código complexo da *framework* fará parte da aplicação e a inversão de controlo que referimos anteriormente não é fluxo habitual de execução.
- Gestão de configurações mais difícil
  As *frameworks* evoluem. Os componentes de acoplamento à *framework* também. As aplicações baseadas na *framework* também. A gestão de configurações deve garantir o correcto *tracking* do progresso dos vários elementos envolvidos.
- *Overfeaturing*
  Há, em muitos projectos de desenvolvimento de *frameworks*, a tendência para integrar funcionalidades na *framework* para reduzir o esforço de desenvolver mecanismos de acoplamento e componentes reutilizáveis.

2.3.2.2.2 Service-oriented Architecture

Ao contrário do que o nome possa indiciar, SOA não é uma arquitectura concreta: é algo que leva a uma arquitectura concreta (Josuttis, 2007). De facto, SOA é um paradigma – um estilo – de desenvolvimento para construir aplicações com base na composição de serviços.

As motivações deste movimento estão bastante alinhadas com as SPL. Ambos procuram reestruturar a forma de desenvolver sistemas de informação, agregando métodos para lidar eficazmente com a crescente complexidade e melhorando a produtividade e qualidade das soluções com base em reutilização massiva.

Embora existam semelhanças nos objectivos, as abordagens são completamente independentes (Institute, 2010a), i.e. é possível construir SPL sem SOA e SOA sem SPL. Contudo, tem sido crescente o número de propostas de desenvolvimento de linhas de produtos orientados a serviços, e.g. (G & Berger, 2008) ou (Cohen, 2010). Na generalidade destas propostas a plataforma comum dos membros da família toma a forma de um *bus* de serviços e a variação dos elementos é conseguida através de diferentes composições desses serviços.

Para além das vantagens associadas ao SOA que os seus defensores proclamam, e.g. o baixo nível de acoplamento de serviços, a sua aplicação no âmbito de uma família de produtos tem imediatamente associada a resolução de um problema: a falta de *standards* universais de descrição, comunicação e descoberta de componentes. Implementando a abstracção "serviço" através de Web Services (i.e. recorrendo a *standards* como SOAP, WSDL e UDDI[3]) a criação dessa plataforma de entendimento fica facilitada.

Em cenários de desenvolvimento de sistemas de informação organizacionais - em particular, aplicações de suporte à execução (e automatização) de processos de negócio – o enquadramento de arquitectura baseadas em serviços numa linha de produto sairá, em muitos casos, favorecido dada a forte vocação da abordagem SOA para o desenvolvimento desse tipo de sistemas.

---

[3] SOAP (Simple Object Access Protocol);
WSDL (Web Services Description Language);
UDDI (Universal Description, Discovery and Integration)

2.3.2.2.3 Outras técnicas de implementação aplicáveis em reutilização

Os próximos pontos contêm uma breve apresentação de mais algumas técnicas que podem ser incorporadas no desenvolvimento de uma linha de produtos (plataforma e componentes).

2.3.2.2.3.1 Compilação condicional

A compilação condicional é um mecanismo aplicável ao desenvolvimento de componentes reutilizáveis por permitir a variação que partes de código sejam, ou não compiladas, de acordo com uma condição.

No exemplo que apresentamos, em ambiente .NET, a compilação condicional é implementada através de directivas de pré-processamento interpretáveis pelo compilador.

```
#define PROTOTYPEVERSION
#if PROTOTYPEVERSION
    Console.WriteLine("Prototype version");
#endif
```

2.3.2.2.3.2 Bibliotecas

As bibliotecas são uma forma de agrupar classes e métodos para poderem ser reutilizados. Por exemplo, os componentes específicos de acoplamento a uma framework podem ser reunidos numa, ou em várias, bibliotecas. Existem dois tipos de bibliotecas (Tirilä, 2003):

- Estáticas: o código presente na biblioteca é inserido na aplicação/componente e compilado com ela.
- Dinâmicas: o código presente na biblioteca não é incluído na aplicação/componente. A biblioteca é ligada no início, ou a pedido, da aplicação.

2.3.2.2.3.3 Generic Programming

Generic Programming é um paradigma de programação para desenhar e desenvolver colecções de algoritmos reutilizáveis e eficientes (Reis, 2005). Muitas das linguagens que hoje conhecemos, e.g. Java ou C#, suportam *Generic Programming*.

Na prática, o que esta tecnologia nos permite é especificar classes ou métodos sem especificar directamente o tipo de dados que pretendemos manipular, essa informação é passada em parâmetro apenas no momento de utilização.

```
public class Stack<T>
{
   T[] m_Items;
   public void Push(T item)
   {...}
   public T Pop()
   {...}
}
```

2.3.2.2.3.4 Aspect-Oriented Programming

Aspect-Oriented Programming (AOP) é um novo paradigma de desenvolvimento que permite capturar aspectos transversais à estrutura de uma aplicação, o que torna possível a programação destes aspectos numa forma modular, evitando problemas de emaranhamento e dispersão de código (Zhang, Cai, & G. Liu, 2008).

Numa perspectiva conceptual de alto nível, a ideia central do AOP passa por efectuar a modularização de problemas recorrendo à definição de componentes funcionais e de aspectos, sendo que, estes últimos representam preocupações – *concerns* – presentes transversalmente em toda a aplicação. O sistema final resultará da conjugação (*weaving*) da implementação dos componentes funcionais e dos aspectos.

O conceito enquadra-se perfeitamente na necessidade de manter um baixo acoplamento entre componentes para facilitar a sua reutilização. Todavia, a falta de suporte nativo de AOP nas principais linguagens de programação, plataformas de desenvolvimento (e.g. Microsoft .Net ou J2EE) e respectivas ferramentas (e.g. compiladores) dificulta em muito a sua adopção como um real mecanismo alternativo.

2.3.3 DESENVOLVIMENTO ORIENTADO POR MODELOS

O paradigma emergente *Model-Driven Development* (Desenvolvimento orientado por modelos) defende a utilização de modelos como elementos nucleares do processo de desenvolvimento de software, enquanto artefactos - como documentação e código - podem ser rapidamente produzidos a partir desses modelos com recurso a transformações automatizadas (Saraiva & Silva, 2010).

A tradicional actividade de modelação produz um conjunto de documentos não estruturados ou semiestruturados - documentos de texto, folhas de cálculo, diagramas, etc. Esta abordagem tem associado algumas desvantagens (Kleppe, Warmer, & Bast, 2003):

- Não acomodam (facilmente) mudança: ainda que o objectivo de cada documento seja retratar um aspecto diferente do sistema, todos estão relacionados e a alteração num artefacto pode ter impacto nos restantes. A consistência tem de ser mantida manualmente.
- Limitam o potencial dos modelos a aspectos descritivos. Esta característica tem associado dois potenciais problemas: a perda da capacidade de automatizar passos pelo tratamento automático da informação modelada e a margem de erro introduzida pela interpretação humana do modelo.

Para evitar estas desvantagens, a abordagem do desenvolvimento orientado por modelos promove a ideia de que os elementos estáticos produzidos pelo processo tradicional devem ser substituídos por artefactos bem definidos, isto é, com um nível de estruturação e formalidade suficiente para ser interpretado, de forma automática, por computador. Para além da capacidade de automatização, a definição formal do modelo facilita a sua reutilização pois limita e direcciona a sua interpretação no futuro.

Se considerarmos a variedade de domínios existentes, podemos facilmente antecipar a dificuldade de expressão formal das abstracções definidas. Neste campo existem, essencialmente, duas abordagens:

- Utilização de uma linguagem de modelação de propósito geral (General-Purpose Modeling Language - GPML), eventualmente com extensões. Por exemplo a Unified Modelling Language (UML).
- A criação de uma linguagem própria, i.e., uma linguagem específica do domínio que estamos a considerar (Domain Specific Modeling Language – DSML).

O principal argumento das DSML face às GPML é que, em alguns cenários, a amplitude das GPML impossibilita-as de ter a riqueza suficiente para especificar aspectos importantes do domínio. No entanto, cada uma tem o seu espaço e, na maioria dos casos, não há nada que impossibilite que operem em conjunto.

Uma das abordagens mais divulgadas de desenvolvimento baseado em modelos é a Model-Driven Architecture (MDA) proposta pelo Object Management Group[4] (OMG). Esta

[4] http://www.omg.org/

metodologia baseia o desenvolvimento de aplicações na construção/transformação de modelos - *Computation Independent Model, Platform Independent Model, Platform Specific Model* (Kleppe, Warmer, & Bast, 2003). O processo de desenvolvimento consiste em criar modelos e aplicar-lhes sucessivas transformações (de acordo com regras criadas para o efeito) até à implementação – código fonte – do sistema.

O MDA nunca chegou a ser uma opção de desenvolvimento realista, em grande parte, devido à falta de *standards* para permitir, em qualquer domínio e num nível de detalhe apropriado, a especificação dos modelos e das suas regras de transformação (MELLOR, Scott, Uhl, & Weise, 2004). As SPL são uma abordagem diferente porque, à partida, limitam a capacidade de resposta a um domínio de actuação pré-estabelecido. Esta restrição favorece o aparecimento de soluções específicas de domínio, ao invés dos *standards* universais. Um exemplo desses elementos específicos de domínio são as *frameworks* de objectos que vimos anteriormente, outro exemplo são as Domain Specific Languages (em que se incluem as Domain Specific Modeling Languages).

2.3.3.1 Domain Specific Languages

A ideia base de uma *Domain Specific Language* (DSL) é estabelecer uma linguagem interpretável por computador, destinada a um problema concreto, por oposição às *General Purpose Language* que são destinadas a um qualquer tipo de problema (Fowler, 2011).

No desenvolvimento actual, estamos constantemente a interagir com DSLs, e.g. quando construímos regras de estilo com Cascading Style Sheet (CSS) ao definirmos instruções para manipulação de dados através Structured Query Language (SQL). Uma característica destas linguagens é a sua elevada capacidade de expressão dentro do domínio mas completa inaptidão a outros contextos (para exemplificar podemos propor o exercício de tentar definir uma regra de estilo com SQL…).

A construção de DSL é uma das técnicas mais utilizadas em ambientes de reutilização. Por um lado, a utilização de uma linguagem estabiliza os conceitos e assegura que todas as descrições/especificações seguem o mesmo padrão, por outro, os modelos formais trazem potenciais ganhos de produtividade pela automatização de tarefas que permitem.

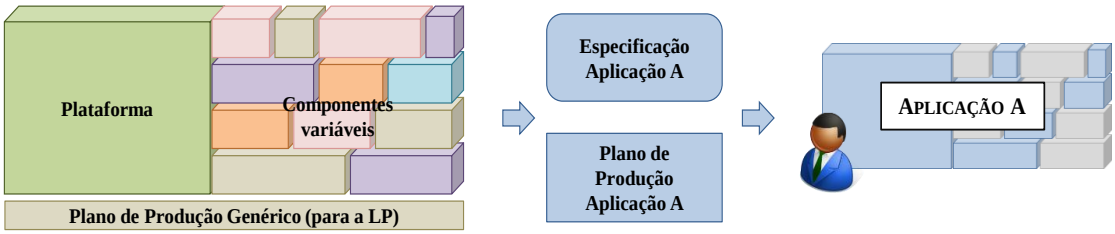

Figura 2-xx – Produção da Aplicação A

No exemplo apresentado no ponto 2.3.1, qualquer elemento reutilizável poderia ser alvo de uma DSL, i.e. potencialmente suportaria uma descrição formal através de uma linguagem criada para o efeito. Para exemplificar de forma mais concreta podemos atentar aos elementos envolvidos na produção da "Aplicação A" que estão representados na Figura 2-xx. Este cenário pode sugerir os seguintes alvos de criação de DSL e potenciais automatizações de tarefas:

- DSL com abstracções sobre as características disponibilizadas pela LP
  Se a especificação da "Aplicação A" for representada num modelo constituído por essas abstracções, a implementação da "Aplicação A" pode vir a ser automatizada.
- DSL com abstracções sobre elementos do plano de produção da LP
  Se o plano de produção da "Aplicação A" for representado num modelo formal com os elementos envolvidos na produção, a configuração do ambiente de desenvolvimento, i.e. da linha de montagem, estaria em condições de ser automatizada.

Estes dois cenários de automatização serão detalhados mais à frente em secção dedicada.

2.3.3.1.1 Fundamentos DSL

Uma linguagem pode tomar várias formas de apresentação. Os exemplos que referimos anteriormente, CSS e SQL, são linguagens textuais mas por exemplo, o UML já é uma linguagem com uma notação, essencialmente, gráfica. Para além da diferença de apresentação o próprio intuito das linguagens é diferente. O UML é uma linguagem de modelação para especificar diferentes aspectos de um sistema de informação enquanto o CSS ou o SQL têm um intuito de expressar instruções de apresentação e manipulação de dados, respectivamente.

Num contexto SPL, a criação de linguagens específicas de domínio tem um papel de apoio à modelação/formalização de especificações com vista à obtenção de automatizações. É com esse enquadramento que apresentaremos nos próximos pontos os aspectos envolvidos na construção de uma DSL.

Uma linguagem de modelação é usualmente baseada em algum tipo de modelo computacional em função do contexto a modelar (Kelly & Tolvanen, 2007). Esta opção divide-se essencialmente em dois tipos:

- Modelação de aspectos estáticos (e.g. linguagens para construir modelos de classes, modelos de características ou modelos de componentes)
- Modelação de aspectos dinâmicos e comportamentais (e.g. linguagens para definir máquinas de estados, diagramas de interacção ou *workflows*)

Independentemente do tipo de linguagem, a construção de uma DSL consiste na definição de uma sintaxe abstracta, uma sintaxe concreta e semântica (Lenz & Wienands, 2006).

2.3.3.1.1.1 Sintaxe abstracta

A sintaxe abstracta de uma linguagem caracteriza, de uma forma abstracta, os tipos de elementos que fazem parte da linguagem e as regras que estipulam como esses elementos

podem ser combinados (Greenfield, Short, Cook, & Kent, 2004), i.e., corresponde à estruturação conceptual da linguagem.

Existem várias formas para especificar sintaxes abstractas. Uma das formas mais utilizadas, em particular para linguagens textuais, é a definição de gramáticas livres de contexto (*contexto-free grammars*), por exemplo através da notação Backus–Naur Form (BNF). Esta abordagem é ainda bastante utilizada, contudo, num contexto de desenvolvimento orientado por modelos, a meta-modelação é o mecanismo mais usual para especificar sintaxes abstractas (Lenz & Wienands, 2006).

O objectivo da meta-modelação é a criação de meta-modelos com a especificação dos elementos disponibilizados na linguagem, i.e. com a especificação da sintaxe abstracta da linguagem. Na realidade, o meta-modelo é ele próprio um modelo. Feita esta observação, seria pertinente perguntar como seria definido o próprio meta-modelo. Uma natural resposta seria "através de um meta-meta-modelo". É fácil inferir que, com este raciocínio, entraríamos num ciclo infinito. Uma estrutura de quatro camadas de modelação é o *standard* adoptado para a resolução deste problema.

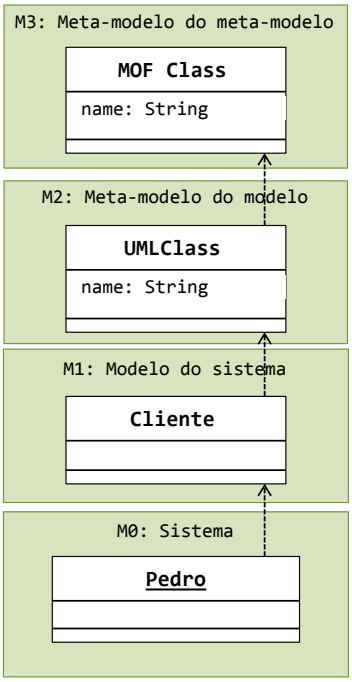

Figura 2-xxi – Arquitectura quatro camadas OMG. Adaptado de (Kleppe, Warmer, & Bast, 2003)

A Figura 2-xxi representa a estrutura de quatro camadas de modelação proposta pelo OMG. Esta arquitectura é partilhada por todas as linguagens estandardizadas pelo instituto, i.e. todas essas linguagens derivam da metalinguagem utilizada na camada de topo (M3), o Meta-Object Facility (MOF). O exemplo da figura refere-se à linguagem UML

Especificado através da linguagem de topo (MOF), na camada M2, temos o meta-modelo UML. Para simplificar a análise e representação, na figura, este meta-modelo é limitado ao elemento "UML Class". Na camada M1 estão os modelos de sistema, i.e. estão os modelos com a instanciação dos elementos definidos no meta-modelo da camada M2. No exemplo, o elemento "UML Class" é instanciado com a abstracção "Cliente" – na actividade de

modelação tradicional trabalhamos nesta camada, criando modelos com base nos elementos definidos pelo meta-modelo. Por último temos a camada M0 com as instâncias de sistema, estas representam a instanciação final que ocorre com o sistema em execução.

Generalizando, podemos concluir que os elementos disponibilizados na camada n como a instanciação dos elementos disponibilizados pela camada n+1 com excepção da camada de topo em que os elementos devem ser instanciações de elementos da própria camada (Kleppe, Warmer, & Bast, 2003).

Ao contrário da abordagem tradicional - em que criamos modelos com base em metalinguagens já existentes (e.g. UML) - no contexto DSL, criamos meta-modelos (camada M2) e depois construímos modelos com base nesses meta-modelos (camada M1). A linguagem de topo (M3) utilizada para a construção do meta-modelo varia consoante o contexto ou, mais concretamente, varia consoante a ferramenta de meta-modelação que for utilizada.

2.3.3.1.1.2 Sintaxe concreta

O propósito de uma sintaxe concreta é disponibilizar um mecanismo de interpretação e manipulação dos elementos definidos pela sintaxe abstracta da linguagem (Greenfield, Short, Cook, & Kent, 2004).

A sintaxe abstracta define os elementos da linguagem, as suas relações e as regras para sua utilização mas mantém-se num nível conceptual, não estabelecendo como se dará a interacção com os utilizadores da linguagem. Essa capacidade de interacção é estabelecida pelo mapeamento entre os conceitos e uma qualquer forma de apresentação, e.g. texto, elementos gráficos, tabelas, forms ou matrizes. Esses elementos de apresentação e o relacionamento com os conceitos abstractos formam a sintaxe concreta da linguagem.

2.3.3.1.1.3 Semântica

A semântica é parte da linguagem que estabelece o significado dos seus elementos. Num contexto de linguagens específicas de domínio, a semântica é definida directamente pelo problema (domínio) (Kelly & Tolvanen, 2007). Em resultado dessa circunstância, tipicamente, a criação de uma DSL apenas requer a criação de uma sintaxe abstracta e de uma sintaxe concreta ficando a semântica pré-estabelecida pelo domínio.

2.3.3.1.2 Mapeamentos e Transformações

Um dos núcleos do desenvolvimento orientado por modelos é a transformação de modelos. O MDA é um exemplo em que essa posição é bem patente. Como tínhamos referido anteriormente, no MDA o sistema é gerado em função de sucessivas transformações de modelos. Essas transformações são feitas com base em mapeamentos (regras de transformação) que definem, formalmente, a relação que existe entre o formato de entrada e o formato de saída.

Existem vários tipos de transformação. A título de exemplo, podemos referir uma das mais usuais: a transformação de um modelo (e.g diagrama de classes UML) em implementação

(código-fonte). Czarnecki resume os tipos de transformação da seguinte forma (Czarnecki & Eisenecker, 2000):

- Vertical: transformação que altera o nível de abstracção da especificação de entrada. Se a alteração for no sentido de baixar o nível de detalhe falamos de refinamento, se for no sentido de elevar o nível de detalhe falamos de abstracção. Isto resulta numa manutenção de estrutura mas alteração do detalhe da informação de entrada.
- Horizontal: transformação que altera a estrutura da especificação mas não muda o nível de abstracção. Um *refactoring* de código é um exemplo de transformação horizontal.
- Oblíquas: operação que muda a estrutura e o detalhe da informação de entrada, resulta da combinação de transformações verticais e horizontais (e.g. tipo de operações efectuadas por um compilador GPL).

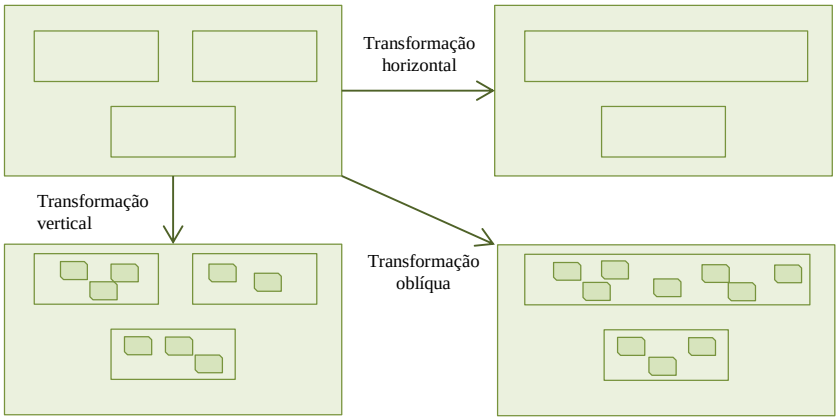

Figura 2-xxii – Tipos de transformação. Adaptado de (Czarnecki & Eisenecker, 2000)

2.3.4 AUTOMATIZAÇÃO

Na secção 2.3.2 apresentámos técnicas e tecnologias para construir plataformas e componentes aplicáveis ao desenvolvimento de famílias de produtos, na secção 2.3.3 expusemos os fundamentos do desenvolvimento baseado em modelos com foco particular na criação de linguagens para modelar aspectos específicos de domínio. Nesta secção analisamos a forma como os pontos anteriores se interligam e se integram numa SPL.

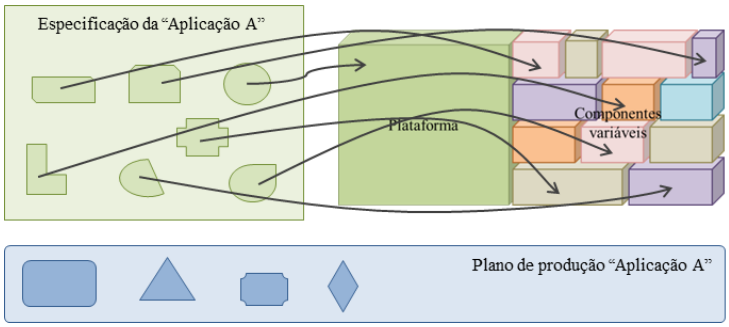

Figura 2-xxiii – Produção "Aplicação A" (especificação vs implementação)

A Figura 2-xxiii apresenta uma visão parcial mas mais detalhada do cenário apresentado na Figura 2-xx: a produção da "Aplicação A". A especificação da "Aplicação A" é materializada num modelo específico de domínio. Este modelo é construído com base numa DSL, cujas abstracções se relacionam com elementos da linha de produtos. Estas relações permitirão que, ao estabelecer-se o modelo da "Aplicação A" se esteja (indirectamente) a referenciar os elementos da LP que implementam essa especificação. Analogamente, o plano de produção da aplicação também pode ser especificado num modelo construído com base numa DSL com abstracções sobre as configurações do ambiente de desenvolvimento necessárias para concretizar a produção da aplicação.

2.3.4.1 Geradores

Um gerador é uma ferramenta que executa transformações sobre modelos (Lenz & Wienands, 2006). Um exemplo de gerador que utilizamos regularmente são os compiladores de uma linguagem de programação como C++, Java ou C#. Estas aplicações executam transformações (oblíquas) ao código de alto-nível que especificamos por forma a originar código máquina ou código intermédio para ser interpretado por um compilador *Just-in-Time* (caso de Java e C#). De forma simples, podemos definir um gerador como uma aplicação que opera sobre um conjunto de parâmetros de entrada, por forma a transformá-los num qualquer output.

Figura 2-xxiv – Gerador

Um gerador não é nenhuma novidade. Para além dos compiladores que já referimos, utilizamos recorrentemente geradores no desenvolvimento tradicional quando, por exemplo, fazemos *drag-and-drop* de um controlo para uma interface de utilizador ou quando recorremos a soluções *Object-Relational Mapping* (ORM) para transformar um modelo de classes na sua representação relacional.

O facto de construirmos geradores particulares para um contexto específico é que representa a inovação. Para além do potencial de geração ser maior, um gerador construído para um domínio específico, traz mais controlo sobre a qualidade do *output* da geração do que um gerador genérico.

Continuando com o exemplo que nos têm acompanhado os geradores são a peça que nos faltava para concretizar as potenciais automatizações que referimos no ponto 2.3.3.1:

- Automatização da implementação da "Aplicação A"
  Um gerador pode, dada a especificação da Aplicação num modelo específico de domínio, automatizar as transformações respectivas aos mapeamentos problema-solução (ver Figura 2-xxiii). Isto pode envolver a geração do código que implementa os componentes ou de código que "cole" um componente existente à plataforma base;
- Automatização da implementação do Plano de Produção
  Um gerador pode, dado o plano de produção especificado num modelo específico desse domínio, gerar código que adapte ferramentas de desenvolvimento (e.g. *plugins*) ou que apoie e automatize a assemblagem de elementos (e.g. *wizards* ou *recipes*).

Embora os geradores sejam, normalmente, associados à produção automática de código, a verdade é que podem fazer muito mais do que isso. (Kelly & Tolvanen, 2007) apresentam os potenciais alvos de trabalho de um gerador:

- Validação adicional de aspectos de domínio nos modelos de entrada;
- Criação de protótipos e simulação;
- Configuração, *Packaging* e *Deployment;*
- Documentação;
- Testes e casos de testes;
- Reportar métricas e estatísticas sobre a utilização dos conceitos do meta-modelo.

A filosofia SPL sugere explicitamente a reutilização de vários destes componentes, por isso mesmo, é extremamente importante adoptar uma estratégia de geração que faça uso das totais capacidades dos geradores.

2.3.4.1.1 Implementação de Geradores

Um gerador específico de domínio é implementado com recurso ao meta-modelo de uma, ou várias, DSLs. Como vimos anteriormente um modelo é uma instância, uma concretização, de um meta-modelo. Um gerador não pode trabalhar exclusivamente ao nível do modelo pois ficaria "preso" a esse modelo, deve por isso recorrer a conceitos abstractos definidos no meta-modelo.

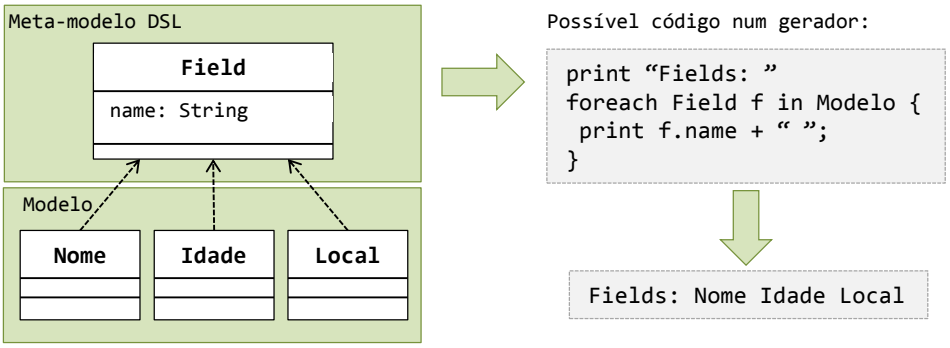

Figura 2-xxv – Exemplo de relação meta-modelo / modelo / gerador

Existem três abordagens à construção de geradores (Czarnecki & Eisenecker, 2000):

- Desenvolver o gerador de raiz como uma aplicação independente
  Opção que envolve mais esforço. Passa por implementar os mecanismos de leitura de entrada (e.g. *parsing* ou análise léxica), interpretador de regras de transformação bem como motor de execução dessas mesmas regras;
- Desenvolver o gerador com base nas capacidades de uma linguagem
  Abordagem menos exigente do que a anterior mas limita o tipo de *inputs* do gerador à variedade disponibilizada pela linguagem;
- Desenvolver o gerador com base numa infra-estrutura de geração
  Construir o gerador com base em mecanismos e ferramentas especializados na construção de geradores.

Hoje em dia, a quantidade e qualidade das ferramentas existentes fazem da terceira abordagem, a solução mais adequada.

2.3.4.1.2 Modelos, Geradores e Frameworks aplicacionais

Modelos, geradores e frameworks aplicacionais são técnicas/tecnologias independentes, contudo, a sua utilização conjunta tem sido adoptada por várias *frameworks* de desenvolvimento, em particular, para contextos SPL. As Software Factories que apresentamos anteriormente são um exemplo dessa posição.

A ideia central desta abordagem passa por permitir a construção rápida e simples de aplicações utilizando modelação de alto-nível de abstracção. Mais uma vez recorremos ao exemplo do ponto 2.3.1 para demonstrar a forma de operacionalizar este método.

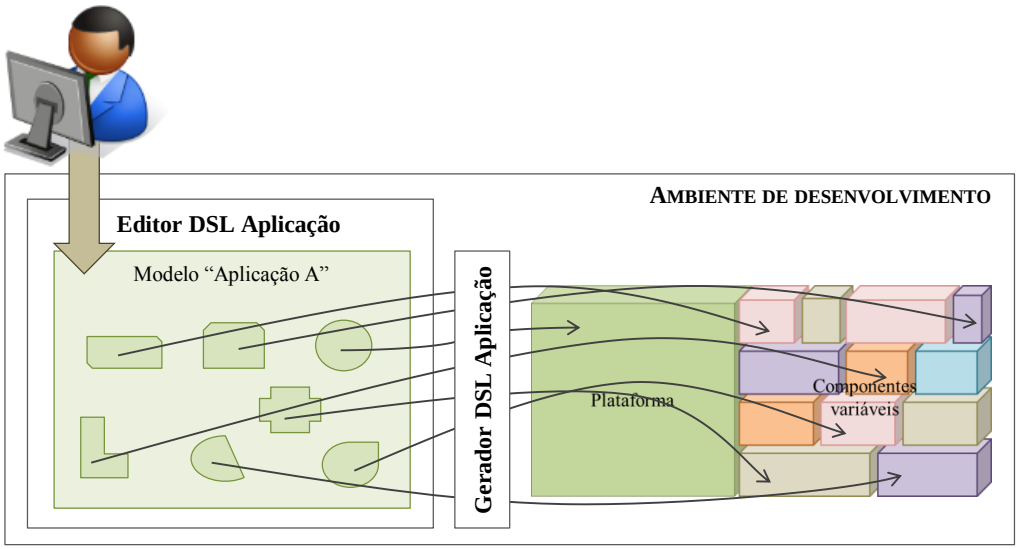

Figura 2-xxvi - Modelo / Gerador / Framework

Como a Figura 2-xxvi denota, a articulação Modelo / Gerador / Framework foi sendo desvendada à medida que fomos apresentando cada um dos pontos respectivos. Do ponto de vista prático, temos um utilizador da infra-estrutura de produção que especifica o modelo da aplicação recorrendo à linguagem de modelação criada para o efeito. Os conceitos definidos nesta DSL são idealmente estabelecidos num alto nível de abstracção, próximo do negócio. Isto faz com que o utilizador da infra-estrutura não tenha de ser um *developer*, e em muitos casos, e.g. onde o negócio é complexo, seja até preferível que a aplicação seja modelada por alguém menos técnico, por exemplo, por um analista de negócio.

Este modelo é então injectado num gerador especificamente criado para interpretar o seu conteúdo e transformá-lo em código que instancie e configure a *framework* e componentes pertencentes à linha de produtos. Como referimos anteriormente, para além da função de instanciação da framework (a única representada na figura), o gerador pode utilizar a informação do modelo para gerar, por exemplo, documentação, casos de teste ou relatórios de instrumentação.

Face ao exposto, podemos concluir que este é um método versátil que permite assegurar, de forma automatizada, a correcta instanciação da *framework* através de um mecanismo de modelação de alto nível de abstracção.

2.3.4.2 Infra-estrutura de desenvolvimento

Uma infra-estrutura de produção tradicional engloba uma linha de montagem. A tradução deste conceito para o desenvolvimento de aplicações passa pela criação/adaptação do ambiente de desenvolvimento à linha de produtos e em particular ao plano de produção. Actualmente, a abordagem mais comum para a implementação deste cenário, resume-se à adaptação de um Integrated Development Environment (IDE) como o Eclipse ou Microsoft Visual Studio.

A configuração do IDE deve ser baseada no contexto da linha de produtos, aspectos específicos do sistema a produzir e nos mecanismos de extensão/adaptação disponibilizado pela(s) ferramenta(s) utilizada(s). Ainda que cada ferramenta disponibilize variados mecanismos de adaptação, existem alguns que já estão presentes na generalidade das opções:

- *Plugins*
  Um *plugin* é um mecanismo de extensão de uma ferramenta. É extremamente polivalente e pode permitir um grau de personalização bastante elevado. Tipicamente um *plugin* é um nova funcionalidade que é definida por quem adapta a ferramenta e pode, por exemplo, servir para executar automaticamente um conjunto de tarefas ou adicionar um nova funcionalidade.
- *Wizards*
  Os *wizards* visam essencialmente acompanhar o utilizador/programador na execução de uma tarefa. No âmbito das SPL, os *wizards* são particularmente interessantes quando a adaptação de um componente variável ou a ligação de um componente à plataforma base tem de ser manual. Neste caso o processo pode envolver uma complexa sequência de passos que se não for correctamente realizada originará problemas. Um *wizard* pode garantir que as alterações só são tornadas efectivas se todos os passos forem correctamente cumpridos.
- *Recipes*
  Como o nome indica são receitas, i.e. são passos pré-definidos para a execução de alguma tarefa ou construção de algum *output*. São semelhantes aos *wizards* mas não visam o acompanhamento do utilizador a executar uma sequência de passos, visam antes a execução automática de passos pré-definidos (sem intervenção do utilizador). Tem muitas inspirações nos mecanismos de *batch* ainda muito utilizados em automatização de tarefas em variados sistemas operativos.

3 PROPOSTA DE SOLUÇÃO

3.1 VISÃO GERAL

Neste capítulo apresentamos a metodologia de desenvolvimento que propomos. A proposta assume como principal preocupação a disponibilização de um conjunto de actividades que constituam uma abordagem de cariz prático e realista para desenvolver sistemas com base em famílias de produtos. São três, os pilares que sustentam a nossa metodologia:

- Englobar práticas e conhecimento proveniente de programas SPL reais;
- Seguir um desenvolvimento orientado a modelos
- Potenciar ao máximo a automatização de desenvolvimento

O modelo proposto foi apelidado de Software Product Line Unified Process (SPLUP). Esta designação deriva da sua orientação a linhas de produto e da vincada inspiração em metodologias baseadas no processo unificado (UP) de desenvolvimento – RUP e OpenUP (ver Anexo – RUP e OpenUP).

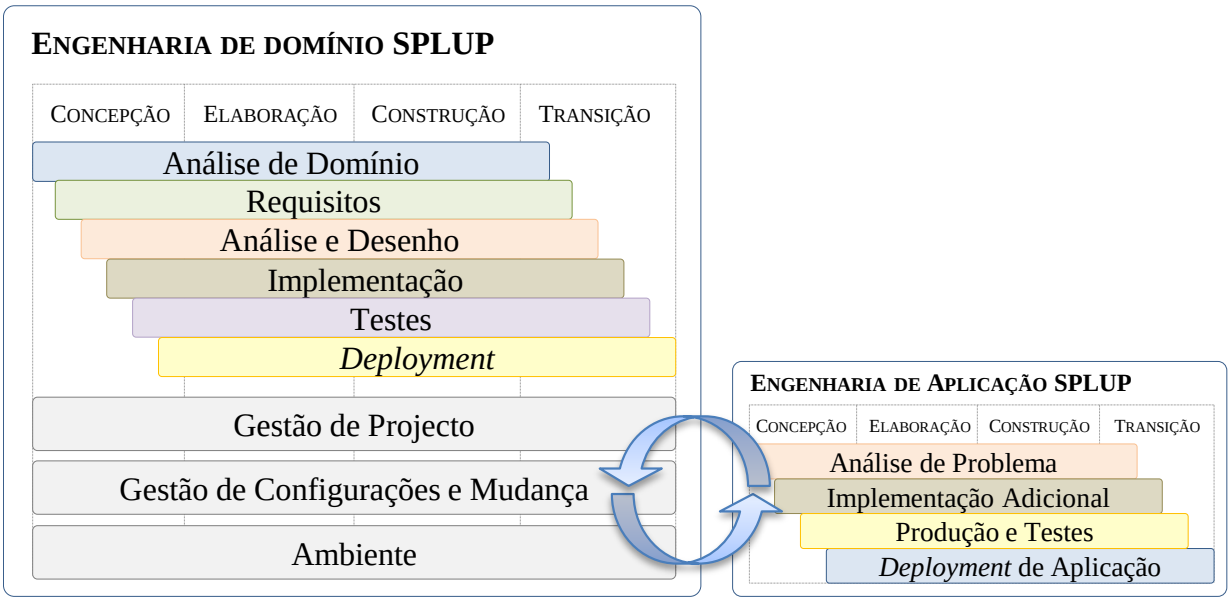

Figura 3-i – SPLUP

Trata-se de um modelo iterativo e incremental um pouco diferente dos moldes a que estamos habituados pois uma linha de produtos tem um ciclo de vida mais longo do que um projecto tradicional.

Os próximos pontos detalharão a metodologia.

3.2 FUNDAMENTOS

3.2.1 ENGENHARIA DE DOMÍNIO E ENGENHARIA DE APLICAÇÃO

Tanto nas *frameworks* apresentadas como na generalidade da literatura, o desenvolvimento baseado em famílias de produtos assenta em dois processos diferentes: a engenharia de domínio e a engenharia de aplicação. Esta distinção é mantida na nossa proposta.

Fazendo a ponte entre as características específicas de cada processo e as particularidades das abordagens formais e ágeis de desenvolvimento, na SPLUP, a engenharia de domínio toma a configuração de um processo formal enquanto a engenharia de aplicação é formatada pelos princípios das metodologias ágeis. Passamos a apresentar os argumentos que, na nossa opinião, sustentam esta posição.

3.2.1.1 Principais razões para a adopção de processo formal na engenharia de domínio:

- Qualquer processo de engenharia de domínio é iniciado pela análise desse mesmo domínio. Esta actividade, na perspectiva de desenvolvimento tradicional traduz-se no estudo do negócio para o qual o sistema de informação está a ser desenvolvido, disciplina essa que é tipicamente associada a metodologias formais já que os processos ágeis defendem uma participação próxima do cliente durante todo o desenvolvimento como fonte alternativa de conhecimento do problema;
- O projecto de engenharia de domínio é, pela sua própria natureza, um processo pesado, de execução e visão a longo prazo, com necessidade de compromisso transversal a toda a organização, com custos consideráveis e, tipicamente, de elevado risco. Este é o cenário típico para a aplicação de um método formal;
- A visão a longo prazo dificulta ainda mais a, tradicionalmente complexa, relação de colaboração do cliente durante a execução do projecto. O contributo do cliente é essencial para qualquer projecto de desenvolvimento, nas SPL não é diferente. Porém, ao contrário da perspectiva tradicional, num contexto SPL, como o âmbito transcende o desenvolvimento de um único sistema, as características das solicitações efectuadas ao cliente podem não ser directamente imputáveis aos seus interesses.
- Um modelo de organização para um projecto de grande dimensão que se baseie em auto-organização de equipas multidisciplinares, eventualmente geograficamente distantes, é no mínimo difícil, para não dizer impossível de implementar;
- Mesmo com pequenos incrementos, num projecto complexo, como o da construção de uma arquitectura de referência flexível e adaptável a um conjunto de sistemas, tipicamente recorre-se a ciclos de desenvolvimento maiores do que semanas. Neste contexto, o princípio de, em períodos curtos e regulares, ter algo que represente valor para o cliente não é aplicável.

3.2.1.2 Principais razões para a adopção de um processo ágil na engenharia de aplicação:

- A engenharia de aplicação lida directamente com o cliente, procurando a solução ideal para as suas necessidades. A saudável colaboração entre ambos é um evidente requisito para que esse objectivo seja atingido. Neste campo, as SPL têm uma

vantagem assinalável: os processos de engenharia de aplicação são de execução rápida (é a vantagem da reutilização massiva) por isso, a disponibilidade requerida ao cliente é bastante limitada no tempo;
- O processo de desenvolvimento é simples (baseado no processo definido pela engenharia de domínio). Por isso mesmo, os ciclos de desenvolvimento são tipicamente curtos, na ordem dos dias ou, no máximo, semanas com entregas de software usável regulares.
- O esforço aplicado à análise do problema é bastante limitado, pois o contexto está previamente estudado pela engenharia de domínio. A orientação principal do trabalho da engenharia de aplicação centra-se no mapeamento do problema do cliente no problema(s) endereçado(s) pela linha de produtos.

3.2.2 INTEGRAÇÃO DE PROPOSTAS

Desde o início da elaboração do trabalho, que existiu uma preocupação em adoptar uma perspectiva realista e prática do modelo a desenvolver. Essa preferência levou-nos a suportar o modelo em alguns pressupostos que agora apresentamos:

- O modelo seria baseado em extensões/adaptações de metodologias (de desenvolvimento tradicional) existentes. O seu mapeamento para as SPL teria duas potenciais vantagens. Por um lado, estaríamos a reforçar a credibilidade da nossa proposta ao aplicar práticas e conceitos testados na indústria há vários anos, por outro, esse reconhecimento generalizado facilitaria a transição da visão tradicional de desenvolvimento para a perspectiva de construção de famílias de produtos.
- O RUP constituiria uma boa base para a engenharia de domínio, a sua elevada versatilidade permitiria uma robusta adaptação a um contexto SPL. Paralelamente, o OpenUp seria uma boa opção para apoio à definição da engenharia de aplicação, pois, para além de ser um processo ágil adaptável, a interligação com o processo de engenharia de domínio sairia facilitada pela marcada inspiração e partilha de práticas que obtém do RUP.
- O enquadramento principal para as adaptações a linhas de produtos seria dado pela FSPLP (Software Engineering Institute). Este trabalho define de forma estruturada e harmonizada um conjunto de práticas que reúne experiência de um largo número de projectos SPL em várias organizações de várias partes do mundo, constituindo, até à data, um dos mais importantes contributos para o desenvolvimento baseado em famílias de produtos;
- Uma perspectiva mais técnica seria fornecida pelas Software Factories. Embora não forneçam uma metodologia para desenvolver famílias de produtos, as Software Factories apresentam uma proposta de articulação de tecnologias que tem sido aplicada com êxito em várias iniciativas e que considerámos como um bom suporte para a nossa proposta.

A abordagem seguida para integração das propostas está detalhada no Anexo – Processo de Integração SPLUP.

3.2.3 PROGRAMA SPLUP

Na análise mais simples do problema, o modelo SPLUP representa um projecto, cuja execução unitária se traduz na criação de uma infra-estrutura de produção e geração de aplicações com base nessa plataforma. Numa perspectiva mais realista, ou o domínio escolhido é muito restrito ou então a organização de uma iniciativa SPLUP é bastante mais complexa e serão necessárias várias iterações ao modelo para construir e evoluir a infra-estrutura de produção e correspondentes aplicações. Este panorama toma a configuração de Programa, ou seja, a construção, utilização e evolução da LP é executada segundo um macroprojecto (Programa) constituído por vários projectos.

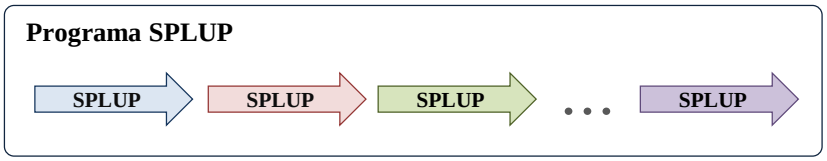

Figura 3-ii – Programa SPLUP

Cada projecto SPLUP é constituído por um processo de engenharia de domínio e *n* processos de engenharia de aplicação. A Figura 3-iii representa um exemplo de um programa SPLUP com dois projectos (assinalados a azul e vermelho).

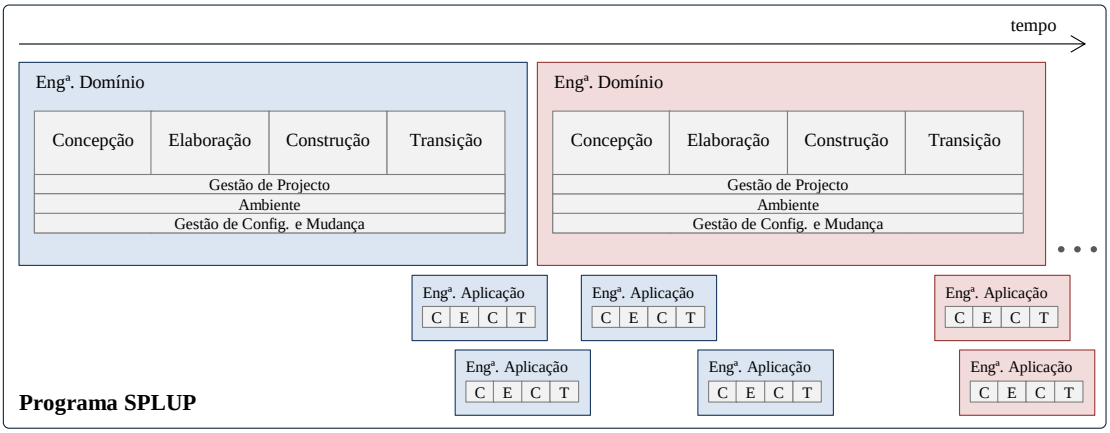

Figura 3-iii - Exemplo de execução Programa SPLUP

À semelhança do RUP e OpenUp, os processos SPLUP (domínio e aplicação) têm 4 fases: Concepção, Elaboração, Construção e Transição. Ainda que a figura não faça referência (por uma questão de clareza de apresentação), para cada uma destas fases poderão ser necessárias várias iterações. As iterações de fase são os ciclos mais pequenos do modelo e tipicamente duram entre poucos dias (Engª de aplicação) a poucos meses (Engª de domínio). O Programa em si poderá durar vários anos.

O último ponto que gostaríamos de notar é o deslocamento dos processos de engenharia de aplicação em relação aos processos de engenharia de domínio respectivos. Como cada projecto inclui a construção/evolução da infra-estrutura de produção, esta é apenas utilizada pela engenharia de aplicação no fim do projecto em curso ou durante a execução do seguinte.

3.2.4 ARTICULAÇÃO DE PROCESSOS

Um dos aspectos que consideramos mais relevantes na nossa proposta, em comparação com as propostas existentes, é a explícita definição da articulação entre o processo de engenharia de domínio e o processo de engenharia de aplicação. É fácil perceber, e a generalidade da literatura também o refere, que sem uma integração dos dois processos não existirão bons resultados.

A forma de integração escolhida foi em grande parte motivada pelo carácter distinto que o RUP atribui a três das suas nove disciplinas: Gestão de Projecto, Gestão de Configurações e Mudança e Ambiente. Estas disciplinas tomam um carácter de acompanhamento transversal à execução do projecto, não participando directamente na criação do sistema mas criando condições para que as tarefas concretas de desenvolvimento sejam correctamente executadas. Aproximando esta transversalidade com uma análise cuidada das áreas de prática e dos riscos mais comuns associados a uma ineficiente integração dos processos, considerámos o seguinte no modelo SPLUP:

- Disciplina Gestão de Projecto
  O gestor de projecto da engenharia de domínio seria responsável por nomear um gestor, que dependesse de si directamente, para cada processo de aplicação. Este gestor de aplicação reportaria ao gestor de domínio mas manteria total independência executiva no processo de produção da aplicação.
- Disciplina Gestão de Configurações e Mudança
  A gestão de Configurações é um dos aspectos impossíveis de separar em dois processos. Os activos criados pela engenharia de domínio têm de ser sujeitos à mesma gestão de configurações dos itens criados por cada processo de aplicação. O mesmo se passa com a gestão de mudança, quer a solicitação de alteração provenha do desenvolvimento da infra-estrutura (engª domínio) ou de clientes atendidos pelo processo de aplicação, deverá ser tratada de forma idêntica.
- Disciplina Ambiente
  A disciplina de Ambiente relaciona-se com a correcta execução do processo de desenvolvimento e com a adequada utilização das ferramentas. Este aspecto é muito relevante para os dois processos mas em particular para a engenharia de aplicação pois é necessário certificar que estão a executar correctamente o plano de produção estipulado pela engenharia de domínio.

Naturalmente que na prática, a integração tem de extrapolar estas áreas, e.g. os componentes implementados tem de ser disponibilizados à equipa de aplicações, mas o que fica assegurado pelo modelo através da definição destas três disciplinas é o fluxo de trabalho necessário para gerir eficientemente essas situações.

Nota: Na apresentação do modelo estas três disciplinas surgem sempre no processo de engenharia de domínio para facilitar a representação, manter a coerência com base RUP e porque é nesse processo que está a maioria do esforço. À parte desta opção de apresentação, os fluxos de trabalho e descrições das disciplinas consideram explicitamente a engenharia de aplicação.

3.3 SPLUP – ENGENHARIA DE DOMÍNIO

O processo de Engenharia de Domínio da SPLUP está organizado em 9 disciplinas:

1. Análise de Domínio
2. Requisitos
3. Análise e Desenho
4. Implementação
5. Testes
6. *Deployment*
7. Gestão de Projecto
8. Gestão de Configurações e Mudança
9. Ambiente

As primeiras seis são exclusivas deste processo e destinam-se à organização do trabalho necessário para construir a infra-estrutura de produção. As três últimas são partilhadas com a Engenharia de Aplicação e destinam-se ao apoio operacional do desenvolvimento da infra-estrutura e ao apoio de articulação entre os dois processos de engenharia.

O processo está organizado em quatro fases, cada uma com um macro objectivo associado (apresentados na Tabela 3-i) que deve definir se a fase está, ou não, concluída.

Fase	**Principal objectivo**
Concepção	Assegurar alinhamento entre os âmbito da LP e os objectivos estratégicos da organização
Elaboração	Garantir uma especificação estável dos elementos da infra-estrutura de produção que implementará a Linha de Produtos
Construção	Ter os elementos que constituem a infra-estrutura de produção implementados com respectiva documentação de apoio à sua utilização
Transição	Garantir a correcta disponibilização da infra-estrutura de produção à equipa da Engenharia de aplicações

Tabela 3-i – *Milestones* Engenharia de domínio SPLUP

De seguida apresentamos em detalhe as referidas disciplinas.

3.3.1 ANÁLISE DE DOMÍNIO

SPLUP – Análise de Domínio
Tagline
A Análise de Domínio é a disciplina destinada à captura e estruturação da informação sobre o domínio a que se destina a Linha de Produtos.
Fluxo de Trabalho
Ver Anexo – Fluxos de Trabalho SPLUP
Objectivos
Definir as características gerais da linha de produtos Reunir e formalizar a informação sobre domínio Analisar mercado e enquadrar iniciativa

Papéis envolvidos	Artefactos a construir
Analista de mercado Analista técnico de domínio	Especificação de Características Modelo de mercado Modelo de domínio

3.3.1.1 Descrição

A disciplina de "Análise de Domínio" está segmentada em dois componentes, o estudo técnico do domínio e a análise de mercado. O primeiro reúne e organiza a informação técnica necessária para definir a linha de produtos e o segundo estuda as condições do mercado em que a linha de produtos se vai inserir.

É importante enquadrar devidamente a participação de elementos externos à organização. Uma linha de produtos é um projecto que visa um tipo de cliente e não um cliente em particular, logo, a participação de um cliente específico pode não ter um enquadramento evidente (pelo menos na sua óptica). Por isso mesmo, o fluxo definido começa por definir uma actividade comum às duas equipas para "Identificação e conciliação de estratégias de análise" no sentido de garantir o funcionamento integrado e minimizar o esforço e recursos envolvidos.

O subprocesso de "Análise de Mercado" não será muito explorado neste trabalho pois, por um lado, extrapola o seu âmbito e, por outro, depende bastante do domínio/mercado que estamos a considerar. Resumidamente, o modelo de mercado, artefacto visado neste subprocesso, terá de fornecer informações sobre os tipos de clientes existentes no domínio, produtos que visem o mesmo mercado e estatísticas sobre tendências existentes.

O "Estudo técnico de domínio" é o subprocesso responsável por recolher informação do domínio focada na implementação da linha de produtos. O modelo de domínio é o artefacto responsável por estruturar a informação recolhida e terá obrigatoriamente de conter três aspectos: um glossário com termos de domínio, a identificação e caracterização das entidades, casos de uso, e a descrição de aplicações existentes (próprias ou de terceiros) aplicáveis ao domínio. O artefacto "Especificação de Características" será refinado ao longo da definição da Linha de Produtos, mas nesta fase deverá apenas conter um modelo de características constituído por declarações de necessidades expressas pelos especialistas consultados, especificadas num alto nível de abstracção sem considerar qualquer preocupação sobre a sua exequibilidade.

3.3.2 REQUISITOS

SPLUP – Requisitos
Tagline
A disciplina de Requisitos reúne a responsabilidade de restringir o problema que a LP deve endereçar e traduzir a sua solução num conjunto de requisitos.
Fluxo de Trabalho
Ver Anexo – Fluxos de Trabalho SPLUP

Objectivos	
Definir os limites do problema a endereçar Especificar a linha de produtos quanto às suas características Identificar requisitos da linha de produtos e processo de produção	
Papéis envolvidos	**Artefactos a construir**
Analista de Domínio Engenheiro de Requisitos	Especificação de Características (refinamento) Especificação de requisitos

3.3.2.1 Descrição

A relação entre as actividades de Análise de Domínio e de Requisitos é bastante forte pelo que, na maioria dos casos, podem/devem ser executadas praticamente em simultâneo (o próprio RUP evidencia esse compromisso no gráfico de contribuição de cada disciplina).

Existem dois importantes componentes no fluxo de Requisitos, restringir o espaço de problema e determinar o correspondente espaço de solução.

A delimitação do espaço do problema desenrola-se com base em três artefactos: o Âmbito da Linha de Produtos (a sua elaboração é da responsabilidade do gestor de projecto), o modelo de domínio e o modelo de características. A análise do domínio visa a caracterização total do domínio e suas necessidades. Esta abrangência é na maioria dos casos, impossível de concretizar, pelo menos no horizonte temporal de um projecto, por isso é necessário restringir o problema de acordo com os objectivos e âmbito da linha de produtos. Este confinamento deve ficar especificado num novo modelo de características e adicionado ao artefacto "Especificação de Características" iniciado na Análise de Domínio.

À delimitação do espaço do problema, segue-se a identificação dos requisitos que permitirão construir uma família de sistemas para o endereçar. Este processo deve ser feito em dois níveis de abstracção distintos:

- Descrever a solução em função da definição das suas características.
  Deve ser elaborado um modelo de características da linha de produtos e adicionado ao artefacto "Especificação de Características";
- Descrever a solução em função dos seus requisitos
  Devem ser construídos modelos com base nos requisitos necessários para implementar as características seleccionadas para a linha de produtos. Neste momento é crucial descer para o nível operacional e começar a pensar em função de componentes e processos de produção. Esta informação, que poderá ser expressa em modelos UML, e.g. diagramas "Use Case", ou em matrizes de requisitos deverá constituir um novo artefacto "Especificação de Requisitos";

Um aspecto a realçar é a importância do *traceability*. A declaração das necessidades (modelo de características), a restrição do problema (modelo de características) e a especificação de requisitos (modelos UML ou matrizes) estão relacionados. É de extrema importância manter essa informação para poder "definir soluções em função do problema". Por isso mesmo, há uma dependência mútua entre os dois artefactos respectivos.

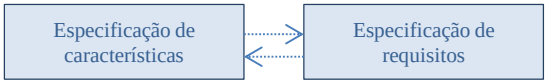

Figura 3-iv – *Traceability* Características - Requisitos

3.3.3 ANÁLISE E DESENHO

SPLUP – Análise e Desenho	
Tagline	
A Análise e Desenho é a disciplina responsável por transformar os requisitos da LP num desenho técnico de componentes que os implementem.	
Fluxo de Trabalho	
Ver Anexo – Fluxos de Trabalho SPLUP	
Objectivos	
Concretizar requisitos no desenho dos componentes que os implementam Assegurar o correcto mapeamento entre os componentes projectados e os requisitos	
Papéis envolvidos	**Artefactos a construir**
Arquitecto de Software	Especificação da Arquitectura da LP Especificação de Componente Especificação do Processo de Produção

3.3.3.1 Descrição

Esta disciplina concentra-se sobre a tradução dos requisitos para o desenho da linha de produtos. Por desenho entenda-se a elaboração do projecto técnico da solução.

A primeira etapa a desenvolver é confrontação dos requisitos com os activos existentes que se apliquem. Esta comparação é baseada no artefacto "Especificação de Reutilização" (que é da responsabilidade do Gestor de Reutilização da disciplina Ambiente) e que contém uma descrição dos requisitos de componentes/sistemas existentes e que estejam de alguma forma relacionados. Este passo é de extrema importância, em particular, quando a organização já opera no domínio em causa e desenvolveu sistemas nessa área.

Com os elementos a reutilizar definidos, o arquitecto de Software deve iniciar o planeamento da abordagem ao desenho dos novos componentes e da, eventual, adaptação dos existentes. Esta planificação envolve especificar em detalhe a implementação de:

- Arquitectura base de todos os membros da família de produtos (construção de uma *framework* ou plataforma aplicacional que agregue os requisitos comuns a todos os elementos e que suporte a extensão e acoplamento de componentes variáveis)
- Componentes reutilizáveis envolvidos para assegurar a implementação dos requisitos variáveis. Devem ser definidos em função da arquitectura base escolhida.
- Processo de produção: definição do processo de produção e desenho dos mecanismos de automatização da linha de montagem (o que inclui desenhar as DSLs e os geradores que os requisitos sugerirem)

O desenho de cada um destes elementos deverá ficar especificado em artefacto próprio. Este artefacto reuniria não só os aspectos individuais de cada elemento mas também o mecanismo e restrições de integração na linha de produtos, e.g. um componente deverá ter explicitamente identificado um modelo de adaptação/integração na linha de produtos.

Mais uma vez é muito importante manter o mapeamento entre elementos de níveis de abstracção diferentes, o que nesta disciplina significa manter a relação entre requisito e desenho do elemento que o implementa. Com esta relação estabelecida, o Arquitecto de Software deve garantir que todos os requisitos são considerados no desenho da solução.

3.3.4 IMPLEMENTAÇÃO

<table>
<tr><th colspan="2">SPLUP – Implementação</th></tr>
<tr><td colspan="2">Tagline</td></tr>
<tr><td colspan="2">Esta disciplina é responsável pela implementação da LP (infra-estrutura e processo de produção) conforme definido pelo Arquitecto de Software.</td></tr>
<tr><td colspan="2">Fluxo de Trabalho</td></tr>
<tr><td colspan="2">Ver Anexo – Fluxos de Trabalho SPLUP</td></tr>
<tr><td colspan="2">Objectivos</td></tr>
<tr><td colspan="2">Implementar as especificações da plataforma, componentes e processo de produção
Extrair a adaptar componentes existentes
Integrar todos os itens envolvidos</td></tr>
<tr><td>Papéis envolvidos</td><td>Artefactos a construir</td></tr>
<tr><td>Developer
Tool Developer
DSL Developer</td><td>Plataforma comum
Componentes
Processo de Produção</td></tr>
</table>

3.3.4.1 Descrição

A disciplina de implementação é responsável por transformar a especificação da linha de produtos numa infra-estrutura de produção operacional, o que se traduz na implementação da sua plataforma comum, componentes variáveis e processo de produção de acordo com a análise e desenho efectuada.

Para além do inevitável desenvolvimento integral de alguns elementos especificados, será comum, dependendo da análise e desenho, esta disciplina envolver reutilização, nomeadamente:

- Extracção de componentes de sistemas existentes: na maior parte dos casos, só a simples extracção torna-se complexa pois o desenvolvimento não é suficientemente modularizado.
- Adaptação de componentes: a organização pode ter desenvolvido ou adquirido componentes para utilizar em sistemas anteriores. A capitalização desse esforço constituirá uma possível opção viável ao desenvolvimento integral.

A arquitectura comum deve ser cuidadosamente implementada, garantindo que disponibiliza os mecanismos de variação adequados para integrar os componentes desenvolvidos. Ainda que não seja da directa responsabilidade da equipa de desenvolvimento, (existe uma equipa de Testes) durante a implementação, o programador deverá efectuar pequenos testes de integração para garantir que o que foi planeado tem os resultados esperados. Em situações em que isso não aconteça, o problema deve ser rapidamente reportado ao Arquitecto de Software que deverá analisar a situação e propor nova solução.

O processo de produção é uma parte crucial do esforço de implementação que garantirá que as "partes funcionarão como um todo". Neste domínio, existem duas grandes áreas de actuação:

- DSLs e Geradores: as linguagens específicas de domínio estabelecerão a forma de especificar as aplicações a construir. Deve por isso garantir-se que cobrem todas as características (comuns ou variáveis) da Linha de Produtos e que estas podem ser expressas num nível de abstracção adequado. O desenvolvimento destas linguagens deve ser realizado em conjunto com a construção de geradores específicos.
- Adaptar/Criar Ferramentas de desenvolvimento: Envolve, por exemplo, aplicações para manipulação de DSLs ou adaptações e extensões de IDEs para *guidance* de produção.

Idealmente, toda a lógica de apoio à produção deveria ficar reflectida nas ferramentas pois permitiria uma validação/acompanhamento dinâmico durante o processo de produção. Todavia, será natural que exista conteúdo de apoio que possa não ser formalizado a esse ponto. Nestes casos, a equipa de desenvolvimento deverá trabalhar em directa articulação com a equipa de Ambiente para garantir que o conhecimento não é perdido.

3.3.5 TESTES

<table>
<tr><th colspan="2">SPLUP – Testes</th></tr>
<tr><td colspan="2">Tagline</td></tr>
<tr><td colspan="2">A disciplina de Testes visa a certificação do funcionamento esperado e adequado dos componentes desenvolvidos ou ferramentas adaptadas</td></tr>
<tr><td colspan="2">Fluxo de Trabalho</td></tr>
<tr><td colspan="2">Ver Anexo – Fluxos de Trabalho SPLUP</td></tr>
<tr><td colspan="2">Objectivos</td></tr>
<tr><td colspan="2">Testar linha de produtos (plataforma, componentes e processo)
Testar ferramentas de apoio à produção
Testar aplicações de exemplo (instanciações da linha de produtos)</td></tr>
<tr><td>Papéis envolvidos</td><td>Artefactos a construir</td></tr>
<tr><td>Tester
Tool Tester
DSL Tester</td><td>Plano de teste</td></tr>
</table>

3.3.5.1 Descrição

As actividades realizadas no âmbito da disciplina de Testes centram-se sobre a avaliação dos resultados do trabalho realizado, i.e. testar a linha de produtos e respectivo processo de produção.

As perguntas tradicionais e genéricas de qualquer fase de Testes também se aplicam neste modelo: *As especificações foram cumpridas? Se não, existem razões documentadas para que assim fosse? O funcionamento é adequado e sem bugs? Responde aos padrões de qualidade definidos?* A diferença é que estas questões são colocadas a uma linha de produtos e não a uma aplicação específica. No contexto multi-sistema, os testes devem ser segmentados e responder às seguintes questões:

- Teste à plataforma: *A plataforma implementa as características comuns? Suporta o correcto acoplamento de componentes variáveis? Cumpre requisitos funcionais e não funcionais? Suporta a extensibilidade necessária para novos contextos? O esforço de manutenção é realista? Suporta configuração da arquitectura, e.g. afinação de parâmetros de performance?*
- Testes a componentes: *Cada componente tem uma especificação de produção associada que facilite o seu consumo? A funcionalidade está bem descrita? Tem o comportamento indicado? Respeita o requisito que lhe deu origem?*
- Teste a linguagens: *As linguagens têm a expressividade suficiente? Estão definidas num grau de abstracção adequado? São integráveis, i.e. as linguagens que dizem respeito ao mesmo aspecto do sistema mas expressam perspectivas diferentes são passiveis de interpretação conjunta? As regras de manipulação de abstracções são demasiado rígidas? Reflectem adequadamente o domínio?*
- Testes a geradores: *Qual a qualidade da informação/código gerado? Permitem a correcta integração entre plataforma, componentes e linguagem? Existe potencial de geração que está subaproveitado?*
- Testes a ferramentas: *As ferramentas apresentam facilidade de utilização adequada? O processo de produção está correctamente traduzido na aplicação? As extensões/adaptações funcionam correctamente? Existe potencial de apoio à produção que não esteja a ser considerado?*

Nem todas as questões são aplicáveis a todos os contextos e certamente dependem das características de cada linha de produto. Porém, observando as diferentes abordagens que requerem, a segmentação dos testes deverá ser sempre seguida.

Cada componente terá uma "Especificação de testes" onde serão planeados os testes a realizar e descritos os resultados obtidos. No final dos testes individuais, deverão ser executados testes a aplicações de exemplo. Dada uma especificação expressa através das linguagens criadas para o efeito, a equipa de testes deverá produzir a corresponde aplicação, testá-la e verificar se está de acordo com o esperado.

3.3.6 DEPLOYMENT

SPLUP – *Deployment*	
Tagline	
Este disciplina visa garantir a correcta passagem da infra-estrutura de produção para a equipa da engenharia de aplicações.	
Fluxo de Trabalho	
Ver Anexo – Fluxos de Trabalho SPLUP	
Objectivos	
Assegurar disponibilidade da infra-estrutura de produção Garantir correcto funcionamento em ambiente de produção	
Papéis envolvidos	**Artefactos a construir**
Engenheiro de suporte	Manual de utilização de infra-estrutura de produção

3.3.6.1 Descrição

O *deployment* é a disciplina que se encarrega da correcta passagem da infra-estrutura de produção para a engenharia de aplicação. O seu objectivo primordial é garantir que a equipa da engenharia de aplicações tem o conhecimento e ferramentas para produzir aplicações dentro da família de produtos existente.

O ambiente de desenvolvimento da linha de produtos não é obrigatoriamente o mesmo do ambiente de desenvolvimento de aplicações. Nesses casos, é necessário assegurar que a infra-estrutura se comporta adequadamente no novo cenário. Nesse sentido, a equipa de *deployment* deve garantir o correcto funcionamento de todos os elementos em ambiente de produção (engenharia de aplicação). Para além do aspecto técnico da transição deve também ser garantido que as pessoas responsáveis se adaptam e sabem actuar no novo contexto. Esta componente do *deploy* deve ser realizada em paralelo com as acções de treino e formação proporcionadas pela equipa do Ambiente e deverá passar pela disponibilização de documentação de suporte, nomeadamente, manuais de utilização da infra-estrutura de produção devidamente adaptados ao perfil das pessoas que o vão utilizar.

3.3.7 GESTÃO DE PROJECTO

SPLUP – Gestão de Projecto
Tagline
Esta disciplina visa garantir as condições necessárias para a correcta execução do processo de construção da LP, bem como, orquestrar os processos de engª aplicação.
Fluxo de Trabalho
Ver Anexo – Fluxos de Trabalho SPLUP
Objectivos
Definir e afinar âmbito da LP e do Projecto Garantir recursos necessários ao projecto Monitorizar e controlar a execução do projecto

Analisar opções de compra vs desenvolvimento Garantir correcta execução dos processo de engenharia de Aplicação	
Papéis envolvidos	**Artefactos a construir**
Gestor de Projecto LP Gestor de Projecto Aplicação	Especificação da LP Plano de Projecto Plano de Produção de Aplicação

3.3.7.1 Descrição

A Gestão de Projecto é uma das três disciplinas que é partilhada com a engenharia de Aplicação. Assim, esta disciplina engloba, não só a gestão das actividades que respeitam à construção da Linha de Produtos, mas também regula a gestão dos projectos para produção de Aplicações. Concretizando, a disciplina tem três vertentes principais:

- Gestão do projecto da LP
- Gestão técnica da LP
- Gestão de projectos de Aplicação

A gestão do projecto da LP engloba tarefas de planeamento e monitorização da execução do projecto. O Gestor de Projecto deve iniciar o projecto pela preparação do projecto, isto inclui estimar os recursos necessários, garantir a sua disponibilidade ou definir um plano de trabalho. O longo prazo do projecto dificulta alguns destes aspectos, que por tradição não são simples. Cumulativamente, o gestor do projecto deve garantir a administração correcta das iterações, planeando-as, segmentando os objectivos do projecto em objectivos de iteração, definindo métricas adequadas e controlando se são atingidos.

Para além da gestão operacional do projecto, a disciplina Gestão de Projecto tem uma participação directa em opções técnicas da linha de produtos:

- Definição do âmbito da LP: É o Gestor de Projecto em articulação com a equipa de Analise de Domínio e Requisitos que deve fixar os limites específicos do domínio e estabelecer as fronteiras da LP;
- Análise Compra vs. Desenvolvimento: O Gestor de Projecto, em directa articulação com as equipas de "Análise e Desenho" e "Ambiente", deve avaliar a opção de desenvolvimento interno ou compra de componentes/ ferramentas;
- Alinhamento com objectivos organizacionais: É o Gestor de Projecto que deve garantir se os objectivos da Linha de Produtos estão alinhados com os objectivos organizacionais e se o patrocínio de topo se mantém durante todo o projecto.

Na fronteira da gestão de projecto e da gestão técnica que o Gestor de Projecto deve realizar, este também deve sempre garantir que o projecto reúne as condições necessárias para continuar. Factores como indisponibilidade de recursos ou desalinhamento com objectivos organizacionais devem constituir fortes motivadores para o abandono. Por isso é essencial que durante a execução do projecto, o Gestor de Projecto promova debates para verificar a viabilidade e vantagens/desvantagens da continuidade do projecto.

A Gestão de projecto de Aplicação é uma das "pontes" que o SPLUP define para integrar a construção da infra-estrutura e a produção das aplicações. O gestor de projecto da LP deve

fazer uma primeira triagem em que exclua as solicitações que não estejam minimamente enquadradas com a LP. Após essa triagem, o Gestor de Projecto LP deve preparar um novo projecto de Aplicação e nomear alguém (Gestor de Projecto de Aplicação) para o gerir. A partir desse momento a participação do Gestor de Projecto LP deverá ser pontual e restrita à definição do âmbito, sendo a gestão operacional do Projecto de Aplicação uma responsabilidade do Gestor de Projecto nomeado.

A gestão de Projecto de Aplicação é muito simples pois a grande maioria dos aspectos será definido pela LP. Para além do trabalho operacional, há dois aspectos bastante pertinentes:

- A formalização do feedback sobre a utilização da linha de produtos
- A interacção com a equipa de gestão de mudança para solicitar desenvolvimentos adicionais

O Gestor de Projecto de Aplicação não deve, em qualquer circunstância, interferir na Gestão do Projecto LP; o oposto pode acontecer, embora deva ser evitado.

3.3.8 GESTÃO DE CONFIGURAÇÕES E MUDANÇA

SPLUP – Gestão de Configurações e Mudança	
Tagline	
Esta disciplina concentra as responsabilidades relativas à gestão de configurações dos itens criados e a gestão das solicitações de alterações à LP.	
Fluxo de Trabalho	
Ver Anexo – Fluxos de Trabalho SPLUP	
Objectivos	
Assegurar adequação de políticas às necessidades do Programa Garantir que o histórico de configurações de todos os itens é mantido Certificar o correcto tratamento de todas as solicitações de alterações	
Papéis envolvidos	**Artefactos a construir**
Gestor de Configurações Gestor de Mudança Comité de Mudança	Declaração de Políticas de Config. e Mudança *Change Tracking*

3.3.8.1 Descrição

À semelhança da Gestão de Projecto e Ambiente, a disciplina de "Gestão de configurações e Mudança" é partilhada com a engenharia de aplicação.

A gestão de configurações é responsável por garantir a administração das versões e configurações de todos os activos, o que inclui:

- Para a engenharia de domínio: artefactos de especificação, elementos implementados (plataforma e componentes), adaptações e extensões de ferramentas, linguagens e geradores criados
- Para a engenharia de aplicação: aplicações produzidas

Pelos exemplos enumerados fica claro que pode ser bastante complexo fazer a gestão de configurações de uma linha de produtos. Há um ponto que deve servir de guia para assegurar

que se está fazer uma gestão de configurações eficiente: a linha de produtos deve sempre garantir que consegue produzir novamente uma aplicação que tenha produzido no passado. É, por isso, essencial manter as várias versões de todos os activos envolvidos, bem como, a configuração que esses activos foram sujeitos para produzir aplicações.

Outro aspecto importante desta disciplina é gestão de mudança. O esforço associado a uma mudança numa LP pode ser consideravelmente elevado por isso, o SPLUP contempla a criação de um "Comité de Mudança" constituído por representantes de todas as disciplinas para avaliar as solicitações de alterações ou novos desenvolvimentos. Essas solicitações tanto poderão vir da equipa de construção da LP como da equipa da engenharia de aplicações. No primeiro caso, as solicitações reflectem necessidades do domínio que não foram correctamente analisadas ou erros de análise que são detectados na execução do projecto. No segundo, as solicitações resultam directamente de necessidades expressas por um cliente que não estejam consideradas na linha de produtos ou indirectamente pelo feedback da equipa de produção sobre a utilização da infra-estrutura da LP.

3.3.9 AMBIENTE

SPLUP – Ambiente	
Tagline	
A disciplina de Ambiente configura e fornece apoio ao processos SPLUP bem como assegura que não há desenvolvimento desnecessário promovendo a reutilização.	
Fluxo de Trabalho	
Ver Anexo – Fluxos de Trabalho SPLUP	
Objectivos	
Garantir que a equipa de domínio segue o modelo de desenvolvimento Assegurar que o processo de produção de aplicações é correctamente executado Dar apoio aos dois processos Promover a melhoria contínua	
Papéis envolvidos	**Artefactos a construir**
Gestor de reutilização Engenheiro de processo Especialista em ferramentas	Relatório técnico de processo Relatório técnico de ferramentas Relatório de gestão de reutilização

3.3.9.1 Descrição

No SPLUP o objectivo da disciplina Ambiente (assegurar o processo e as ferramentas necessários para criar um ambiente de desenvolvimento apropriado) é materializado da seguinte forma:

- Relativamente à engenharia de domínio:
  - Certificar a adequação entre o processo de desenvolvimento e as condições existentes
  - Garantir que as equipas conhecem o processo de desenvolvimento e que dispõem das ferramentas necessárias para o desempenhar

    - Maximizar o potencial de reutilização, intervindo com esse propósito específico.
- Relativamente à engenharia de aplicação:
    - Assegurar que o processo de produção está ser cumprido certificando-se que a equipa de engenharia de aplicação detém o conhecimento necessário.

O modelo propõe que a gestão da reutilização fique associada um papel específico que tenha a responsabilidade de monitorizar o desenvolvimento procurando alvos de reutilização. Nessas situações, o gestor de reutilização deverá preparar uma proposta especificando as capacidades e enquadramento do activo existente e solicitar a sua consideração no processo de desenvolvimento. A participação do gestor de reutilização não se deve cingir à fase de desenvolvimento, pelo contrário, deve estar presente em todo o processo, promovendo a reutilização de especificações de requisitos, desenho de componentes, casos de teste ou qualquer outro artefacto.

Para além da promoção da perspectiva de consumo de activos existentes, o gestor de reutilização é responsável por gerir o Repositório de activos. Todos os artefactos deverão ser devidamente arquivados nesse repositório para que possam ser reutilizados no futuro. O fundamento do SPLUP para atribuir essa responsabilidade a um papel específico e não ao autor de cada artefacto, nasce da habitual dificuldade de cumprimento "distribuído" de normas de versionamento e, em particular, de convenções de descrição de artefactos, o que inevitavelmente resulta num repositório inutilizável.

3.4 SPLUP – ENGENHARIA DE APLICAÇÃO

O processo de Engenharia de Aplicação da SPLUP está organizado em 4 disciplinas:

1. Análise do Problema
2. Implementação Adicional
3. Produção e Testes
4. *Deployment* da Aplicação

O objectivo central destas disciplinas é organizar o trabalho necessário para construir aplicações com base na infra-estrutura de produção disponibilizada pela Engenharia de Domínio.

À semelhança da Engenharia de domínio, este processo também está organizado em quatro fases, cada uma das quais com um macro objectivo associado (apresentados na Tabela 3-ii) que deve definir se a fase está, ou não, concluída.

Fase	**Principal objectivo**
Concepção	Assegurar com o Cliente a correcta análise do problema
Elaboração	Garantir a correcta caracterização do produto em função das necessidades do Cliente e dos objectivos da LP
Construção	Efectuar eventual desenvolvimento adicional ao sistema produzido
Transição	Garantir a correcta disponibilização do sistema aos utilizadores finais

Tabela 3-ii – *Milestones* Engenharia de domínio SPLUP

De seguida apresentamos em detalhe as referidas disciplinas.

3.4.1 ANÁLISE DO PROBLEMA

SPLUP – Análise do Problema	
Tagline	
Disciplina responsável por analisar o problema a endereçar pela aplicação e enquadrar a solução no âmbito da Linha de Produtos.	
Fluxo de Trabalho	
Ver Anexo – Fluxos de Trabalho SPLUP	
Objectivos	
Conseguir enquadrar a LP nas necessidades do Cliente Especificar produto Fornecer feedback sobre LP	
Papéis envolvidos	**Artefactos a construir**
Cliente Analista de Negócio Gestor de Aplicação Comité de Mudança	Especificação de produto Avaliação de LP

3.4.1.1 Descrição

A análise do problema é a primeira disciplina do processo de engenharia de Aplicação do SPLUP, sendo o seu principal objectivo alinhar as características do problema do cliente com as características da linha de produtos.

A análise do problema deve ser feita entre o Cliente e um Analista de Negócio. O Analista de Negócio deve negociar a solução e orientá-la para as características disponibilizadas pela Linha de Produtos e assim evitar o desenvolvimento adicional específico. Haverá, naturalmente, casos em que isso é impossível, todavia, existirão outras situações em que o argumento do acréscimo de tempo/custo de desenvolvimento necessário poderá favorecer a exclusão dessa via.

A infra-estrutura de produção deve ser utilizada desde cedo para gerar protótipos e auxiliar na validação da especificação problema. Como os modelos utilizados para gerar os protótipos são especificados com abstracções próximas do negócio serão facilmente entendidas pelo Cliente por isso também podem contribuir para essa validação.

Com a análise do problema concretizada, o analista de negócio deverá reunir num modelo as características da solução encontrada e preparar um documento com o feedback da LP sobre a sua utilização e opinião do Cliente.

3.4.2 IMPLEMENTAÇÃO ADICIONAL

SPLUP – Implementação Adicional	
Tagline	
A disciplina Implementação Adicional é responsável por implementar pequenas extensões e componentes que tenham sido autorizados pelo Comité de Mudança	
Fluxo de Trabalho	
Ver Anexo – Fluxos de Trabalho SPLUP	
Objectivos	
Implementar componentes adicionais	
Papéis envolvidos	**Artefactos a construir**
Developer Aplicação	Especificação de Componente

3.4.2.1 Descrição

O fluxo de trabalho considerado por esta disciplina apenas ocorre em situações em que a parte da solução para o problema do cliente não está incluído na Linha de Produtos mas existe interesse em avançar com o seu desenvolvimento. Essa motivação pode estar relacionada com futuras reutilizações do novo componente ou, por exemplo, com a importância desse adicional para conseguir a satisfação do Cliente.

O fluxo é simples e a sua execução não deverá exceder uma semana. O procedimento inicia-se com a análise da Especificação do Produto (onde está descrito o componente a

desenvolver) e com a avaliação dos mecanismos de extensão disponibilizados pela plataforma comum e estudo de componentes com algumas semelhanças que possam já existir.

À preparação da abordagem segue-se a implementação do componente com definição da respectiva especificação em artefacto próprio e a realização de uma bateria de testes para certificar o seu correcto funcionamento (individual e integrado na plataforma).

3.4.3 PRODUÇÃO E TESTES

SPLUP – Produção e Testes	
Tagline	
Produção e testes é a disciplina responsável pela produção da aplicação e verificação do seu correcto funcionamento.	
Fluxo de Trabalho	
Ver Anexo – Fluxos de Trabalho SPLUP	
Objectivos	
Construir o modelo da aplicação Produzir a aplicação com base no plano de produção Garantir o correcto funcionamento da aplicação	
Papéis envolvidos	**Artefactos a construir**
Developer Aplicação	Relatório de testes de aplicação

3.4.3.1 Descrição

A disciplina de Implementação e Testes é responsável por executar o plano de produção da Aplicação e testar o sistema em ambiente de produção (Cliente).

A produção da aplicação é totalmente definida pelo plano de produção. Independentemente disso, o processo passará por criar modelos através de DSL que reflictam as características que o sistema deve implementar e concretizar - de forma totalmente manual, automática ou mista - o processo de montagem de componentes. O fluxo de implementação termina com a bateria de testes que o plano de produção definir para as aplicações geradas.

3.4.4 DEPLOYMENT DA APLICAÇÃO

SPLUP – Deployment da Aplicação
Tagline
A disciplina *Deployment* da aplicação concentra-se na correcta disponibilização da aplicação ao cliente
Fluxo de Trabalho
Ver Anexo – Fluxos de Trabalho SPLUP
Objectivos
Construir package de instalação

Instalar e testar aplicação em ambiente de produção	
Papéis envolvidos	**Artefactos a construir**
Engenheiro de Suporte	Relatório de instalação de aplicação

3.4.4.1 Descrição

O *deployment* da aplicação consiste na disponibilização da aplicação ao cliente. A primeira tarefa a realizar é a criação de ficheiros de instalação e a preparação de documentação de suporte à aplicação. Naturalmente que este passo só se aplica quando infra-estrutura de produção não os criar automaticamente.

Para além dos testes genéricos definidos no plano de produção para teste a qualquer aplicação gerada, é crucial verificar o correcto funcionamento das aplicações no ambiente de produção (Cliente).

4 AVALIAÇÃO DA PROPOSTA

4.1 PRESSUPOSTOS

Neste capítulo pretendemos estudar um caso de aplicação do modelo SPLUP. Esta análise terá duas vertentes complementares:

- Validação dos fluxos de trabalho e actividades contempladas no modelo SPLUP;
- Validação prática da articulação de tecnologias incorporadas no modelo SPLUP e que foram referidas ao longo trabalho;

Estas duas vertentes foram concretizadas através da instanciação do modelo SPLUP num cenário fictício (o mais realista possível) conjuntamente com a construção de um protótipo para uma infra-estrutura de produção enquadrada nesse cenário.

Como já foi referido, as disciplinas Gestão de Projecto, Gestão de Configurações e Mudança e Ambiente dão um apoio transversal ao processo de desenvolvimento pelo que, para melhor esclarecer o desdobramento desse contributo genérico, considerámos mais interessante enquadrar as suas actividades ao longo da exposição de cada uma das restantes disciplinas.

4.2 ENQUADRAMENTO

A empresa ITCORP, existente desde 1993, é uma organização portuguesa, a operar exclusivamente em Portugal, que se dedica ao desenvolvimento de sistemas de informação vocacionados para a área do ensino. Entre as dezenas de projectos que realizou no passado destacam-se, pela representatividade que mantêm, os seguintes sectores de actuação:

- Projectos de desenvolvimento de SI para a gestão de colégios
- Projectos de desenvolvimento de SI para estabelecimentos de ensino secundário e ensino superior
- Projectos de desenvolvimento de SI para gestão de bibliotecas
- Projectos de desenvolvimento de SI e conteúdos para ensino à distância (*e-learning*)

A ITCORP tem uma situação financeira estável e conta, actualmente, com mais de 70 funcionários altamente motivados. Um dos valores mais importantes para a organização é qualidade dos serviços e dos produtos que fornecem. A maturidade dos processos internos e as variadas certificações que reúnem são a prova dessa postura.

No sentido de se manter sempre actualizada, a empresa tem vindo analisar o potencial de enquadramento das linhas de produtos na sua actividade. O extenso conhecimento que detém sobre os domínios onde actuam e a elevada experiência da equipa de desenvolvimento são dois aspectos que orientaram a organização para avançar com uma prova de conceito.

A escolha da área para o projecto-piloto foi simples. Desde há 2 anos, que a ITCORP tem procurado uma solução para reduzir o elevado esforço de manutenção das dezenas de sistemas de informação que desenvolveu para a gestão de estabelecimentos de ensino superior pois algumas aplicações ainda são em formato desktop e estão instaladas em máquinas dispersas

por todo o país. Estes sistemas de informação foram desenvolvidos "à medida" de acordo com as necessidades de cada cliente mas, a primeira opinião da generalidade da equipa, é que existem inúmeras semelhanças ao nível da funcionalidade que apresentam.

Depois da análise das alternativas, a metodologia escolhida foi o SPLUP por ter muitos paralelismos com o método que seguem actualmente, o RUP. Deu-se então a nomeação de um Gestor de Projecto e arranque formal da iniciativa.

4.3 PROJECTO SPLUP

4.3.1 ENGENHARIA DE DOMÍNIO

4.3.1.1 Análise de domínio

A indicação prévia da área de actuação da linha de produtos estava estabelecida: a gestão de informação de estabelecimentos de ensino superior.

A análise de mercado revelou aquilo que era esperado pela organização: a implementação do processo de Bolonha trouxe muitas alterações que ainda estão a decorrer, como por exemplo, a criação de novos cursos e planos curriculares, o aumento de alunos nos 2ºs e 3ºs ciclos ou transformações nos modelos de financiamento das instituições. A conclusão da observação desse contexto foi a constatação da existência de margem de progressão dos produtos assegurada para os próximos anos. Por outro lado, a análise de mercado revelou que poucas organizações se dedicam em exclusivo a esta área e as que o fazem não têm a experiência da ITCORP. Ainda assim, foram encontrados no mercado três produtos que se destinam a este campo. Esta informação foi formalizada no artefacto Modelo de Mercado.

Graças aos vários anos de experiência da organização na área alvo, o estudo técnico de domínio desenrolou-se, quase em exclusivo, com especialistas internos. O jargão, o modelo e processos de negócio, as necessidades típicas são bem conhecidos pela ITCORP pelo que a tarefa de análise foi tomada em grande parte pela modelação e formalização dessa informação. Uma componente importante desse trabalho foi a generalização de conceitos e a criação de abstracções sobre as necessidades específicas que estão projectadas nas aplicações analisadas (desenvolvidas pela ITCORP ou concorrentes no mercado). O resultado deste estudo ficou reflectido nos artefactos Modelo de Domínio e Especificação de Características da LP. O diagrama de características de primeiro nível incluído no Modelo de Domínio é apresentado na Figura 4-i.

Gestão de Projecto	Determinação de âmbito do Projecto Identificação de riscos Planeamento e acompanhamento de iterações
Gestão de Config.s e Mudança	Atribuição de versões a artefactos criados
Ambiente	Promoção da melhoria contínua Preparação da organização para o processo SPLUP Classificação e arquivo de artefactos criados

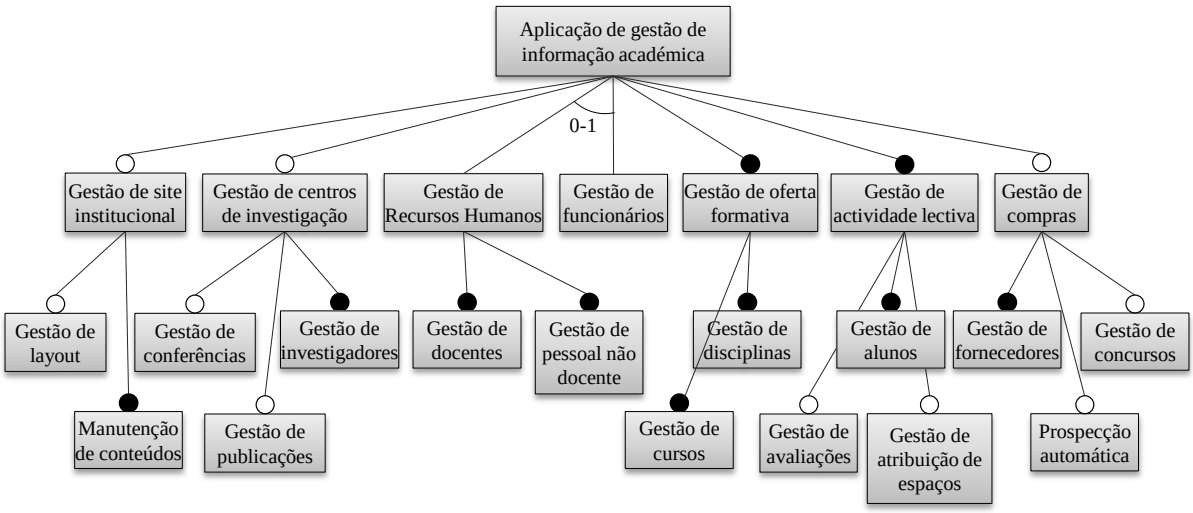

Figura 4-i – Modelo de Características incluído no Modelo

4.3.1.2 Requisitos

A equipa de requisitos começou por analisar os Modelos de Mercado e de Domínio para se inteirar do contexto. De seguida, em directa articulação com o Gestor do Projecto, iniciou a análise das características da LP e a definição do espaço do problema a endereçar.

O carácter piloto deste projecto incentivava a definição de um âmbito bastante restrito para assegurar a sua concretização mas ao mesmo tempo suficientemente abrangente para ser demonstrador do potencial. Foi então decidido que a linha de produtos se debruçaria sobre a gestão de alunos. Várias foram as justificações para esta opção:

- O aluno é uma entidade central e a sua gestão é um aspecto muito importante para o negócio;
- O público-alvo das aplicações (professores) tinha dimensão suficiente para ter uma avaliação e opinião representativa da qualidade da solução;
- Embora não fizesse parte da intenção do projecto, seria um domínio que poderia ser reutilizado noutras LP (e.g. ensino secundário).

Feita a restrição do problema e actualizado o documento de Especificação de Características da LP, o Engenheiro de Requisitos, acompanhado do analista que participou no estudo técnico de domínio, iniciou a definição dos requisitos. A família de aplicações seria muito simples e traduzir-se-ia nos seguintes requisitos de alto nível:

- Inserir, remover e listar alunos (existentes numa base de dados);
- Filtrar resultados de listagens;
- Exportar listagens para ficheiros Excel;
- Personalizar parte estética da aplicação (nesta fase só o logótipo);
- Escolher os atributos do Aluno disponíveis para cada aplicação.

Para melhor clarificar a variabilidade incluída nos requisitos, o analista de domínio, juntamente com o engenheiro de requisitos, efectuaram um modelo de características para a família de aplicações de gestão de alunos.

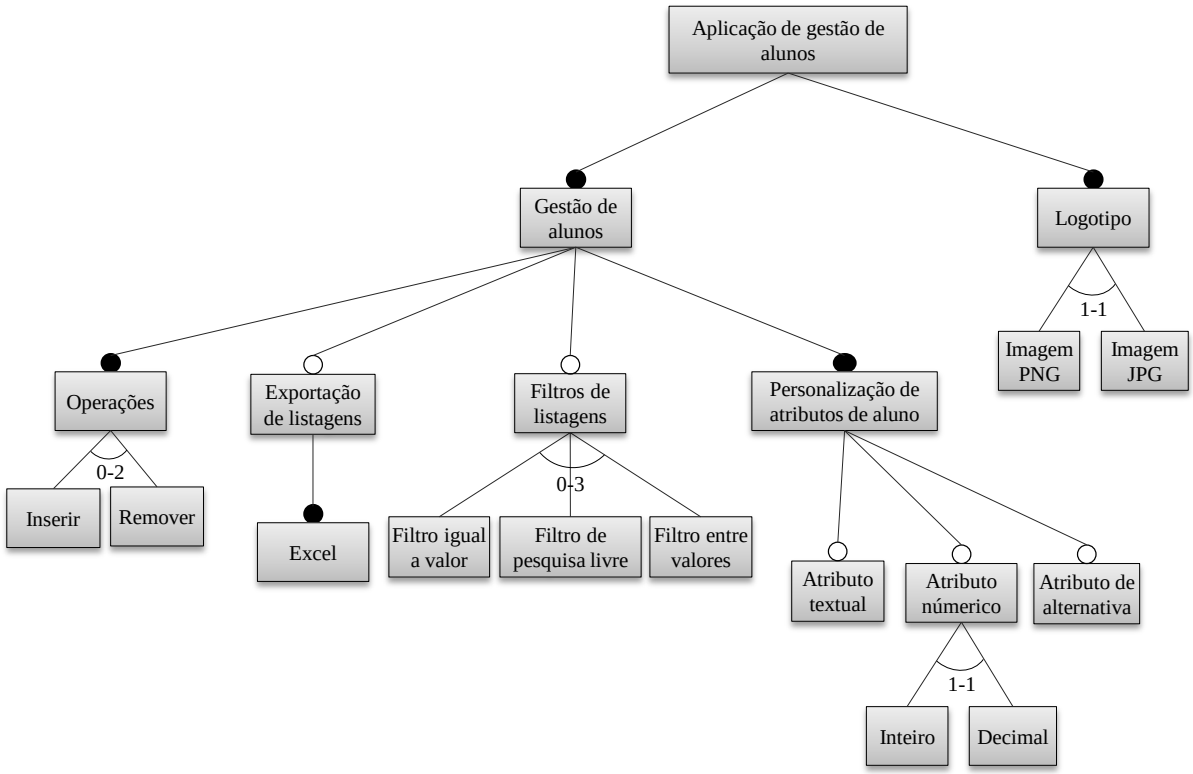

Figura 4-ii – Modelo de características da família de aplicações de gestão de alunos

Este modelo e restante informação sobre requisitos ficou documentada no artefacto "Especificação de Requisitos".

Gestão de Projecto	Delimitação do problema e refinamento do âmbito da LP Monitorização e controlo Planeamento e acompanhamento de iterações
Gestão de Config.s e Mudança	Atribuição de versões a artefactos criados
Ambiente	Promoção da melhoria contínua Preparação da organização para o processo SPLUP Classificação e arquivo de artefactos criados Apresentação de propostas de reutilização de use cases de sistemas existentes

4.3.1.3 Análise Desenho

O Arquitecto de Software nomeado para o projecto, em conjunto com o Gestor de Projecto, começou por identificar as opções tecnológicas para o projecto:

- A família de sistemas assentaria num modelo cliente-servidor com acesso pela internet (aplicações web);

- A arquitectura comum da família de sistemas seria baseada numa *framework* orientada a objectos (doravante designada por *framework*) pois a ITCORP tinha experiência de desenvolvimento nessa área;
- A linguagem de programação utilizada seria C# (Microsoft Framework .Net v4.0) em conjunto com a tecnologia Silverlight para interfaces de utilizador.
- A persistência dos dados seria feita em base de dados Microsoft SQL Server 2008 e deveria funcionar em modo (camada) isolado por web services.
- O IDE de apoio ao desenvolvimento da Linha de Produtos e à produção das aplicações seria o Microsoft Visual Studio 2010 com respectivo SDK para configurações e extensões avançadas;
- Para construção de DSLs e geradores seria utilizado o Microsoft DSL Tools (plugin de Visual Studio que inclui as ferramentas de modelação e infra-estrutura T4 para geração de código);

De seguida, com base nas opções tecnológicas e nos requisitos anteriormente definidos, o Arquitecto de Software iniciou uma análise dos sistemas e componentes propostos para reutilização e identificou um conjunto de reutilizações para minimizar o desenvolvimento integral. Esta informação foi integrada no desenho da *framework* e componentes que se seguiu.

Em paralelo com a definição da *framework* e componentes, o Arquitecto, com participação do Gestor de Projecto, definiu a infra-estrutura de produção com um objectivo ambicioso. Em situações em que os componentes estivessem desenvolvidos, i.e. em situações em que não existissem requisitos do problema que não estivessem cobertos pela linha de produtos, a geração da aplicação deveria ser totalmente automática. Só em casos em que houvesse implementação de componentes específicos é que o processo de produção da aplicação seria semiautomático. Com esse objectivo em mente e baseando-se no modelo de Domínio e modelo de Características da LP, iniciou a especificação das principais abstracções a serem consideradas numa DSL destinada à modelação dos sistemas, bem como, o desenho do gerador respectivo. A construção da DSL e gerador bastava para a implementação do processo de produção não sendo necessário nenhum desenvolvimento de adaptação de IDEs.

O Arquitecto deu por terminado os trabalhos com a verificação de que os requisitos haviam sido inteiramente considerados nos elementos especificados. Como *output* final, o Arquitecto apresentou à equipa de implementação os artefactos de especificação da *framework*, componentes e processo de produção (DSL e gerador).

Gestão de Projecto	Realização de análises de compra vs desenvolvimento Acordar ajustes no âmbito da LP Monitorização e controlo Planeamento e acompanhamento de iterações
Gestão de Config.s e Mudança	Atribuição de versões a artefactos criados

Ambiente	Avaliação de ferramentas Promoção da melhoria contínua Classificação e arquivo de artefactos criados Apresentação de propostas de reutilização de componentes Apresentação de propostas de reutilização de (partes) de sistemas Apresentação de propostas de reutilização de modelos

4.3.1.4 Implementação

A equipa de implementação foi segmentada de acordo com o tipo de elemento a implementar: *framework*, componentes, DSL e gerador.

O maior esforço de desenvolvimento esteve na construção da *framework* e na programação orientada por padrões de desenho que esta requeria. Grande parte dos componentes resultou, conforme especificação, da adaptação/extracção de partes de sistemas existentes. Para garantir o correcto funcionamento entre *framework* e componentes, os dois núcleos da equipa efectuaram regularmente pequenos testes informais de integração.

O desenvolvimento da DSL e gerador também foi exigente devido essencialmente à inexperiência da equipa nestas áreas. Contudo, como alguns membros da equipa já haviam participado em projectos com base *model-driven*, o resultado do trabalho teve a qualidade esperada. Meta-modelo da DSL para modelação das aplicações pode ser consultado no Anexo – Meta-Modelo DSL (ITCorp).

Gestão de Projecto	Monitorização e controlo Planeamento e acompanhamento de iterações
Gestão de Config.s e Mudança	Atribuição de versões a artefactos criados Gestão de versões da framework Gestão de versões de componentes Gestão de versões de geradores Gestão de versões de meta-modelos DSL
Ambiente	Promoção da melhoria contínua Disponibilização correcta de ferramentas Classificação e arquivo de artefactos criados

4.3.1.5 Testes

A equipa de testes começou por alinhar estratégias para garantir que os diversos elementos, que seriam avaliados individualmente, funcionariam correctamente em conjunto. Esse planeamento envolveu a criação de cenários e procedimentos de teste específicos para cada componente e para diversas configurações de integração.

A opção da equipa de Testes foi semelhante à equipa de Implementação: segmentar a equipa de acordo com o tipo de elemento a testar (*framework*, componentes, DSL ou gerador). A inexperiência de desenvolvimento em áreas como as DSL ou geradores foi notada e vários problemas foram detectados nos testes, porém, o rápido reporte da situação à equipa responsável minimizou o impacto das alterações.

Gestão de Projecto	Monitorização e controlo Planeamento e acompanhamento de iterações
Gestão de Config.s e Mudança	Atribuição de versões a artefactos criados Gestão de versões da framework Gestão de versões de componentes Gestão de versões de geradores Gestão de versões de meta-modelos DSL
Ambiente	Promoção da a melhoria contínua Disponibilização correcta de ferramentas Classificação e arquivo de artefactos criados

4.3.1.6 Deployment

Como a decisão de desenho indicava a mesma ferramenta para o desenvolvimento da linha de produtos e para a produção de aplicações, o *deployment* foi simples e a componente técnica resumiu-se à disponibilização dos componentes desenvolvidos e à instalação do editor da DSL e do gerador respectivo.

Também a componente de disponibilização de documentação de suporte foi facilitada pela total automatização do processo de produção. Ainda assim, foi disponibilizado um manual de utilização da infra-estrutura de produção para especificar os procedimentos em casos de produção semiautomática.

Gestão de Projecto	Monitorização e controlo Planeamento e acompanhamento de iterações Avaliação de objectivos Avaliação de continuidade da LP Preparação de projectos de Aplicação
Gestão de Config.s e Mudança	Atribuir versões a artefactos criados Gestão de versões da framework Gestão de versões de componentes Gestão de versões de geradores Gestão de versões de meta-modelos DSL
Ambiente	Promoção da melhoria contínua Disponibilização correcta de ferramentas Classificação e arquivo de artefactos criados

4.3.2 ENGENHARIA DE APLICAÇÃO

O objectivo deste projecto não é substituir as aplicações existentes pelos sistemas produzidos pela linha de produtos. Essa será a meta a longo prazo mas não a do projecto-piloto. O intuito do projecto-piloto é avaliar a capacidade da organização transitar da abordagem de desenvolvimento tradicional para um paradigma orientado à construção de famílias de sistemas.

Como era importante simular o processo num ambiente real, a ITCORP optou por apresentar o projecto a dois clientes de longa data e solicitar a sua participação. O ISLX e o ISPT (nomes de instituições de ensino superior fictícias) aceitaram o desafio.

4.3.2.1 Aplicação para ISLX

4.3.2.1.1 Análise do Problema

A análise do problema foi feita em função das (actuais) limitações da LP. Nem o cliente, nem o domínio eram novos, por isso o analista de negócio, baseada na aplicação existente, começou por anunciar uma solução com as características mínimas (essencialmente inserir e remover alunos). Desde logo, o Cliente manifestou três necessidades que não eram atendidas pelas capacidades mínimas:

1. Pesquisar fichas de aluno pelo nome do aluno
2. Arquivar fichas de aluno (para os alunos que terminam o vinculo com o ISLX mas não podem ser removidos do sistema)
3. Exportar listas de alunos em formato Microsoft Excel

O analista de negócio sabia que a LP não considerava nenhuma solução directa para a necessidade número 2 por isso propôs uma alternativa. Alertando o Cliente para a demora e custo do desenvolvimento adicional associado, o Analista gerou um protótipo em que a entidade "Aluno" tinha um campo especial "Arquivado" e a listagem continha um filtro que permitisse restringir os resultados da listagem de acordo com a informação que constasse nesse campo (Sim ou Não), conseguindo uma funcionalidade próxima da requerida pelo Cliente. Após alguns testes ao protótipo, o Cliente decidiu aceitar a solução.

Alcançado o acordo, seguiu-se a formalização das características do produto a gerar, acordadas com o Cliente. Paralelamente, o analista de negócio preparou o documento de feedback onde referiu a necessidade de "Arquivar fichas de alunos" e reforçou a importância do protótipo para chegar a acordo com o Cliente.

Gestão de Projecto	Monitorização e controlo Planeamento e acompanhamento operacional Gestão dos recursos Gestão do feedback
Gestão de Config.s e Mudança	Atribuição de versões a artefactos criados Gestão de versões dos modelos criados
Ambiente	Promoção da melhoria contínua Disponibilização correcta da infra-estrutura de produção Classificação e arquivo de artefactos criados

4.3.2.1.2 Desenvolvimento adicional

Não ocorreu pois a LP implementava todos os elementos necessários.

4.3.2.1.3 Produção e Testes

A solução encontrada era completamente coberta pela linha de produtos. Nestas condições o processo de produção da aplicação fica reduzido à construção do modelo da aplicação. Esta modelação ficou a cargo de um Developer de Aplicação que utilizou o Visual Studio 2010 devidamente configurado a *framework*, componentes, DSL e gerador.

O manual de utilização da infra-estrutura de produção disponibilizado pela equipa de Ambiente especificava um conjunto de testes para validar a aplicação gerada. Esses testes foram executados e nenhum problema foi detectado.

Vídeo com demonstração do processo de produção: http://www.youtube.com/watch?v=46ozC62VfCo

Gestão de Projecto	Monitorização e controlo Planeamento e acompanhamento operacional Gestão dos recursos
Gestão de Config.s e Mudança	Atribuição de versões a artefactos criados Gestão de versões dos modelos criados Gestão de versões das aplicações
Ambiente	Promoveção da melhoria contínua Disponibilização correcta da infra-estrutura de produção Classificação e arquivo de artefactos criados

4.3.2.1.4 Deployment

Dado o carácter temporário e limitado da solução, a aplicação foi instalada em servidores da ITCORP e disponibilizada na internet para o Cliente experimentar. Não se tratava propriamente de um ambiente de produção mas ainda assim foram repetidos alguns testes para assegurar o correcto funcionamento da aplicação.

Gestão de Projecto	Monitorização e controlo Planeamento e acompanhamento operacional Assegurar recursos
Gestão de Config.s e Mudança	Atribuir versões a artefactos criados Gerir versões das aplicações
Ambiente	Promover a melhoria contínua Assegurar correcta disponibilização da infra-estrutura de produção Classificação e arquivo de artefactos criados

4.3.2.2 Aplicação para ISPT

4.3.2.2.1 Análise do Problema

O enquadramento do ISPT é semelhante ao do ISLX e isso reflectiu-se na abordagem seguida pelo Analista de Negócio. A diferença principal colocou-se nas necessidades específicas que variavam consideravelmente em relação ao ISLX. Para além das necessidades já referidas (pesquisa por nome, arquivo de fichas e exportação para *excel*) o ISPT considerava essencial:

1. Expor a informação dos alunos para ser utilizada noutras aplicações
2. Ter a capacidade de importar fichas de alunos criadas noutros sistemas
3. Ordenar alunos de acordo com qualquer critério
4. Ter a capacidade de configuração avançada que permitisse construir filtros de listagens à medida
5. Exportar listas de alunos para HTML

Os pontos 1, 3 eram directamente considerados pela LP. Os restantes careciam de atenção especial:

- Importação de fichas de alunos (ponto 2): não havia qualquer indicação na documentação da infra-estrutura de produção que permitisse criar uma funcionalidade que sequer se aproximasse desse comportamento.
- Filtros à medida (ponto 4): a LP não disponibilizava nenhuma funcionalidade que permitisse o utilizador final criar novos filtros. Contudo, a capacidade de gerar qualquer tipo de filtro era possível na altura da geração.
- Listas em HTML (ponto 5): a LP só disponibilizava a exportação para Excel. A exportação para outros formatos era referida na documentação da infra-estrutura de geração como algo possível já que existiam interfaces preparadas para tal. Havia, contudo, necessidade de desenvolvimento adicional.

O Analista de negócio iniciou, em conjunto com o Cliente, a construção de um protótipo com as características já existentes na LP e com filtros para todos os campos (para tentar demover o Cliente da necessidade de criação de novos filtros). O Cliente aceitou a alternativa dos filtros mas rejeitou por completo a não inclusão da importação de fichas e listagens em HTML.

O Analista de negócio comunicou ao gestor de Projecto tal impasse. Em resultado, a situação foi avaliada pelo Comité de Mudança da LP. O Comité analisou as condições colocadas pelo Cliente, confrontou-as com o interesse da LP e chegou à seguinte conclusão:

- Existiam condições para avançar com o desenvolvimento dos novos formatos de exportação. O esforço seria reduzido pois as interfaces estavam criadas e existia uma elevada probabilidade do componente ser reutilizado noutras aplicações. O trabalho a realizar foi estimado em 2 dias de desenvolvimento.
- Não havia interesse em avançar com a capacidade de importação pois esse componente obrigaria a alterar a infra-estrutura, o que seria impraticável no âmbito de um projecto de Aplicação. Em todo o caso, essa informação ficou registada para futura evolução da LP.

O Analista de negócio transmitiu essas conclusões ao Cliente. Dado o carácter experimental do projecto, o Cliente decidiu aceitar a proposta embora tenha alertado que, se fosse num cenário menos experimental, poderia não ter a mesma deliberação. Foi acordado o prazo máximo de uma semana para a solução ser disponibilizada com os componentes adicionais.

O Analista de Negócio procedeu então à especificação do produto (características existentes e desenvolvimento adicional) e à avaliação da LP.

Gestão de Projecto	Monitorização e controlo Planeamento e acompanhamento operacional Gestão dos recursos Gestão do feedback
Gestão de Config.s e Mudança	Atribuição de versões a artefactos criados Gerir solicitações de mudança e articulação com Comité de Mudança Gerir versões dos modelos criados
Ambiente	Promoveção da melhoria contínua Disponibilização correcta da infra-estrutura de produção Classificação e arquivo de artefactos criados

4.3.2.2.2 Desenvolvimento adicional

Dada a deliberação do comité de Mudança em avançar com o desenvolvimento do novo formato de exportação, o gestor de Projecto nomeou um *Developer* para implementar esse novo componente.

O *developer* de Aplicação começou por analisar a documentação da *framework* e os componentes propostos pelo engenheiro de reutilização. Dada a similaridade entre o componente que faz a exportação para Excel e o novo componente, o desenvolvimento foi bastante rápido e não chegou aos dois dias estimados. Após conclusão de testes e elaboração da especificação dos novos componentes o *developer* deu o desenvolvimento por concluído.

Gestão de Projecto	Monitorização e controlo Planeamento e acompanhamento operacional Gestão dos recursos
Gestão de Config.s e Mudança	Atribuição de versões a artefactos criados Gestão de versões dos componentes
Ambiente	Promoção da melhoria contínua Disponibilização correcta de ferramentas Classificação e arquivo de artefactos criados Apresentação de proposta de reutilização de componentes

4.3.2.2.3 Produção e Testes

A solução definida não era totalmente coberta pela linha de produtos, seria necessário recorrer ao processo de produção semiautomático. De acordo com o manual de infra-estrutura de processo, o processo seria constituído pela criação do modelo da aplicação e depois o acoplamento manual dos novos componentes desenvolvidos (que a DSL ainda não considerava). Os trabalhos decorreram como previsto e de acordo com os testes que se seguiram a aplicação ficou a funcionar correctamente.

Vídeo com demonstração do processo de produção: http://www.youtube.com/watch?v=R1HOpbPLW3M

Gestão de Projecto	Monitorização e controlo Planeamento e acompanhamento operacional Gestão dos recursos
Gestão de Config.s e Mudança	Atribuição de versões a artefactos criados Gestão de versões dos modelos criados Gestão de versões das aplicações
Ambiente	Promoção da a melhoria contínua Disponibilização correcta da infra-estrutura de produção Classificação e arquivo de artefactos criados

4.3.2.2.4 Deployment

Tal como no caso do ISLX, considerando o carácter temporário e limitado da solução, a aplicação foi instalada em servidores da ITCORP e disponibilizada na internet para o Cliente testar. Não se tratava propriamente de um ambiente de produção mas ainda assim foram repetidos alguns testes para assegurar o correcto funcionamento da aplicação.

Gestão de Projecto	Monitorização e controlo Planeamento e acompanhamento operacional Gestão dos recursos
Gestão de Config.s e Mudança	Atribuição de versões a artefactos criados Gestão de versões das aplicações
Ambiente	Promoção da a melhoria contínua Disponibilização correcta da infra-estrutura de produção Classificação e arquivo de artefactos criados

4.4 CONSIDERAÇÕES SOBRE O CASO DE ESTUDO

O objectivo principal da elaboração do caso de estudo é iterar o modelo desenvolvido para demonstrar como este é aplicável tanto na construção de uma infra-estrutura de produção de sistemas como na sua instanciação (neste caso para dois clientes, ISLX e ISPT).

As situações apresentadas neste caso de estudo são simples mas reflectem alguns aspectos importantes do modelo:

- A capacidade de reutilização massiva utilizando os componentes reutilizáveis e plataforma de produção. A aplicação para o ISLX é construída 100% com componentes já existentes (embora possa parecer um pouco idílico, em cenários em que a organização tem um profundo conhecimento de domínio, este nível de reutilização é possível). A aplicação do ISPT também é construída com base nos componentes existentes embora a sua especificidade obrigue ao desenvolvimento de componentes adicionais (apesar do desenvolvimento adicional, a reutilização massiva mantém-se).
- A "integração" do cliente no processo de construção da aplicação. Em todos os processos de definição da solução, o analista de negócio define a solução com o cliente através de construção rápida de protótipos que asseguram a mútua compreensão dos conceitos envolvidos.
- O complexo processo decisório associado ao desenvolvimento adicional. A linha de produtos tem um âmbito pré-definido. Como a infra-estrutura de produção e componentes reutilizáveis são desenvolvidos com base nessa especificação, alterações a esse âmbito devem ser cuidadosamente analisadas para garantir que os resultados esperados são alcançados.
- A análise de domínio apresentada foi extremamente simples mas realça um aspecto importante. Ao contrário da perspectiva tradicional, é imperativo (e não só desejável), que a equipa de desenvolvimento conheça em profundidade o domínio em causa para conseguir antecipar assertivamente os requisitos que estarão envolvidos na linha de produtos.
- A construção do protótipo da plataforma de produção permite ter uma perspectiva prática de como um plano produção é executado num contexto de linhas de produto. Como é demonstrado pelos vídeos que apresentámos, o processo de construção é simples e muitas vezes pode ser desempenhado por elementos da equipa menos técnicos e mais relacionados com o negócio.

Ainda neste espaço de considerações, gostávamos de realçar mais um aspecto. Porque acreditamos que simplicidade do caso de estudo possa induzir a uma conclusão errada sobre uma eventual desvantagem de aplicar um esforço adicional na construção de uma infra-estrutura de produção e componentes reutilizáveis, salientamos que se trata apenas do projecto-piloto e que só com a futura aplicação da infra-estrutura de produção aos restantes clientes é que a ITCORP espera amortizar o seu investimento.

5 CONCLUSÕES E TRABALHOS FUTUROS

5.1 CONCLUSÕES

O objectivo principal do trabalho foi a definição de um modelo de desenvolvimento baseado em linhas de produto. Esse objectivo foi cumprido com a definição do modelo SPLUP.

Tal como planeado, o modelo desenvolvido é independente do domínio de aplicação, i.e., é suficientemente genérico para ser aplicado a vários tipos de famílias de produtos. Apesar desta abrangência de aplicação, o modelo assegura a definição de um processo concreto que permite, de forma realista, apoiar e guiar a sua execução.

Consideramos que as principais contribuições e aspectos diferenciadores do modelo SPLUP face às abordagens que analisámos são:

- Disponibiliza um modelo de desenvolvimento completo para um programa SLP, incluindo:
  - Configuração do ciclo de desenvolvimento;
  - Determinação de objectivos para diferentes fases do projecto;
  - Definição das disciplinas técnicas envolvidas,
  - Definição de fluxos de trabalho e papéis.
- É baseado em boas práticas e técnicas existentes:
  - É fundamentado numa framework que sistematiza um conjunto de práticas de desenvolvimento de linhas de produto com mérito reconhecido pela comunidade: o FSPLP;
  - Agrega um conjunto de tecnologias aplicadas com sucesso em inúmeras iniciativas de LP (e.g. domain-specific languages ou geradores de código)

Para além da abrangência e fundamentação em "boas práticas" do SPLUP, acreditamos que um importante contributo da nossa proposta está na forma como é concretizada a articulação dos dois processos que constituem o desenvolvimento de linhas de produtos: a engenharia de domínio e a engenharia de aplicação. Nas propostas analisadas, é consensual a necessidade de articulação, mas, em nenhuma delas esse aspecto é concretizado. Pelo contrário, no SPLUP, a ligação dos dois processos é pré-estabelecida com a definição de disciplinas/fluxos de trabalho específicos com responsáveis associados (actores), garantindo que os principais pontos de comunicação são assegurados.

Outro aspecto que consideramos importante no modelo que propomos é o facto de a sua organização ser inspirada em metodologias reconhecidas (RUP e OpenUP). Na nossa opinião, esta opção favorece a transição de uma abordagem de desenvolvimento tradicional para a abordagem baseada em linhas de produto, pois facilita a compreensão do processo e das suas implicações.

O SPLUP é uma metodologia exigente porque os princípios SPL genéricos também o são. Ainda assim, tal como comprovámos no caso de estudo apresentado, acreditamos que é possível implementar o modelo num cenário real pois , ainda que simples, o caso apresentado, representa um cenário bastante comum: uma empresa tecnológica presente há vários anos num determinado domínio de actuação, neste caso no desenvolvimento de sistemas de informação para estabelecimentos de ensino, que graças a essa experiência, reúne uma carteira de clientes e conhecimento importante sobre esse domínio.

Partindo desse cenário, o caso de estudo permitiu validar a abordagem metodológica proposta pelo modelo para guiar a reformulação da estratégia de desenvolvimento tradicional rumo a uma abordagem baseada em família de produtos que potencie o extenso conhecimento de domínio e técnico e que o materialize num conjunto coerente de elementos reutilizáveis integrados numa plataforma de produção (família de produtos).

Ainda no âmbito do caso de estudo, o protótipo desenvolvido foi importante para confirmar as potencialidades da integração das várias tecnologias consideradas no modelo. De facto, verificou-se que a integração dessas tecnologias permite construir aplicações num nível de abstracção superior ao tradicional, maioritariamente orientado por modelos, o que faz com que a instanciação da infra-estrutura se torne num processo rápido e simples e que, inclusive, pode ser incluído numa fase preliminar do projecto, numa lógica de prototipagem, por forma a permitir uma participação activa do cliente na definição das características da solução.

5.2 TRABALHOS FUTUROS

O desenvolvimento baseado em linhas de produto engloba uma reestruturação completa do ciclo tradicional de desenvolvimento com alterações consideráveis em todas as fases do projecto. Esta transversalidade impossibilitou o estudo de todas as suas implicações, em particular, quando se trata de uma análise limitada pelo âmbito restrito de uma dissertação de mestrado.

Neste contexto, apresentamos algumas limitações que o SPLUP apresenta e que se poderão perfilar como potenciais trabalhos futuros:

- FSPLP tem muito mais informação a explorar do que a que considerámos neste trabalho, e.g. cada área de prática tem identificação de potenciais riscos, técnicas aplicáveis ou até uma bibliografia de referência para cada área. Um trabalho futuro poderá enriquecer o modelo com essa informação;
- O RUP também poderá proporcionar uma contribuição adicional, que vá muito para além da vertente de inspiração que foi adoptada neste trabalho. Vários artefactos, papéis ou fluxos de trabalho podem ser adaptados e incorporados no SPLUP;
- As tecnologias escolhidas focam essencialmente o lado da implementação (código) do projecto. De acordo com a filosofia SPL, todos os artefactos são potencialmente reutilizáveis, logo seria interessante, um trabalho futuro enriquecer o modelo

SPLUP com técnicas e tecnologias menos ligadas à implementação, considerando, por exemplo, aspectos como a reutilização de documentação, a reutilização de testes ou ainda a gestão e reutilização de conhecimento;

Na nossa opinião, os três pontos que acabámos de enumerar constituem oportunidades para uma imediata evolução do trabalho que desenvolvemos. Adicionalmente, existem vários aspectos do desenvolvimento baseado em linhas de produto que não pudemos abordar, mas que são muito importantes para qualquer cenário de implementação real. Dentro destes distinguimos os que consideramos mais relevantes:

- Aspectos económicos: definir formas de cálculo da rentabilidade dos sistemas incluídos numa família de produtos e modelos de avaliação económica da rentabilidade da linha de produtos (e.g. definição do break-even point de acordo com as características do projecto em causa).
- Modelos de maturidade: integrar o processo de desenvolvimento com os níveis de maturidade de um modelo de maturidade (por exemplo com o Capability Maturity Model Integration – CMMi)
- Aplicação do modelo em domínios específicos, de forma a consolidar os mecanismos de extensão e de personalização do modelo.

Para além destes potenciais aspectos de evolução, a própria validação do modelo poderá ser aprofundada com a recolha de opiniões de especialistas e eventual realização de um projecto-piloto real.

5.3 ARTIGOS

Durante o desenvolvimento deste trabalho foram submetidos dois artigos científicos:

- "SPLUP: Software Product Line Unified Process" submetido para CENTERIS'2011 - Conference on ENTERprise Information Systems
- "Software Product Lines: a realistic path to software development industrialization?" submetido para a 11ª Conferência da Associação Portuguesa de Sistemas de Informação (CAPSI 2011).

6 BIBLIOGRAFIA

Ambler, S. W. (1999). A Manager ' s Introduction to The Rational Unified Process (RUP).

Beck, K., Beedle, M., Bennekum, A. van, Cockburn, A., Cunningham, W., & Fowler, M. (2001). Agile Manifesto. Retrieved from http://agilemanifesto.org/principles.html.

Bosch, J. (2000). *Design and use of software architectures: adopting and evolving a product-line approach*.

Bosch, J., Molin, P., Mattsson, M., & Bengtsson, P. (1997). Object-Oriented Frameworks - Problems & Experiences. *Business*.

Bragança, A. (2007). Methodological Approaches and Techniques for Model Driven Development of Software Product Lines. *PhD Thesis, Information Systems Department, University of Minho*.

Chakravarthy, V., Regehr, J., & Eide, E. (2008). Edicts : Implementing Features with Flexible Binding Times. *Interpreting*, 1-12.

Clements, P., & Northrop, L. (2001). *Software Product Lines: Practices and Patterns* (p. 608). Addison-Wesley Professional. Retrieved November 5, 2010, from http://www.amazon.com/Software-Product-Lines-Practices-Patterns/dp/0201703327.

Cohen, S. (2010). Managing Variation in Services in a Software Product Line Context. *Program*, (May).

Corporation, R. S. (2005). Rational Unified Process Best Practices for Software Development Teams. *Rational Software White Paper*.

Cusumano, M. (1989). Software Factories - A Historical Interpretation.

Czarnecki, K., & Eisenecker, U. (2000). *Generative programming: methods, tools, and applications* (p. 832). Addison Wesley. Retrieved November 5, 2010, from http://books.google.com/books?id=4CPmr3qcVvYC&pgis=1.

Eveleens, L., & Verhoef, C. (2010). The Rise and Fall of the Chaos Report. *Project Management Focus*.

Fayad, M. (1997). Object-Oriented Application Frameworks. *ACM Computing Surveys*, *32*(1es), 28-es. doi: 10.1145/351936.351964.

Fowler, M. (2011). Martin Fowler - Domain Specific Languages. Retrieved February 13, 2011, from http://martinfowler.com/bliki/DomainSpecificLanguage.html.

Froehlich, G., Hoover, H. J., Liu, L., & Sorenson, P. (1995). Designing Object-Oriented Frameworks. *Framework*, 1-30.

G, S., & Berger, T. (2008). Service-Oriented Product Lines : Towards a Development Process and Feature Management Model for Web Services. *Information Systems*.

Greenfield, J., Short, K., Cook, S., & Kent, S. (2004). *Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools* (p. 500). Wiley. Retrieved November 5, 2010,

from http://www.amazon.com/Software-Factories-Assembling-Applications-Frameworks/dp/0471202843.

Greenfield, J., Short, K., Studio, V., Tools, E., & Corporation, M. (2004). Moving to Software Factories. *MSDN*.

IEEE. (1990). IEEE Standard Glossary of Software Engineering Terminology. *Office*, *121990*.

Institute, S. E. (2010a). Framework for Software Product Line Practice. Retrieved from http://www.sei.cmu.edu/productlines/.

Institute, S. E. (2010b). Product Line Hall of Fame. Retrieved February 1, 2011, from http://www.splc.net/fame.html.

Jha, M., & Brien, L. O. (2009). Identifying Issues and Concerns in Software Reuse in Software Product Lines. *Most*.

Johnson, R. E. (1991). Designing Reuseable Classes. *Information Systems*, (217), 1-27.

Josuttis, N. (2007). *SOA in Practice*.

Kelly, S., & Tolvanen, J.-P. (2007). *Domain-Specific Modeling*.

Kleppe, A., Warmer, J., & Bast, W. (2003). *MDA Explained: The Model Driven Architecture(TM): Practice and Promise* (p. 192). Addison-Wesley Professional. Retrieved November 5, 2010, from http://www.amazon.com/MDA-Explained-Architecture-Practice-Promise/dp/032119442X.

Landin, N., Niklasson, A., Bosson, G., & Technology, E. S. (1995). Development of Object Oriented Frameworks. *Analysis*.

Lee, K., Kang, K. C., & Lee, J. (2002). Concepts and Guidelines of Feature Modeling for Product Line Software Engineering. *Department of Computer Science and Engineering - Pohang University of Science and Technology*.

Lenz, G., & Wienands, C. (2006). *Practical Software Factories in .NET (Books for Professionals by Professionals)* (p. 240). Apress. Retrieved November 5, 2010, from http://www.amazon.com/Practical-Software-Factories-Books-Professionals/dp/159059665X.

McIlroy, M. D. (1968). Mass Produced Software Components, *Software E*(October 1968), 1-12.

MELLOR, S. J., Scott, K., Uhl, A., & Weise, D. (2004). *MDA Distilled* (p. 176). Addison-Wesley Professional. Retrieved November 5, 2010, from http://www.amazon.com/MDA-Distilled-Stephen-J-MELLOR/dp/0201788918.

Northrop, L. (2003). Software Product Lines Introduction. *Engineering*, 1-105.

Northrop, L. (2008). Software Product Lines Essentials. *Engineering*.

Pohl, K., Böckle, G., & Linden, F. van der. (2005). *Software product line engineering: foundations, principles, and techniques* (p. 467). Birkhäuser. Retrieved November 5, 2010, from http://books.google.com/books?id=jiTvwsMEgL8C&pgis=1.

Probasco, L. (2000). The Ten Essentials of RUP. *Rational Software White Paper*.

Process, U. (2004). Introduction to OpenUP (Open Unified Process). *Eclipse EPF*, 1-9.

Project, E. P. F. (2011). OpenUP. Retrieved March 5, 2011, from http://epf.eclipse.org/wikis/openup/.

Reis, G. D. (2005). What is Generic Programming ? *Computer*, 1-11.

Saraiva, J., Silva, A. (2010). A Reference Model for the Analysis and Comparison of MDE Approaches for Web-Application Development, J. Software Engineering & Applications, 2010, 3: 419-425.

Sochos, P., Philippow, I., & Riebisch, M. (2004). Feature-Oriented Development of Software Product Lines : Mapping Feature Models to the Architecture.

Steyaert, P. (1997). From custom applications to domain-specific frameworks. *Communications of the ACM*, *40*(10), 70-77. doi: 10.1145/262793.262807.

Svahnberg, M. (2000). Variability in Evolving Software Product Lines.

Sven, N. S., & Saake, A. G. (2011). Flexible Feature Binding in Software Product Lines. *Automated Software Engineering*.

Tirilä, A. (2003). Variability enabling techniques for software product lines.

Vrani, V., & Snirc, J. (2006). Integrating Feature Modeling into UML.

Wthreex. (2011). Rational Unified Process. Retrieved February 4, 2011, from http://www.wthreex.com/rup/smallprojects/index.htm.

Ye, H., & Liu, H. (2005). Approach to modelling feature variability and dependencies in software product lines. *Electrical Engineering*, 101-109. doi: 10.1049/ip-sen.

Zhang, J., Cai, X., & Liu, G. (2008). The Role of Aspects in Software Product Lines. *2008 International Conference on Computer Science and Information Technology*, 588-592. Ieee. doi: 10.1109/ICCSIT.2008.135.

Ípka, M. (2005). Exploring the Commonality in Feature Modeling Notations, 139-144.

7 ANEXO – ÁREAS DE PRÁTICA FSPLP

Documentação completa FSPLP: http://www.sei.cmu.edu/productlines/frame_report/index.html

ÁREAS DE PRÁTICA DE ENGENHARIA DE SOFTWARE

Definição de Arquitectura

A arquitectura de uma aplicação é o alicerce de qualquer sistema de informação. Numa abordagem de linha de produtos, existem dois tipos de arquitecturas que devem ser considerados, a arquitectura da linha de produtos, enquanto figura base dos elementos core, e a arquitectura dos produtos gerados, fruto do exercício da variabilidade sobre a primeira.

A arquitectura da linha de produtos é o principal activo reutilizável. Deve ser construído em directa articulação com o âmbito definido para a linha de produto, estabelecendo directamente os componentes disponíveis e suas relações e formulando assim a capacidade de variação existente.

A arquitectura base disponibiliza um conjunto de pontos de variação que podem ser utilizados de acordo com a especificação do produto concreto que se pretende construir. Cumulativamente, a própria arquitectura base pode ser construída de forma modular e assim disponibilizar um conjunto de configurações à caracterização da arquitectura do produto final.

Avaliação de Arquitectura

A avaliação da arquitectura procura aferir se os objectivos para a SPL, e.g. metas de negócio ou dimensões âmbito, são reflectidos na arquitectura. Como referido anteriormente, o termo arquitectura, num contexto SPL, pode ser visto em duas perspectivas distintas: enquanto elemento core reutilizável ou como estrutura base dos produtos gerados. Dados os objectivos distintos das duas perspectivas, a avaliação deve efectuada para os dois elementos de forma diferenciada.

É essencial validar se a arquitectura da SPL disponibiliza os mecanismos de variação necessários para assegurar o cumprimento do âmbito definido para a linha de produtos bem como se os objectivos de negócio são atingidos. Paralelamente, deve ser avaliada a relação entre as características da arquitectura do produto gerado (baseada na arquitectura base da linha de produtos) e os requisitos particulares do cenário de implementação (e.g. performance, disponibilidades, etc).

Desenvolvimento de componentes

A Framework define o desenvolvimento de componentes como a actividade de "implementar uma funcionalidade específica no contexto de uma arquitectura de software". Na prática, o desenvolvimento de componentes, corresponde à actividade de implementar uma funcionalidade, de uma forma reutilizável e contextualizada pela arquitectura alvo, disponibilizando um conjunto de interfaces que permitam a sua integração com outros componentes.

Exploração de activos existentes

Os activos existentes são uma importante fonte para a construção de uma linha de produtos. A actividade de exploração dos activos existentes visa a análise dos projectos desenvolvidos pela organização com o objectivo de os reaproveitar numa óptica de reutilização. Para tal é necessário compreender o que existe, verificar possível alinhamento com a linha de produtos e, eventualmente, generalizar o activo de forma a torná-lo reutilizável. O termo activo pode representar, uma aplicação, documentação, um componente, um modelo, um caso de testes, uma especificação de negócio, etc.

Engenharia de requisitos

A actividade de engenharia de requisitos é, à semelhança da abordagem tradicional, uma das unidades mais importantes e simultaneamente mais difíceis de realizar na iniciativa SPL. Ainda que não haja distinção no processo genérico de engenharia de requisitos típico (recolha, análise, especificação, verificação e gestão) aplicado a uma SPL ou aplicado a um projecto one-off, existem diferenças na forma de realizar cada fase. No âmbito de uma SPL, a fase de recolha envolve a antecipação de requisitos, i.e. ao contrário da visão tradicional, a recolha não se baseia apenas cenário presente mas também na previsão das funcionalidades/características que produtos semelhantes poderão requerer. Por outro lado, a fase de análise não se foca apenas na capacidade de executar dos requisitos: numa SPL esta fase é dedicada ao estudo dos aspectos comuns e variáveis que a linha de produtos deve oferecer.

Integração de software

À integração de software respeita a combinação dos componentes em sistemas. A Framework propõe o apuramento da estimativa do esforço envolvido nesta fase pela identificação do tipo de "personalização" de produto necessária.

Quando as generalizações consideradas pela linha de produtos cobrem, por completo, a especificação do produto a construir o esforço de integração é baixo, resume-se à configuração/parametrização dos elementos reutilizáveis de acordo com o plano de produção. Por outro lado, quando as características do produto são tão particulares que não permitem a reutilização dos componentes (mesmo considerando os mecanismos de variação/extensão estabelecidos), estes têm de ser alterados – eventualmente outros construídos - o que exige um elevado esforço de integração.

Testes

Os testes, à semelhança da perspectiva tradicional, visam antecipar problemas e verificar o cumprimento dos requisitos. No entanto, na abordagem SPL, os testes têm uma particularidade: são reaproveitados, isto é, dado o domínio restrito partilhado pelos produtos, os mesmos testes são aplicáveis a vários produtos e os resultados comparáveis.

À semelhança do exposto para outras áreas de prática, também os testes devem ser diferenciados para o desenvolvimento dos activos core e para a construção dos produtos. Os activos reutilizáveis devem ser testados em função do seu papel na linha de produtos. Para tal, deve verificar-se a sua correcção enquanto elementos independente mas também no ambiente de integração potencial.

Quanto ao desenvolvimento de produtos, para além da validação do cumprimento dos requisitos, os testes devem ser incorporados como parte do processo produtivo, isto é, a saída de cada fase deve ser testada antes de ser de entrada na fase seguinte.

Compreender domínios relevantes

Um domínio corresponde a uma área de conhecimento aplicável à construção de um sistema. Usualmente, o desenvolvimento de um sistema envolve conhecimentos de vários domínios.

A relevância da actividade de "compreensão do domínio" é já reconhecida nas abordagens tradicionais de desenvolvimento. A multiplicidade de cenários de implementação, bem como a sua complexidade, são dois dos principais factores que justificam essa importância para o alinhamento final entre as necessidades e os resultados dos projectos.

No âmbito de uma linha de produtos, a análise correcta dos domínios tem uma importância reforçada pelo carácter de longo prazo associado a um desenvolvimento orientado por linhas de produtos. Para além dessa leitura, esta actividade contribuirá, activamente, para a definição do âmbito da linha de produtos e permitirá a identificação dos mecanismos de variação apropriados.

Utilização de software disponível externamente

Os componentes de uma linha de produtos não necessitam, obrigatoriamente, de ser desenvolvidos de raiz. Em alternativa, a consideração de componentes (software) disponível do mercado é, em muito casos, uma forma de se encontrarem importantes economias no esforço dispendido na construção da linha de produtos.

Contudo a aquisição de componentes deve seguir um rigoroso processo que envolva um planeamento adequado do enquadramento do activo na visão da linha de produtos, bem como a avaliação do comportamento do componente de forma a verificar a sua fiabilidade.

ÁREAS DE PRÁTICA DE GESTÃO TÉCNICA

Gestão de Configurações

A gestão de configurações é a actividade responsável pela avaliação, coordenação, aprovação/reprovação, e implementação das alterações nos artefactos que são usados para construir e manter um sistema de software.

No desenvolvimento tradicional, a gestão de configurações de um sistema dá origem a um processo isolado, isto é, as alterações efectuadas em diferentes aplicações correspondem a processos de gestão de configurações diferentes. Numa estratégia de linha de produtos, existem três tipos de processo de gestão de configurações que não podem funcionar de forma isolada. À semelhança da abordagem tradicional, um processo de gestão de configurações deve ser estabelecido para cada produto gerado, no entanto, este tem de estar em directa articulação com o processo de gestão de configurações da linha de produtos – alterações na linha podem ter impacto directo nos produtos. Ao mesmo tempo, os activos reutilizáveis também estão sujeitos a alterações logo, a cada activo deve estar associado um processo de gestão de alterações específico. A Figura 7-i ilustra a referida articulação entre os três tipos de processos de GM.

Figura 7-i – Gestão de Configurações

Uma correcta gestão de configurações deve permitir replicar a geração de qualquer versão de um produto.

Análise Desenvolver, Comprar, Extrair, Subcontratar

Esta actividade corresponde à avaliação da abordagem a seguir na construção (de elementos) da linha de produtos. Os autores distinguem duas perspectivas de incorporação de elementos:

- Por esforço interno: pelo desenvolvimento de raiz (Desenvolver) ou pela exploração de sistemas existentes (Obter internamente)
- Por recurso ao exterior: pela aquisição de componentes disponíveis no mercado (Comprar) ou pela encomenda do desenvolvimento a uma entidade terceira (Encomendar).

A análise é guiada por factores como o custo de cada opção, restrições de calendário ou disponibilidade de recursos. Importa ainda que referir que o mesmo elemento pode ser construído agregando várias opções, e.g. um activo core pode ser constituído por um componente adquirido no mercado e personalizado/alterado por esforço interno.

Medição e Monitorização

O alinhamento entre os objectivos estratégicos da linha de produtos e as práticas de desenvolvimento deve ser sempre verificado. Ao invés do controlo informal, a Framework propõe a definição dos objectivos em formato quantificável em conjunto com a determinação de um conjunto de métricas que permitam aferir o seu cumprimento.

A diferença principal desta actividade entre o desenvolvimento tradicional e o desenvolvimento num ambiente de linha de produtos reside no facto dos objectivos/métricas deverem ser definidos para as três actividades base do processo: o desenvolvimento dos activos reutilizáveis, a produção dos produtos e a gestão da iniciativa. Todas as perspectivas da iniciativa devem ser monitorizadas e avaliadas de forma independente (por forma a identificar lacunas mais facilmente) e depois conjugadas num resultado global.

Disciplina de processo

Uma das formas que o desenvolvimento tradicional encontrou para lidar com a complexidade dos projectos é a definição rigorosa (bem como o seu contínuo melhoramento) dos processos envolvidos no desenvolvimento. A área de prática "Disciplina de Processo" representa esse esforço aplicado ao desenvolvimento de linhas de produto.

O desenvolvimento de uma linha de produtos deverá ser um esforço coordenado entre várias equipas (tipicamente a equipa responsável pelo desenvolvimento dos activos core não é a mesma da actividade de construção de produtos nem tão pouco da equipa de gestão). Acresce ainda que uma equipa afecta a uma actividade base é muitas vezes constituída por várias "sub-equipas" que se dedicam a diferentes aspectos do desenvolvimento. Esta organização distribuída de recursos, reforça a importância da gestão dos processos associados à coordenação de trabalhos de forma a minimizar número de incidentes e melhorar a produtividade.

Definição do âmbito

O âmbito de um projecto permite limitar o campo de actuação, caracterizando qual o resultado expectável.

Ao contrário da generalidade dos projectos de desenvolvimento tradicional, a definição do âmbito nas SPL é altamente formal, especificando o que é previsível, a curto, médio e longo prazo. Numa perspectiva prática, o âmbito de uma linha de produtos estabelece as características dos produtos que a linha de produtos é capaz de gerar.

O âmbito guiará directamente que activos core são construídos, bem como, a decisão de construção de um produto (variando a aprovação de produção em função das características do produto a gerar vs características consideradas pelo âmbito).

Planeamento técnico

A Framework distingue dois tipos de planeamento: o técnico e o organizacional (outra área de prática). O planeamento técnico foca a organização do esforço de desenvolvimento associado a um projecto.

As linhas de produtos definem dois importantes tipos de planos: os planos de desenvolvimento e manutenção dos activos reutilizáveis e os planos de produção de produtos. Estes planos normalmente são instâncias de um plano genérico de desenvolvimento e manutenção de activos reutilizáveis e de um plano genérico de produção de elementos da família (respectivamente). O planeamento é essencial para a iniciativa, nomeadamente para garantir a capacidade de monitorização e mediação de desempenho mas também para assegurar condições para que a coordenação de esforços esteja sincronizada.

Gestão do risco (perspectiva técnica)

Tal como a actividade de planeamento, a gestão do risco também foi distinguida numa perspectiva técnica e outra organizacional. O risco representa algo que pode prejudicar o correcto andamento do projecto. A gestão do risco visa a identificação das áreas de risco, a definição de mecanismos de resposta e a determinação de papéis e responsabilidades de intervenientes.

O processo de gestão de risco é semelhante à perspectiva tradicional, todavia, numa SPL, a concretização de um risco, i.e. a ocorrência de um problema, pode ter consequências em vários produtos da mesma linha de produtos, logo a importância de um mecanismo de gestão de risco eficaz é reforçada.

Ferramentas

Para a Framework, um ambiente integrado de desenvolvimento orientado a linhas de produtos deve reunir as seguintes capacidades:

- Suportar um processo de desenvolvimento baseado em linha de produtos;
- Deve assegurar a integridade entre as características dos produtos considerados pelo âmbito e as características dos produtos gerados;
- Deve ser capaz de representar os aspectos comuns e variáveis em todas as perspectivas em que façam sentido, por exemplo, um aspecto variável da arquitectura deve estar reflectido nos requisitos;
- Deve gerir as alterações, mantendo a capacidade de regressão de versões de componentes reutilizáveis e produtos;
- Deve disponibilizar um processo de desenvolvimento baseado na arquitectura base, fornecendo ao *developer* do produto as várias perspectivas necessárias para especificar a solução;
- Permitir a construção de produtos a partir da especificação das suas características desejáveis;

- Suportar aspectos de gestão técnica e negócio, como por exemplo disponibilizar métricas de aferição de desempenho;
- Ser evolutiva e flexível, por exemplo, deve permitir a incorporação de novas tecnologias ou novas abordagens de produção.

Estas capacidades estão tipicamente distribuídas entre várias aplicações.

ÁREAS DE PRÁTICA DE GESTÃO ORGANIZACIONAL

Construção de caso de negócio

Um *business case* é uma ferramenta de apoio à decisão que simula um determinado cenário de negócio com o objectivo de facilitar a previsão de cenários futuros. Tal como os restantes artefactos, também um *business case* pode ser reutilizado, e.g. um business case para avaliação da construção de um activo reutilizável.

No âmbito de um desenvolvimento orientado a uma linha de produtos, um *business case* é utilizado para justificar o esforço necessário, i.e., para garantir se uma SPL é ou não viável assegurando que os recursos necessários são compensados pelos resultados esperados. Para além da justificação do esforço, um *business case*, pode ser utilizado para avaliar a inclusão, ou não, de um produto na linha de produtos. A especificação de um produto pode não encaixar totalmente no âmbito definido para a linha de produtos, no entanto, a sua produção pode/deve ser equacionada, questionando até que ponto se justifica aumentar o âmbito da linha e considerar a construção das características adicionais como activos reutilizáveis.

Gestão do Relacionamento com o Cliente

A relação entre o cliente e a equipa de desenvolvimento é um factor chave para o sucesso de qualquer projecto. As expectativas devem estar alinhadas, a colaboração deve ser incentivada, os riscos mitigados e os problemas resolvidos. Numa SPL os desafios são partilhados com a abordagem tradicional, o que muda é a perspectiva. Passamos de uma óptica "1 cliente, 1 produto" para uma lógica de "x clientes, x produtos".

Numa SPL, a solicitação de uma funcionalidade dá origem à natural avaliação da sua executabilidade – semelhante à abordagem tradicional – mas também ao interesse de englobar essa funcionalidade em função dos objectivos e do futuro da linha de produtos.

O desenvolvimento orientado a linhas de produtos deve ser explicado ao cliente, i.e., o corpo comercial deve apresentar a abordagem de desenvolvimento e enaltecer os ganhos - para o cliente, e.g. qualidade da solução, tempo de desenvolvimento - daí provenientes. A abordagem deve orientar o cliente no sentido do enquadramento das suas necessidades no âmbito da família de produtos; a solução deve ser negociada com o cliente em função de um leque limitado de opções, alertando-o para o facto de que funcionalidades que requeiram desenvolvimento adicional terão consequências como o aumento do tempo de desenvolvimento ou custo da solução.

Desenvolver uma estratégia de aquisição

A dinâmica que caracteriza o contexto organizacional actual obriga as empresas a manterem estruturas flexíveis de forma a responder eficazmente às constantes solicitações do mercado. Este enquadramento favoreceu a massificação do recurso a terceiros para colmatar necessidades internas. Geralmente, esta opção toma uma de duas formas: a compra de (partes) de produtos/serviços para que sejam inseridos no processo de desenvolvimento (e.g. COTS) ou a contratação temporária de recursos.

O processo Análise Desenvolver, Comprar, Extrair, Subcontratar avalia as alternativas colocadas aquando da decisão de construção de um activo reutilizável. Quando a opção recai sobre comprar ou encomendar o desenvolvimento a uma entidade terceira, a organização cria uma dependência entre o seu processo produtivo (que controla directamente) e uma entidade externa cujas condições do resultado, apesar de contratualizado, não controla. Este factor não deve ser desprezado. A organização deve ter um plano de aquisição para mitigar os riscos associados a esta dependência directa com o exterior.

Outro aspecto associado à motivação da elaboração do plano de aquisição é diversidade de activos reutilizáveis existentes numa linha de produtos, o que, muitas vezes, resulta na necessidade da manutenção e gestão de uma complexa rede de fornecedores – comunicação, renegociação, aspectos contratuais, qualidade de serviço, etc.

Financiamento

A adopção de uma SPL requer investimento em formação, tecnologia, estudos de mercado, novos recursos, ferramentas, etc. O desenvolvimento tradicional, tipicamente, imputa directamente o investimento necessário ao cliente do projecto. Numa SPL existe um conjunto de elementos partilhados entre as várias soluções disponibilizadas. Essa partilha inviabiliza a imputação directa do custo a um projecto particular.

Esta área de prática pretende realçar a importância da estruturação do modelo de financiamento do investimento necessário para construir e manter um desenvolvimento orientado por uma linha de produtos.

Relativamente ao desenvolvimento dos activos reutilizáveis é importante que o modelo de financiamento considere as várias vertentes envolvidas: a análise e planeamento da família de produtos, o desenvolvimento de activos reutilizáveis, o desenvolvimento da infra-estrutura e a manutenção e evolução da linha de produtos.

Por outro lado, a própria produção de membros da família, i.e. a construção de produtos, acarreta custos específicos que não devem ser menosprezados. Além de uma solução poder envolver custos relativos a desenvolvimentos específicos, o modelo de financiamento deve contemplar o impacto financeiro na linha e sua manutenção.

Lançamento e institucionalização

A mudança de paradigma de desenvolvimento para uma SPL acarreta uma alteração organizacional e tecnológica profunda. Esta transição, pela sua dimensão, deve ser cuidadosamente gerida para garantir que a iniciativa tem os resultados esperados.

Uma linha de produtos é um projecto a longo prazo que envolve mudanças de comportamentos em todos os principais actores de um projecto de desenvolvimento de sistemas de informação. Não são "apenas" os produtos que passam a ser construídos com base em activos reutilizáveis, é o gestor de projecto que deve reestruturar a forma como estima os custos de cada programa de desenvolvimento, o arquitecto que passa da concepção de uma infra-estrutura com base em requisitos existentes para uma óptica de utilização a longo prazo ou ainda a equipa comercial que deve abordar um cliente de uma forma orientada à linha de produtos.

O lançamento de uma linha de produtos (ou extensão de existente) deve ser preparada num enquadramento que detalhará planos de acção e projectos piloto para todas as áreas afectadas pela mudança. Esta abordagem sistémica permitirá minimizar os riscos associados à transição. Por outro lado, a institucionalização, visa a incorporação da nova abordagem na cultura organizacional existente e a sua adopção como prática organizacional.

Análise de Mercado

A análise do mercado é um dos primeiros passos na construção de uma SPL. A caracterização da linha de produtos – a definição do âmbito, a determinação das metas a atingir, etc – é directamente influenciada pelos potenciais clientes, fornecedores e concorrentes, i.e. pelo mercado.

É no mercado que está a informação que permitirá optar ou abandonar a transição para SPL, bem como, o fundamento para a avaliação da sua continuidade e evolução. Assim, a Framework distingue esta prática como transversal a todo o ciclo de vida do programa que deve funcionar em directa articulação com todas as actividades que envolvam opções tácticas/estratégias – por exemplo, tanto na avaliação do desenvolvimento de um activo reutilizável, como na decisão de construção de um produto que extrapole o âmbito da linha de produtos, para além das especificidades técnicas, a componente "interesse para o mercado" deve ser considerada.

Operações

Uma operação é definida pela Framework como o elemento que interliga políticas e estratégias organizacionais, processos de negócios e planos de trabalho, num "corpo coerente e unificado de políticas e práticas". Podemos apresentar o principal objectivo desta área de prática como a definição da articulação que deve existir entre os componentes definidos pelas restantes áreas de prática.

Dada a complexidade de um programa SPL, a operacionalização deve ser devidamente validada, documentada e mantida. Um ponto é salientado pela Framework, não é possível generalizar esta actividade, i.e. a definição das operações está intimamente ligada com a

organização pois não é possível definir um plano de operações sem a conhecer detalhadamente – a quantidade e caracterização dos recursos existentes, a estrutura e cultura organizacional, o enquadramento no mercado, etc.

Planeamento Organizacional

O planeamento organizacional complementa o planeamento técnico. Este último foca o apoio ao desenvolvimento de um projecto enquanto o planeamento organizacional se coloca numa perspectiva de topo, assegurando a visão estratégica do programa.

Para além do planeamento estratégico genérico – e.g. planeamento da gestão de configurações, da formação e treino dos colaboradores, da gestão do risco - existem dois tipos de planos específicos que devem ser elaborados no desenvolvimento orientado a linhas de produtos: o plano de adopção de linha(s) de produtos e plano de financiamento dos activos reutilizáveis. O destaque dado a ambos os planos é justificado pelo grande impacto que têm no sucesso do programa.

O nível de topo atribuído a esta prática não deve significar a falta de ligação com a operacionalização, pelo contrário, o plano estratégico deve estabelecer os pontos de ligação com o plano operacional de forma a poderem ser estabelecidas métricas e objectivos de verificação da execução do plano organizacional.

Gestão do risco (perspectiva organizacional)

À semelhança do Planeamento Organizacional, esta área de prática coloca a gestão de risco numa perspectiva de topo, por oposição à Gestão do Risco (perspectiva técnica).

Uma linha de produtos envolve a coordenação de vários processos e equipas associadas a vários aspectos da linha de produtos ou a vários projectos distintos. Ainda que sobre o objectivo partilhado de contribuir para os objectivos da organização, os projectos têm muitas vezes objectivos/características opostas – e.g. a calendarização de um projecto pode prejudicar directamente outro. Esta relação de dependência cria potenciais riscos que transcendem a perspectiva de um projecto particular, e que por isso mesmo, devem ser mitigados por uma gestão de risco estratégica que assegure a visão da organização.

A Framework apresenta sete princípios para uma efectiva gestão de risco:

1. Favorecer a comunicação entre os intervenientes (Princípio Base)
2. Integrar a gestão de risco na operação (evitar que a gestão de risco seja vista como um acréscimo de trabalho)
3. Criar um ambiente saudável para trabalho em equipa (favorece a necessária comunicação)
4. Processo de gestão de risco contínuo (as mudanças constantes devem ser permanentemente avaliadas)
5. Incentivar o foco nos objectivos da organização em todos os intervenientes (por oposição ao foco exclusivo nos objectivos particulares do projecto)
6. Reforçar a importância da perspectiva organizacional de longo prazo
7. Assegurar a partilha de visão de produto, da direcção da organização.

Estruturação da Organização

À estruturação da organização cabe a tarefa de definir actores e papéis (perfis e responsabilidades) e designar recursos. Esta actividade também é, de forma mais ou menos formal, desempenhada nos projectos de desenvolvimento tradicional. Numa SPL, a diferença reside na criação de novos papéis e actores e na alteração da estratégia de estruturação orientada pela linha de produtos e não pelos projectos que desenvolvem.

De acordo com a Framework, uma estrutura organizacional para um programa SPL deve, pelo menos, assegurar a atribuição das seguintes tarefas:

1. Determinação da estratégia de produção
2. Determinação do âmbito da linha de produtos
3. Produção e manutenção da arquitectura da linha de produtos
4. Determinação dos requisitos da linha de produtos e dos seus membros
5. Desenho, produção e manutenção dos activos reutilizáveis (e respectivos planos de produção)
6. Avaliação da utilidade dos activos reutilizáveis
7. Acompanhamento da evolução dos activos reutilizáveis
8. Produção de produtos
9. Determinação dos processos a seguir (com aferição do seu cumprimento)
10. Manutenção do ambiente de desenvolvimento
11. Previsão de tendências, tecnologias, e outros aspectos que possam influenciar o futuro do programa

Previsão de alterações tecnológicas

Um programa SPL é estruturado para longo prazo pois só assim se obterão as economias de escala que justificam a produção de activos reutilizáveis. Uma das dificuldades que se coloca a qualquer programa de longo prazo, é a mudança constante do mercado. A actividade de análise de mercado endereça parte do problema. A outra parte é deixada para esta área de prática, a previsão de alterações tecnológicas.

A tecnologia é uma área em constante ebulição, tudo muda muito rapidamente. Para que uma linha de produtos não definhe é necessário manter uma actualização tecnológica constante, prevenindo eventuais antecipações de competidores. A Framework distingue dois tipos de previsão que devem ser efectuados:

- A previsão das tecnologias de apoio ao desenvolvimento: Que linguagens de programação estão a ser criadas? Quais as suas melhorias? Que novas ferramentas de desenvolvimento surgiram? Existem boas práticas que tomem o lugar de standard no mercado e que não estejamos a considerar?

- A previsão de novas soluções/tecnologias no mercado que representem potencial aos clientes: Existem sistemas de gestão de bases de dados mais eficientes? Existem standards novos? Existem plataformas de software mais interessantes?

Para além dos tipos de previsão apresentados são ainda apontados dois aspectos importantes desta actividade num programa SPL:

- Ainda que possa suscitar alguma comparação com a abordagem informal dessa actividade na perspectiva tradicional, numa linha de produtos, esta prática deve ser altamente estruturada e desempenhada numa base regular.
- A previsão tecnológica deve ser guiada de acordo com o âmbito da linha de produtos e considerando a totalidade da sua capacidade de variação, i.e. quanto mais alargado for o âmbito da linha, potencialmente maior será o espectro de tecnologias a cobrir.

Treino

O treino reforça as competências das pessoas, permite que adquiram novos conhecimentos e melhoem os que possuem. Nesta área de prática, os autores propõem a elaboração de um plano de treino que cubra os intervenientes no programa SPL.

A Framework refere a necessidade de compromisso das duas partes envolvidas:

- Quem constrói o plano de treino, deve assegurar o nível adequado de treino, i.e. deve realizar uma identificação das áreas de treino e deve preparar um conjunto consistente e ponderado de tarefas para propor ao elemento da equipa. Simultaneamente, deve assegurar a consistência entre os planos individuais e os planos de grupo.
- Por outro lado, quem realiza o treino, deve aceitar a importância do treino, assegurar o comprometimento com o plano, reportar resultados e sugerir alterações/melhoramentos.

À semelhança das restantes actividades de planeamento apresentadas noutras áreas de prática, o planeamento de treino deve também ser feito a nível estratégico, garantindo que os objectivos de negócio são os últimos visados.

8 ANEXO – RUP E OPENUP

RATIONAL UNIFIED PROCESS (RUP)

Documentação completa RUP: http://www.wthreex.com/rup/smallprojects/index.htm

O Rational Unified Process (RUP) é um processo prescritivo de desenvolvimento de sistemas, muitas vezes aplicado no desenvolvimento de sistemas baseados em tecnologias orientadas a objectos e/ou componentes (Ambler, 1999).

O RUP pretende ser um processo adaptável e moldável às particularidades de cada projecto. Existem no entanto alguns princípios base que não devem ser afectados nesse processo de adaptação (Probasco, 2000):

- Desenvolvimento da Visão doProjecto para caracterizar o sistema numa fase inicial e assim a garantir o futuro alinhamento com as expectativas dos clientes;
- Planeamento, porque só uma abordagem estruturada permitirá ultrapassar a complexidade crescente dos projectos actuais;
- Identificação e mitigação de riscos do projecto.
- Elaboração e análise de casos de negócio de forma a garantir o alinhamento com objectivos de negócio, e.g. ROI do projecto
- Enfoque na definição da melhor arquitectura para minimizar o esforço de desenvolvimento e o correcto funcionamento do sistema;
- Utilização de prototipagem rápida para validar opções durante o desenvolvimento;
- Avaliação e análise de resultados para detectar necessidades de ajustes;
- Motivação para a acomodar a mudança de forma controlada;
- Apoio e suporte ao utilizador com formação, testes e avaliação de satisfação

Previsão da adaptação do processo durante o ciclo de desenvolvimento pois eventuais mudanças poderão requerer alterações de processo.

Conforme apresentado na Figura 8-i, o ciclo de desenvolvimento considerado pelo RUP está organizado em duas dimensões principais: as fases e as disciplinas.

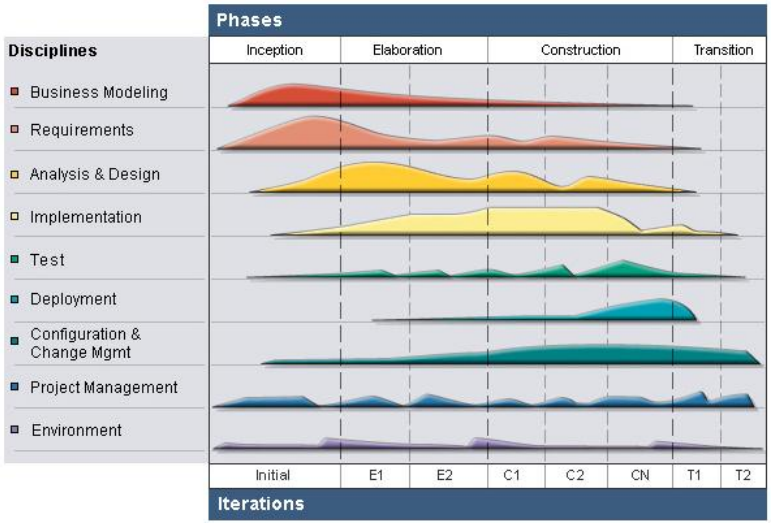

Figura 8-i – Ciclo de desenvolvimento RUP. Retirado de (Wthreex, 2011)

A nível temporal (plano horizontal), o RUP identifica quatro fases:

- Concepção *(Inception)*: Definição do âmbito do projecto e planeamento de actividades de alto-nível; Caracterização de alto nível do produto a desenvolver; Identificação de riscos; Avaliação de custos e orçamentação; Alinhamento das expectativas dos participantes com as condicionantes do projecto.
- Elaboração *(Elaboration)*: Refinamento de especificações de alto-nível; Mitigação de riscos; Definição da arquitectura do sistema e verificação do cumprimento dos requisitos; Desenvolvimento de protótipos que permitam aferir o cumprimento dos objectivos estipulados; Planeamento detalhado da fase de construção;
- Construção *(Construction)*: Construção do sistema focada na qualidade do produto final; Execução do plano de construção com o mínimo de custos possíveis; Realização de testes regulares e verificação do cumprimento dos objectivos; Detalhe e eventual alteração de especificações para garantir sincronização com implementação; Coordenação das equipas e respectivo trabalho envolvido;
- Transição *(Transition)*: Verificação do cumprimento das expectativas dos utilizadores; Realização de testes em ambiente de produção; Correcção (e documentação) de eventuais problemas; Implementação de pequenas alterações requisitadas pelos utilizadores; Formação e treino dos utilizadores; Elaboração e distribuição de documentação de apoio à utilização; Obtenção e validação formal de cliente sobre adequabilidade do sistema construído;

Cada fase define um *milestone* com os objectivos propostos para considerar a fase terminada (R. S. Corporation, 2005). Observando a complexidade normalmente envolvida nestes tipos de projectos, o RUP propõe uma estratégia *divide&conquer* para o cumprimento desses objectivos, sugerindo a divisão das fases em iterações bem delineadas e planeadas de acordo com as características concretas do projecto em causa.

Fase	**Principal objectivo**
Concepção	Reunir consenso entre todos os envolvidos sobre os objectivos do projecto

Elaboração	Criar uma arquitectura do sistema robusta para fornecer uma base estável para o esforço da fase de construção
Construção	Esclarecer eventuais requisitos em falta e concluir desenvolvimento com base na arquitectura definida na fase de Elaboração
Transição	Garantir a correcta disponibilização do sistema aos seus utilizadores finais

Tabela 8-i – Principal objectivo de cada fase RUP. Baseado em (Wthreex, 2011)

Para alcançar os objectivos propostos para cada fase, para além da orientação iterativa e incremental proposta, o RUP identifica, numa perspectiva operacional através da definição de um conjunto de disciplinas e actividades que devem ser executadas. São propostas nove disciplinas base que aqui são apresentadas com a identificação do seu objectivo principal:

- Modelação de Negócio *(Business Modeling)* - compilação da informação necessária para compreender a organização alvo;
- Requisitos *(Requirements)* - caracterização do sistema a construir para endereçar o problema visado;
- Análise e Desenho *(Analisys and Design)* – especificação e modelação detalhada do sistema a desenvolver de acordo as características (requisitos) identificados;
- Desenvolvimento *(Development)* – construção do sistema em sintonia com a especificação definida;
- Teste *(Test)* – avaliação da qualidade do produto e verificação do funcionamento correcto do sistema e alinhamento com os requisitos identificados;
- *Deployment* – passagem do sistema de ambiente desenvolvimento para ambiente de produção, disponibilizando o apoio necessário aos utilizadores.
- Gestão de Configurações e Mudança *(Change Management and Configuration)* – gestão harmoniosa das alterações existentes (e dos impactos resultantes), assegurando a repercussão dessas modificações em todos os artefactos.
- Gestão de Projecto *(Project Management)* –gestão e planeamento do projecto por forma a correcta, e controlada, execução da abordagem definida.
- Ambiente *(Environment)* – Adaptação do processo às características do projecto, garantindo ferramentas e conhecimento adequado.

A contribuição de cada disciplina para cada fase é identificada pelo gráfico respectivo (ver Figura 8-i). Por exemplo, a disciplina de Modelação de Negócio tem uma elevada contribuição na fase de Concepção e quase nenhuma na fase de Transição.

Para facilitar a atribuição de responsabilidades e a alocação de tarefas, o RUP define uma hierarquia de papéis que devem ser atribuídos a todos os participantes do projecto.

Analysts	***Developers***	***Testers***	***Production & Support***	***Managers***	***General***
Requirements Specifier	*Database Designer*	*Test Analyst*	*Process Engineer*	*Change Control Manager*	*Any Role*

Stakeholder	*Designer*	*Test Manager*	*System Administrator*	*Configuration Manager*	*Review Coordinator*
System Analyst	*Implementer*	*Tester*		*Management Reviewer*	*Reviewer*
	Integrator			*Project Manager*	*Stakeholder*
	Software Architect			*System Administrator*	*Technical Reviewer*
	User-Interface Designer			*Test Manager*	

Tabela 8-ii – Hierarquia de Papéis RUP

O RUP é uma metodologia extremamente detalhada, disponibilizando para cada disciplina um conjunto de fluxos de trabalho, com condições de entrada e de saída, associando a cada actividade os papéis envolvidos, com as responsabilidades respectivas, e artefactos a produzir.

OPEN UNIFIED PROCESS (OPENUP)

Documentação completa OpenUP: http://epf.eclipse.org/wikis/openup/

Em 2001 surgiu um movimento constituído por um conjunto de profissionais TI que defendia uma alternativa à abordagem formal de desenvolvimento. Os fundamentos dessa nova proposta foram materializados num manifesto (*Agile Manifesto)* que juntava 12 princípios básicos para uma abordagem ágil de desenvolvimento (Beck et al., 2001):

1. Satisfação do cliente como prioridade máxima, assegurando entregas de *software* útil (cujo valor seja facilmente identificado pelo cliente) em fases iniciais do desenvolvimento;
2. Aceitação das solicitações de requisitos: o modelo de desenvolvimento deve reconhecer e contemplar essas necessidades;
3. Entrega de *software* funcional em espaços temporais curtos (na ordem de poucas semanas);
4. Participação próxima do cliente ao longo do processo;
5. Promoção de um ambiente de desenvolvimento com pessoas motivadas, em cujo trabalho se deve confiar;
6. Recurso a reuniões cara-a-cara com frequência (forma ideal de passar informação);
7. Garantia que a principal métrica de progresso deve ser o *software* utilizável (para o cliente)
8. Promoção do desenvolvimento sustentável (os envolvidos devem manter o ritmo de participação constante)
9. Agilidade suportada pela excelência técnica e boas concepções de sistemas;
10. Foco na simplicidade;
11. Confiar na auto-organização das equipas;
12. Promoção de ajustamentos regulares às opções tomadas;

O Open Unified Process (OpenUp) é uma metodologia ágil (Process, 2004) com muitas "inspirações" no RUP. Este processo baseia-se num modelo elaborado pela IBM, o Basic Unified Process (BUP), que em 2006 foi transferido para a comunidade open-source Eclipse

(mais concretamente ao núcleo Eclipse Process Framework - EPF), que adoptou o termo OpenUp e que actualmente assegura a sua manutenção e disponibiliza ferramentas de suporte.

Tal como o RUP, o OpenUp desenvolve-se em fases e iterações (Project, 2011). A Tabela 8-iii indica o principal objectivo associado a cada fase.

Fase	Principal objectivo
Concepção	Avaliar o custo-benefício do projecto e decidir sobre continuação ou abandono
Elaboração	Obter acordo sobre os requisitos e sobre a arquitectura da solução
Construção	Ter o sistema desenvolvido e documentação de suporte criada
Transição	Garantir a aceitação do sistema por parte do cliente

Tabela 8-iii – Principal objectivo da cada fase OpenUP. Baseado em (Project, 2011)

Cada fase terá iterações que deverão ser curtas (poucas semanas) e pensadas para que o seu resultado seja um sistema ou acréscimo de funcionalidades úteis para o cliente. A metodologia aponta ainda para um grão mais fino de segmentação do ciclo de desenvolvimento, o que apelidam de micro incremento (*micro-increment*) relativo à contribuição individual de cada elemento da equipa do projecto – o que, tipicamente diz respeito a algumas horas/dias de trabalho. O micro incremento é a métrica de progresso individual que, devidamente integrada no conjunto de contribuições, permitirá aferir a evolução da iteração e consequentemente do projecto.

Como apresentado na Figura 8-ii, à semelhança do RUP, o OpenUp mantém a estruturação do ciclo de desenvolvimento do projecto em quatro fases, sensivelmente com os mesmos significados.

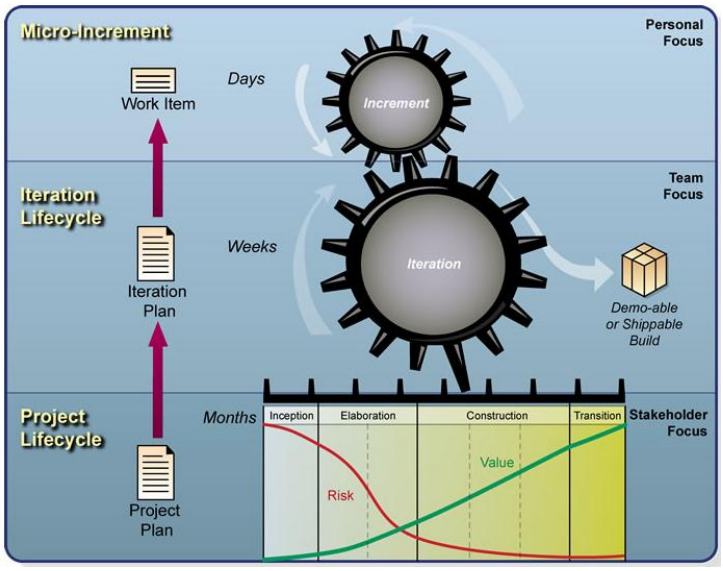

Figura 8-ii – Modelo OpenUp. Retirado de (Project, 2011)

O OpenUp também define disciplinas que devem ser realizadas durante o projecto, mas, ao contrário do RUP, no OpenUp, a contribuição de cada disciplina para cada fase não é explícita. A ideia é libertar o projecto de processos rígidos e momentos pré-estabelecidos de

concretização de tarefas. As disciplinas consideradas no OpenUp são as seguintes (com significados e objectivos semelhantes ao RUP) (Project, 2011):

1. *Requisitos (Requirements)*
2. *Arquitectura (Architecture)*
3. *Desenvolvimento (Development)*
4. *Testes (Test)*
5. *Gestão de Projecto (Project Management)*
6. *Gestão de Configurações e Mudança (Configuration and Change Management)*

Mantém-se o alinhamento entre RUP e o OpenUp embora neste último não sejam consideradas as disciplinas de Modelação de Negócio, *Deployment* e Ambiente. Também no OpenUp cada disciplina tem associado um fluxo de trabalho para a sua execução.

Na definição dos papéis, o OpenUp oferece uma classificação bastante mais simplificada do que o RUP. O OpenUp comtempla sete tipos de papéis que devem ser desempenhados durante o projecto (Process, 2004):

- Arquitecto *(Architect)*: responsável pela definição da arquitectura do sistema
- Gestor de Projecto *(Project Manager)*: responsável pelo planeamento e correcta execução do projecto
- Analista *(Analyst)*: responsável por representar o cliente e as suas preocupações
- *Tester*: responsável pela concepção e realização dos testes necessários
- Qualquer Papel *(Any Role)*: papel utilizado para representar um qualquer participante do projecto
- *Developer*: reúne as responsabilidades de desenho e implementação do sistema
- *Stakeholder*: simboliza os interessados nos resultados do projecto.

9 ANEXO – PROCESSO DE INTEGRAÇÃO SPLUP

DEFINIÇÃO DO PROCESSO DE ENGENHARIA DE DOMÍNIO (BASE RUP)

1º Definição de disciplinas Engª Domínio SPLUP (adaptação das disciplinas RUP)

Na Tabela 9-i apresentamos a relação que propomos para mapear as disciplinas RUP e o seu significado num cenário SPL. Evitámos transfigurar as disciplinas tentando, sempre que possível, manter o espírito original de cada uma. A realidade é que o mapeamento acabou por ser bastante escorreito, residindo a maior variação na passagem de desenvolvimento de uma aplicação para a construção de uma infra-estrutura de produção de aplicações.

Disciplina RUP	Significado da disciplina na SPLUP
Modelação de Negócio (Análise de Domínio)	Estudo do domínio/negócio ao qual a linha de produtos se destinará. Implicará realizar um estudo técnico sobre o domínio, definir os seus limites e características. Nota: Única disciplina em que optámos por alterar a designação original para "Análise de Domínio" por considerarmos que seria um termo mais adequado.
Requisitos	Definição dos requisitos da linha de produtos, e.g. identificação dos requisitos comuns e variáveis dos membros da família ou determinação dos requisitos da infra-estrutura de produção e do processo de produção.
Análise e Desenho	Análise dos requisitos previamente definidos e respectiva concretização, em particular, na definição de uma arquitectura comum para os membros da família, na especificação de componentes variáveis e infra-estrutura de produção.
Implementação	Implementação da infra-estrutura de produção, da arquitectura definida para os membros da família, dos componentes reutilizáveis ou de qualquer outro elemento que participe no processo de produção.
Testes	Testes dos elementos criados, inclusive testes à arquitectura comum, a componentes reutilizáveis, a diferentes combinações de configurações (exemplos de aplicações da família) ou ainda testes às ferramentas de apoio à produção.
Deployment	Disponibilização dos meios de produção, i.e. passagem da infra-estrutura de produção à engenharia de aplicação. Inclui, por exemplo, preparar documentação no processo, nas ferramentas ou na abordagem a seguir.
Gestão de Projecto	Disciplina partilhada com engenharia de aplicação. Abrange responsabilidades que passam por (para LP e Projectos de aplicação): garantir a disponibilização atempada dos recursos necessários para execução do projecto, planeamento, controlar e mitigar riscos ou garantir o constante alinhamento entre as metas organizacionais e os objectivos da LP.
Gestão de Configurações e Mudança	Disciplina partilhada com engenharia de aplicação. Para além das mudanças respeitantes ao dinamismo do domínio, será responsável por gerir as solicitações de novas características para os produtos, o que se traduzirá numa avaliação de custo-benefício para a LP. A gestão de configuração numa SPL é bastante complexa pois terá de gerir um grande conjunto de elementos – arquitectura base, componentes, configurações, processos, modelos, aplicações geradas etc - durante um alargado período de tempo.
Ambiente	Disciplina partilhada com engenharia de aplicação. De acordo com o RUP, a disciplina ambiente pretende "oferecer à organização o ambiente de desenvolvimento de *software* — processos e ferramentas — que dará suporte à equipa de desenvolvimento". Este propósito é mantido na SPLUP mas paralelamente, ganha uma interpretação mais abrangente traduzida na atribuição de uma nova responsabilidade: promover e garantir a correcta reutilização, controlando e monitorizando todo o processo por forma a limitar o esforço e assegurar que não existe desenvolvimento desnecessário.

Tabela 9-i – Disciplinas SPLUP

2º Enquadramento FSPLP com as disciplinas Engª Domínio SPLUP

As 29 áreas de prática FSPLP estão organizadas segundo "macro áreas": gestão organizacional, gestão técnica e engenharia de *software*. Não existe qualquer identificação da localização temporal de cada uma.

A Tabela 9-ii resume um possível mapeamento entre as disciplinas SPLUP e as áreas de prática. O relacionamento não foi directo. O âmbito alargado das áreas de prática tornava possível associar várias áreas de prática a todas as disciplinas. Assim, em algumas áreas, vimo-nos forçados a efectuar as associações com base no que considerámos ser o núcleo da área, não atentando a alguns aspectos menos relevantes. Mesmo com esta limitação existem áreas associadas a mais do que uma disciplina mas acreditamos que o processo tenha mantido o intuito inicial de cada uma.

Disciplina SPLUP	Tipo de Área	Área de Prática FSPLP
Análise de Domínio	Gestão Técnica	Definição do âmbito
Análise de Domínio	Engenharia Software	Compreensão dos domínios relevantes
Análise de Domínio	Gestão Organizacional	Análise de mercado
Análise de Domínio	Gestão Organizacional	Previsão de alterações tecnológicas
Análise de Domínio	Engenharia Software	Exploração de activos existentes
Requisitos	Engenharia Software	Definição da arquitectura
Requisitos	Engenharia Software	Engenharia de requisitos
Requisitos	Engenharia Software	Exploração de activos existentes
Análise e Desenho	Engenharia Software	Definição da arquitectura
Análise e Desenho	Engenharia Software	Exploração de activos existentes
Análise e Desenho	Engenharia Software	Avaliação da arquitectura
Implementação	Engenharia Software	Desenvolvimento de componentes
Implementação	Engenharia Software	Integração de componentes de software
Testes	Engenharia Software	Avaliação da arquitectura
Testes	Engenharia Software	Testes
Deployment	Gestão Técnica	Suporte de ferramentas
Deployment	Gestão Técnica	Planeamento técnico
Deployment	Gestão Organizacional	Treino
Gestão de Projecto	Gestão Técnica	Análise Desenvolver, Comprar, Extrair, Subcontratar
Gestão de Projecto	Gestão Técnica	Medição e Monitorização
Gestão de Projecto	Gestão Técnica	Definição do âmbito
Gestão de Projecto	Gestão Organizacional	Gestão do risco (perspectiva organizacional)
Gestão de Projecto	Gestão Técnica	Gestão do risco (perspectiva técnica)
Gestão de Projecto	Gestão Organizacional	Construção de um caso de negócio
Gestão de Projecto	Gestão Organizacional	Financiamento
Gestão de Projecto	Gestão Organizacional	Lançamento e institucionalização
Gestão de Projecto	Gestão Organizacional	Planeamento organizacional
Gestão de Projecto	Gestão Organizacional	Estruturação da organização
Gestão de Projecto	Gestão Organizacional	Desenvolvimento de uma estratégia de aquisição
Gestão de Projecto	Gestão Organizacional	Gestão do relacionamento com o cliente
Gestão de Conf. e Mudança	Gestão Técnica	Gestão de configurações
Gestão de Conf. e Mudança	Gestão Organizacional	Gestão do relacionamento com o cliente
Ambiente	Gestão Técnica	Disciplina de processo
Ambiente	Gestão Técnica	Suporte de ferramentas
Ambiente	Engenharia Software	Utilização de software disponível externamente
Ambiente	Gestão Técnica	Planeamento técnico
Ambiente	Gestão Organizacional	Definição da operacionalização
Ambiente	Gestão Organizacional	Gestão do relacionamento com o cliente

Tabela 9-ii – Mapeamento disciplinas Engª Domínio SPLUP e Áreas de Prática FSPLP

3º Enquadramento Software Factories e Engª Domínio SPLUP

As Software Factories são apresentadas como um paradigma de desenvolvimento focando os aspectos genéricos que os processos que nela se baseiam poderão seguir. Não define metodologia, nem tão pouco actividades a desempenhar.

A contribuição das Software Factories estaria na articulação de técnicas e metodologias, em particular, na conjugação do paradigma de desenvolvimento de linhas de produtos, com desenvolvimento orientado a modelos e técnicas de automatização. A forma como o SPLUP integra as tecnologias segue o padrão preconizado pela Software Factories.

4º Criação do conteúdo do processo de Engª Domínio SPLUP

Por último, utilizando o enquadramento que acabámos de apresentar, definimos o conteúdo de cada disciplina SPLUP, identificando o seu intuito de forma detalhada, especificando para cada disciplina um fluxo de trabalho, papéis a desempenhar e artefactos a produzir.

DEFINIÇÃO DO PROCESSO DE ENGENHARIA DE APLICAÇÃO (BASE OPENUP)

1º Definição de disciplinas Engª Aplicação SPLUP

A engenharia de aplicação é um processo de desenvolvimento muito particular pois o formato em que é executado depende do plano e infra-estrutura de produção estipulado pela engenharia de domínio. Acresce ainda, que dada a forte componente de automatização, os projectos são tipicamente muito curtos (na ordem de dias) e resumem-se, na essência, à análise do problema do cliente e à produção (automática ou semiautomática) da solução.

As particularidades da engenharia de aplicação impossibilitaram a criação de disciplinas baseadas na adaptação das consideradas no OpenUP. Todavia, na Tabela 9-iii é apresentado o conjunto de disciplinas genéricas consideradas no OpenUp independentemente dos mecanismos de produção disponibilizados pela engenharia de domínio.

Disciplina SPLUP	Contextualização
Análise do Problema	Análise do contexto do Cliente e enquadramento na família de produtos,negociando uma solução em função das características da LP e preparação da especificação de produção do produto.
Produção e Testes	Modelação da aplicação e produção (automática, semiautomática ou manual) da aplicação. Execução de testes à aplicação gerada.
Desenvolvimento Adicional	Implementação de algum componente que não esteja considerado na LP
***Deployment* Aplicação**	Disponibilização da solução ao Cliente, acompanhado de testes em ambiente de produção e preparação de documentação de suporte

Tabela 9-iii – Disciplinas SPLUP (Exclusivas da Engenharia de Aplicação)

Os motivos da escolha do OpenUp para "inspiração base" da engenharia de aplicação foram diferentes dos associados ao RUP na engenharia de domínio. A grande contribuição não está no apoio à definição de disciplinas e fluxos de trabalho, mas na partilha de valores (e.g. entregas rápidas e de valor acrescentado ou forte participação do Cliente) e semelhança de formato do trabalho a realizar (e.g. iterações de duração muito curta ou auto-organização de equipas).

2º Enquadramento FSPLP com as disciplinas Engª Aplicação SPLUP

À semelhança do processo de engenharia de domínio também a engenharia de aplicação se baseará no conhecimento documentado nas áreas de prática FSPLP. O processo é mais simples do que a Engenharia de domínio por isso o mapeamento não cobriu todas as áreas de prática.

Disciplina SPLUP	Tipo de Área	Área de Prática FSPLP
Análise do Problema	Gestão Organizacional	Gestão do relacionamento com o cliente
Análise do Problema	Engenharia de Software	Compreensão dos domínios relevantes
Desenvolvimento Adicional	Engenharia de Software	Exploração de activos existentes
Desenvolvimento Adicional	Engenharia de Software	Desenvolvimento de componentes
Desenvolvimento Adicional	Gestão Técnica	Testes
Produção e Testes	Gestão Técnica	Disciplina de processo
Produção e Testes	Gestão Técnica	Testes
Produção e Testes	Engenharia de Software	Integração de componentes de software
Deployment Aplicação	Gestão Técnica	Testes

Tabela 9-iv - Mapeamento disciplinas Engª Aplicação SPLUP e Áreas de Prática FSPLP

3º Enquadramento Software Factories e Engª Aplicação SPLUP

O enquadramento das Software Factories na Engenharia de Aplicação é semelhante ao contexto apresentado na Engenharia de Domínio. A Engenharia de aplicação trabalhará com o mesmo conjunto de técnicas e tecnologias aplicadas na construção da infra-estrutura de produção mas numa óptica de consumo.

4º Criação do conteúdo do processo de Engª Aplicação SPLUP

A abordagem seguida para a criação do conteúdos das disciplinas deste processo foi idêntica à seguida na definição do processo da engenharia de domínio, isto é, para cada disciplina identificámos os seus objectivos, associámos fluxos de trabalho e papéis a cumprir e definimos artefactos a produzir.

10 ANEXO – FLUXOS DE TRABALHO SPLUP

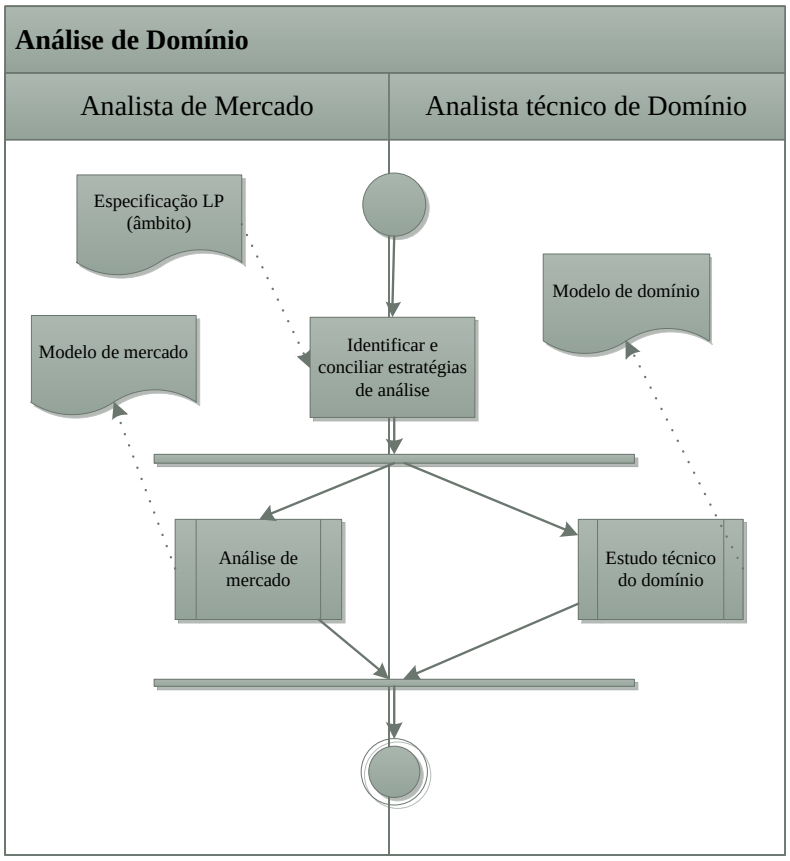

Figura 10-i –Análise de Domínio

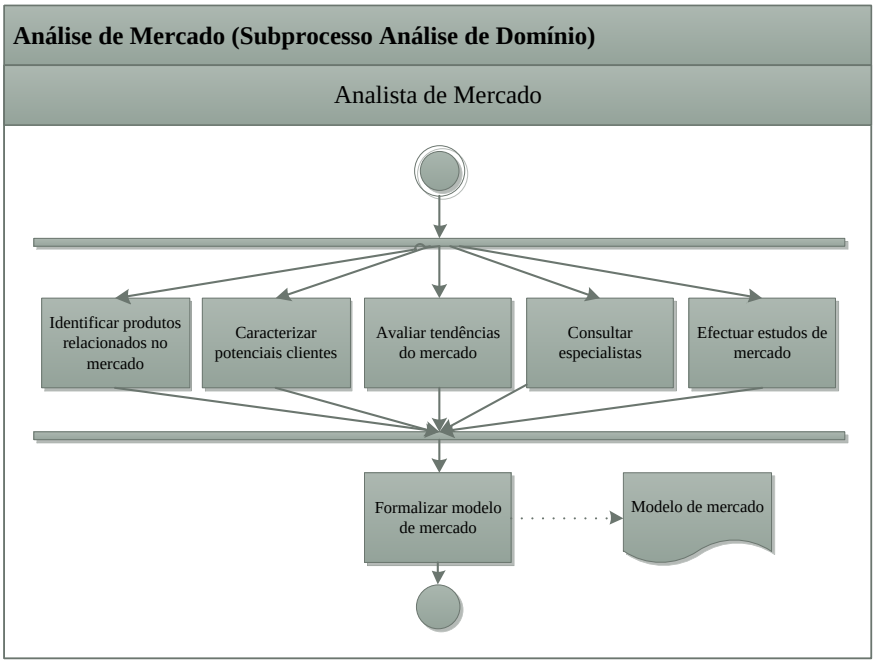

Figura 10-ii – Análise de Mercado (Subprocesso Análise de Domínio)

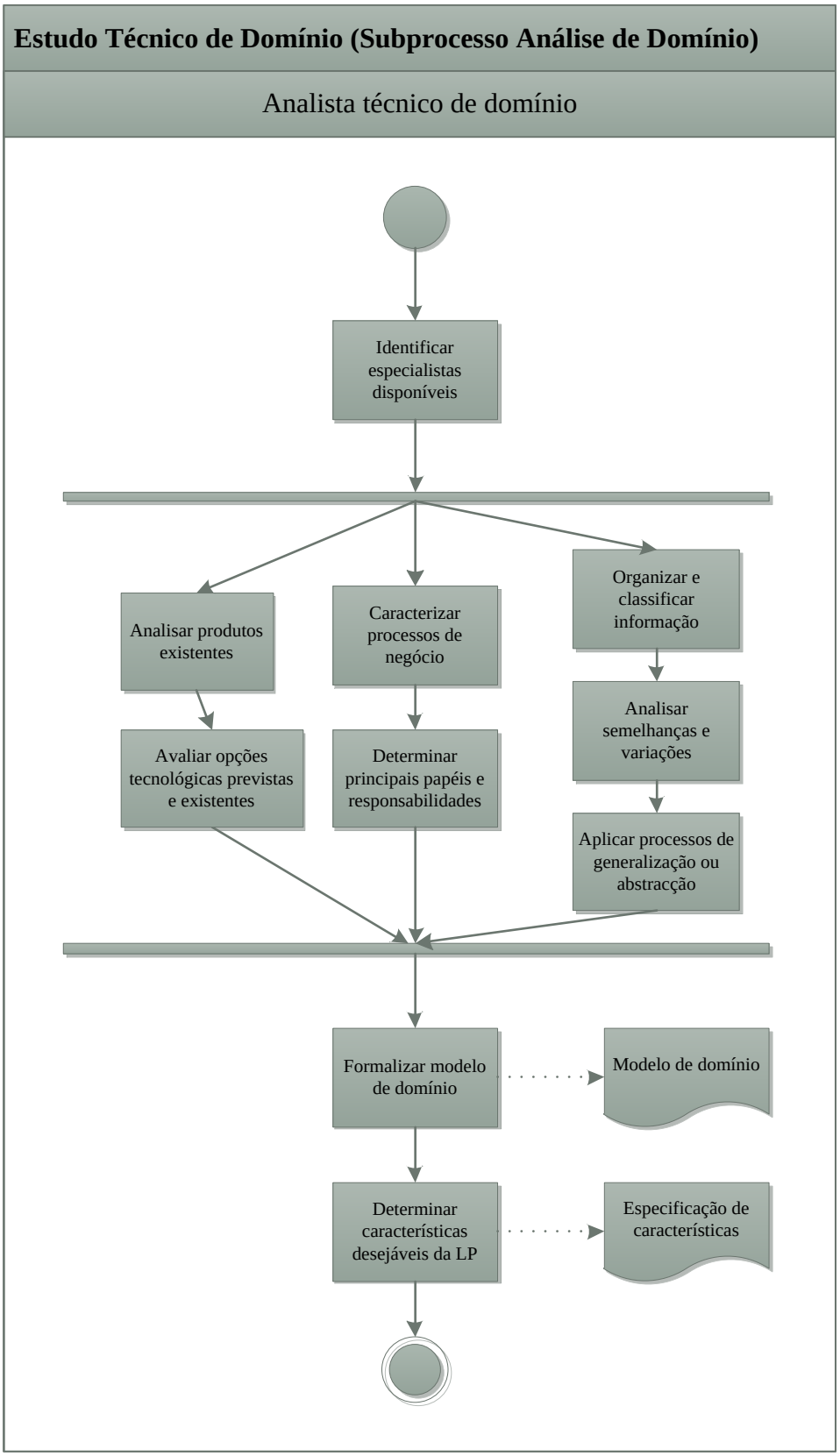

Figura 10-iii – Estudo técnico de domínio (Subprocesso Análise de Domínio)

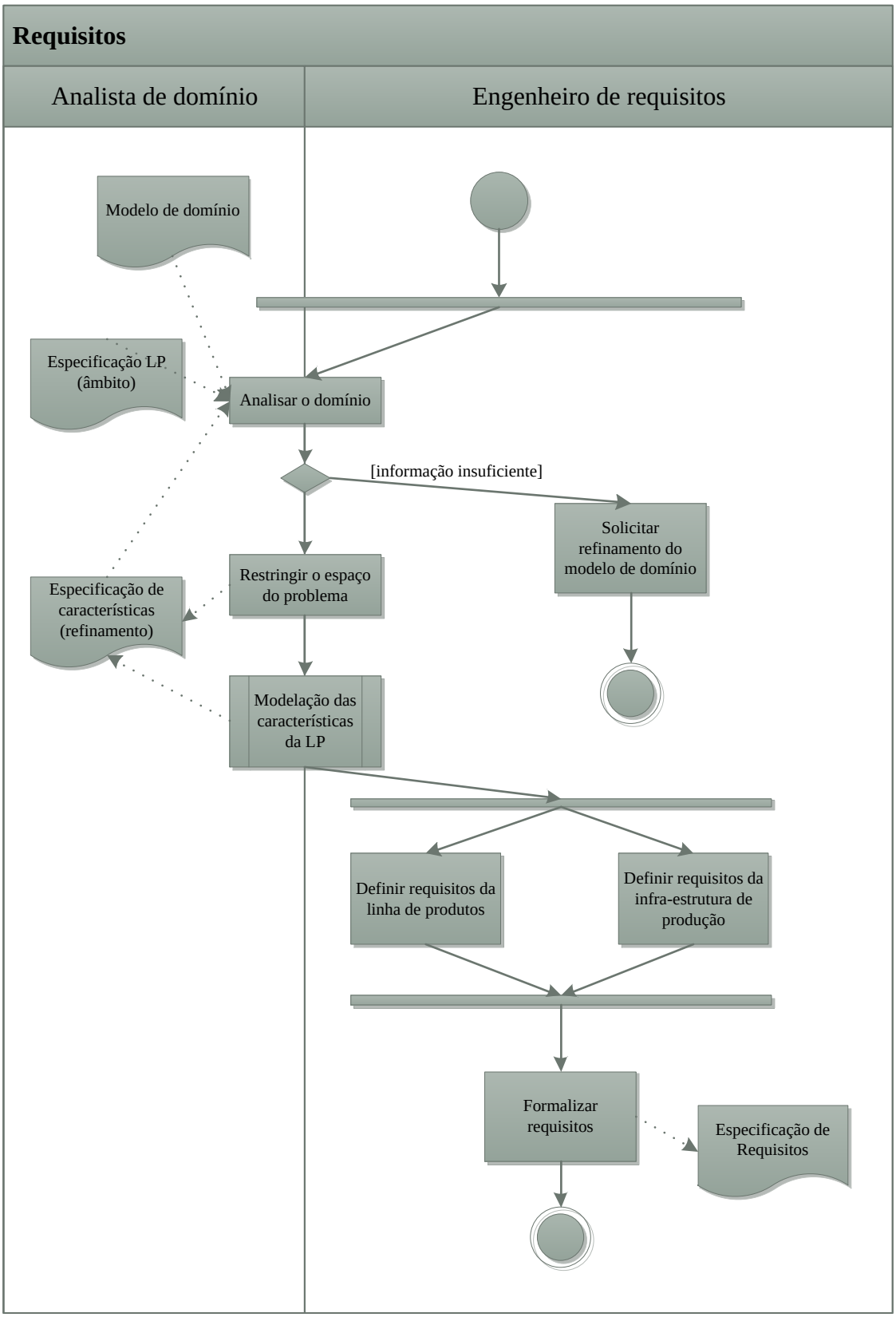

Figura 10-iv - Requisitos

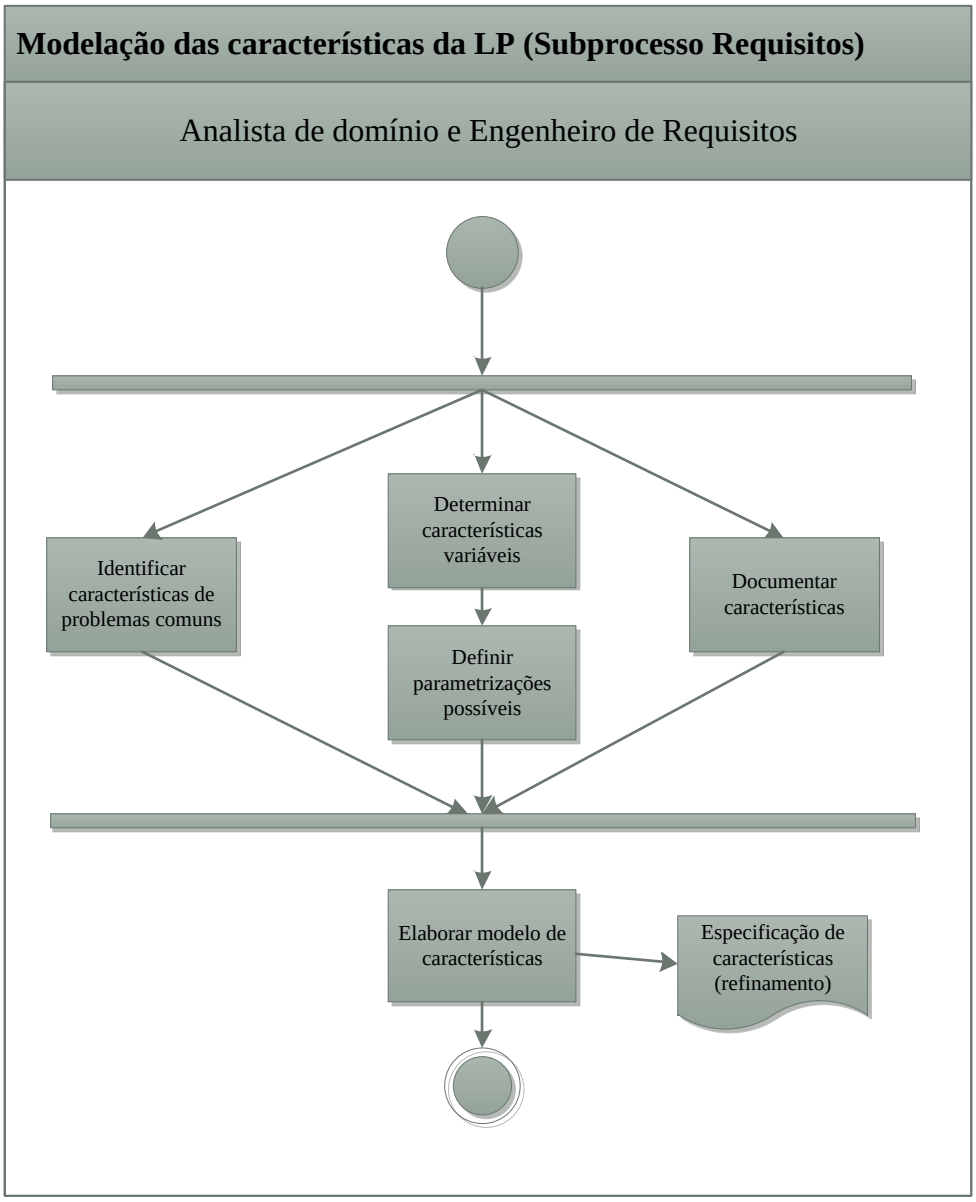

Figura 10-v – Modelação das características da LP (Subprocesso Requisitos)

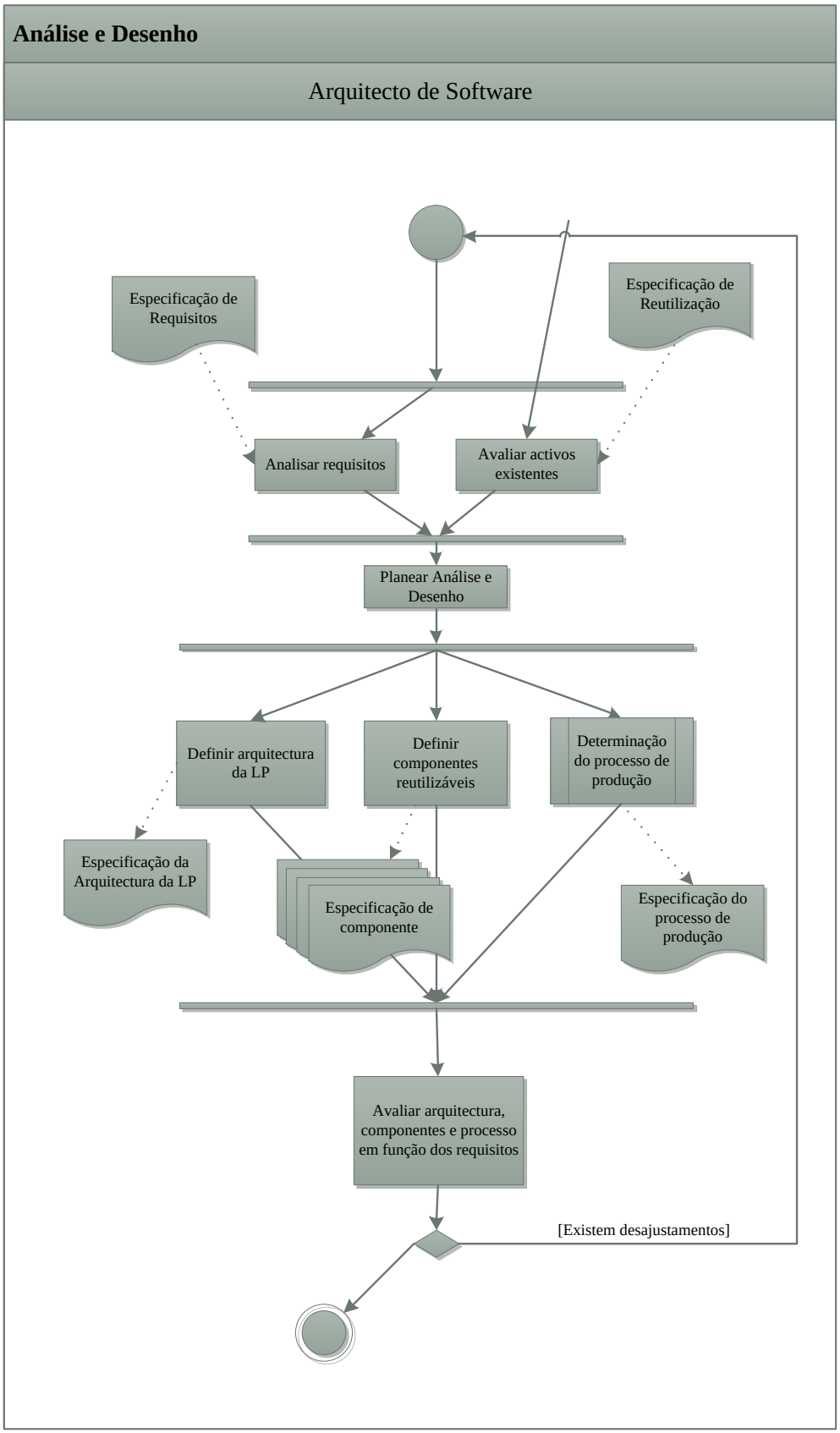

Figura 10-vi – Análise e Desenho

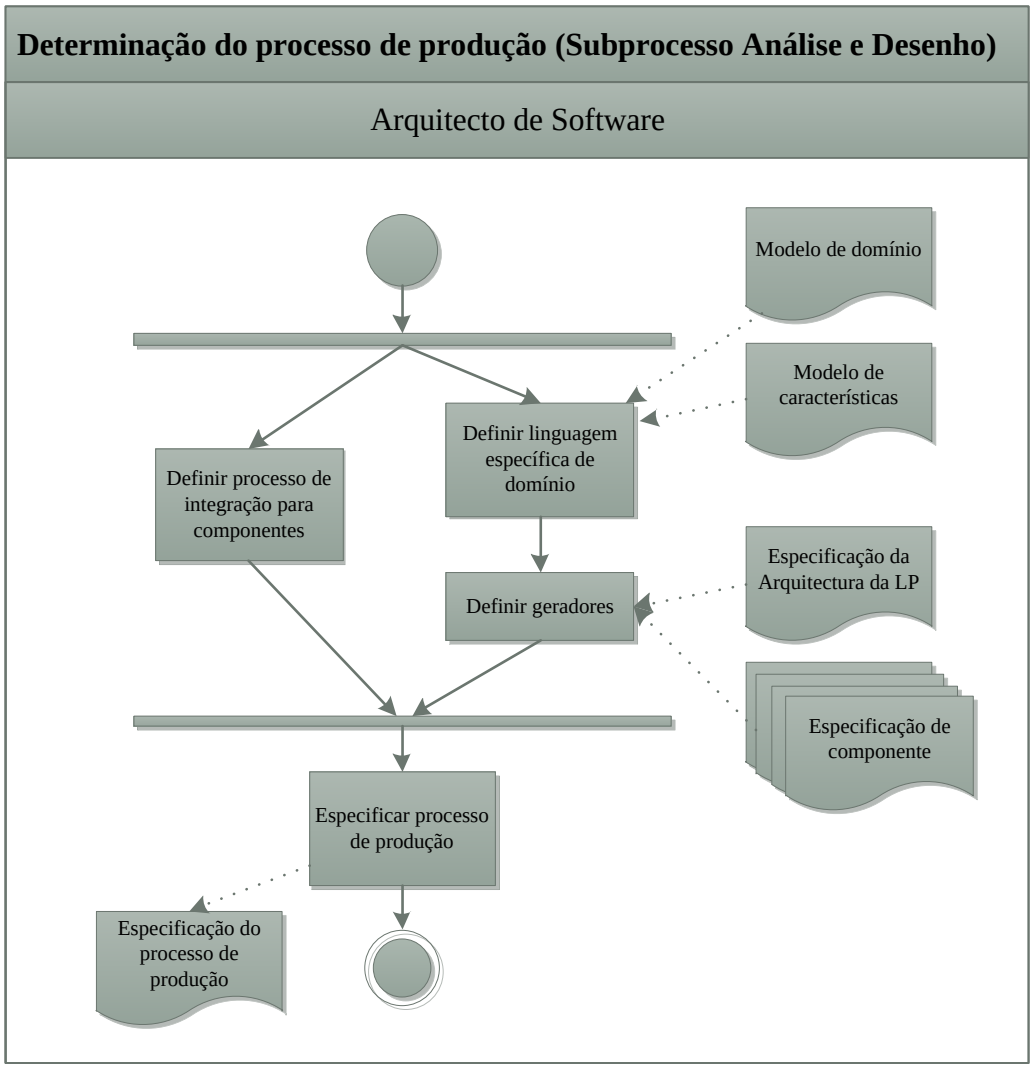

Figura 10-vii - Determinação do processo de produção (Subprocesso Análise e Desenho)

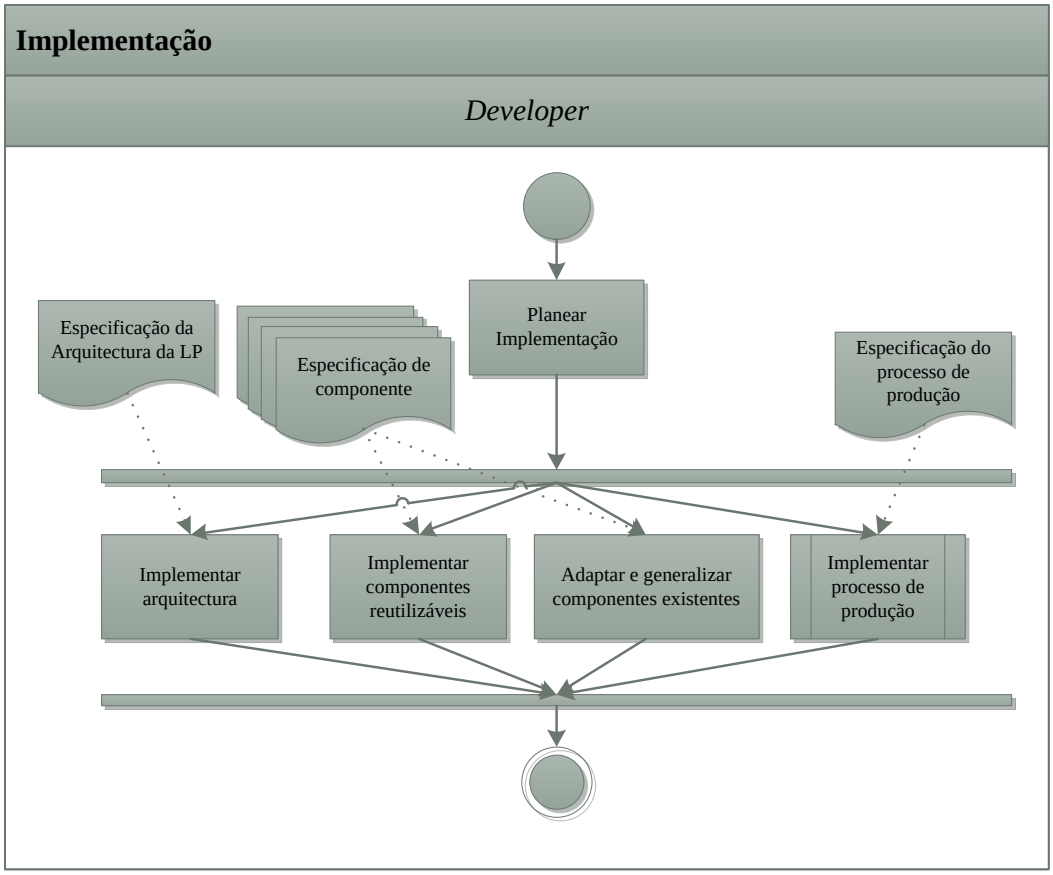

Figura 10-viii – Implementação

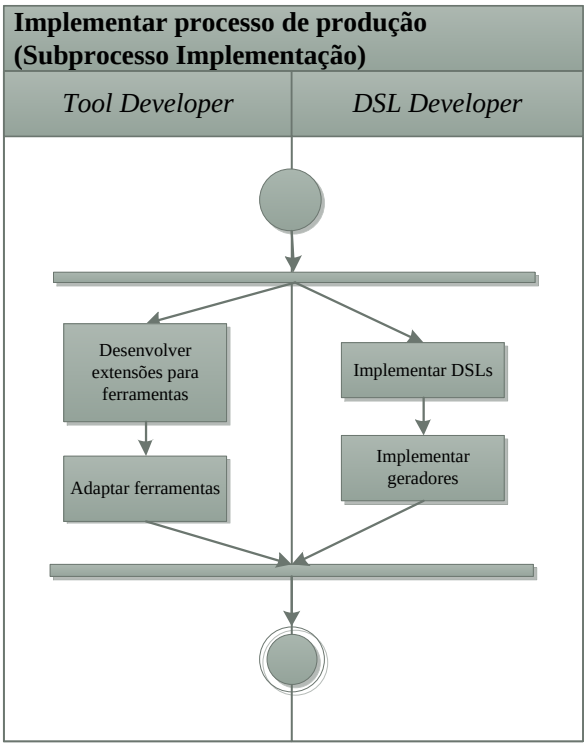

Figura 10-ix - Implementar processo de produção (Subprocesso Implementação)

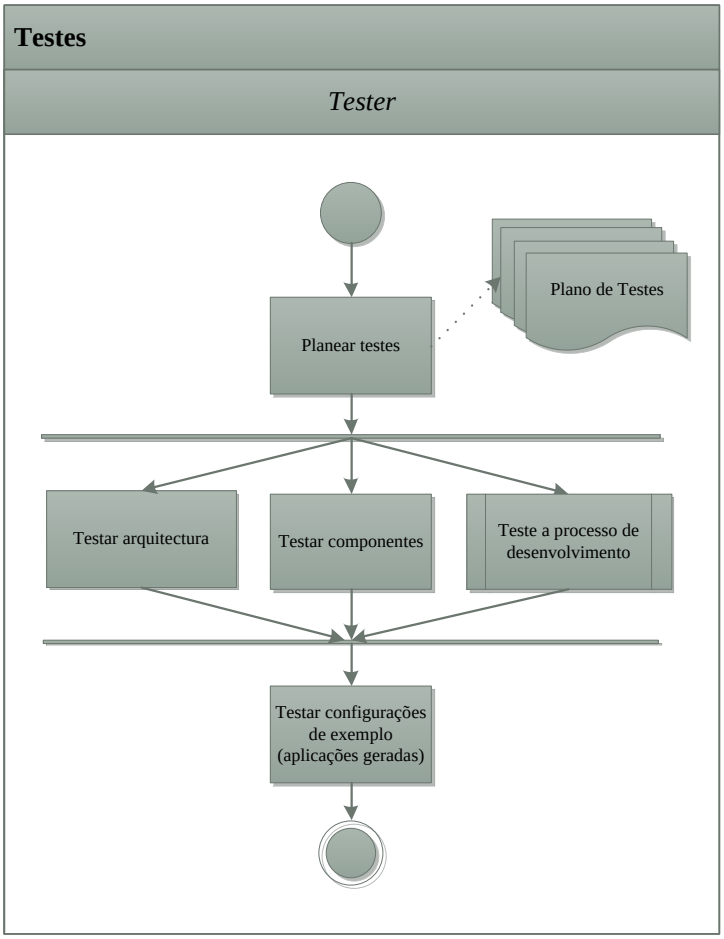

Figura 10-x - Testes

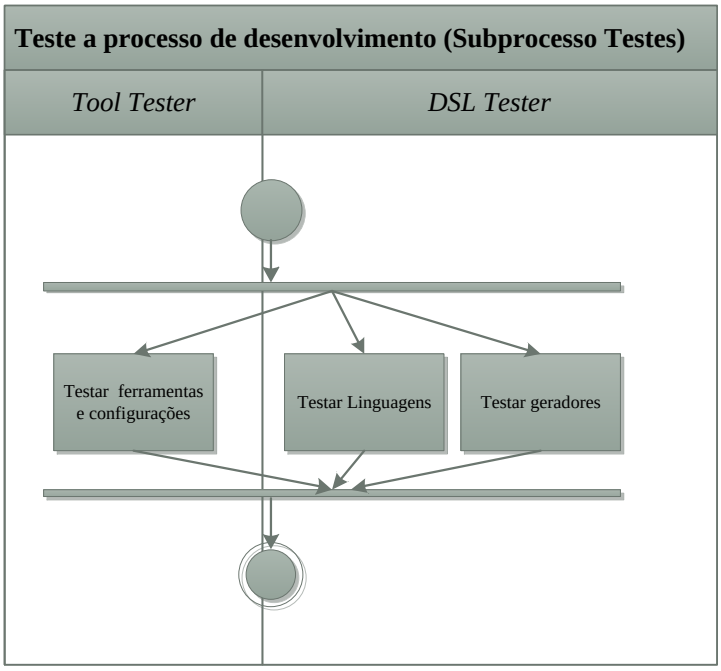

Figura 10-xi - Teste a processo de desenvolvimento (Subprocesso Testes)

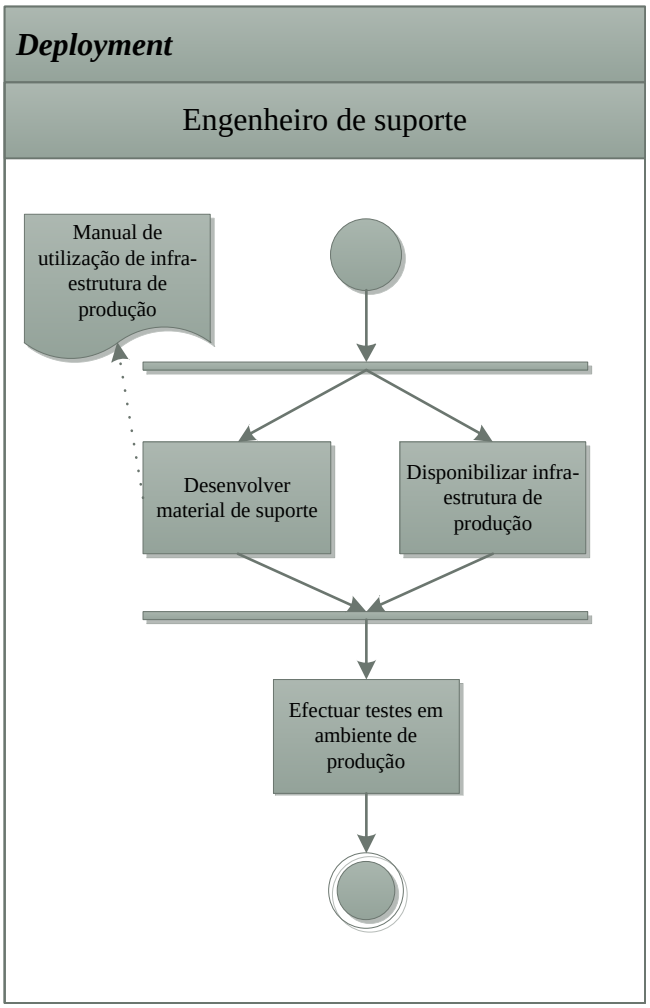

Figura 10-xii - Deployment

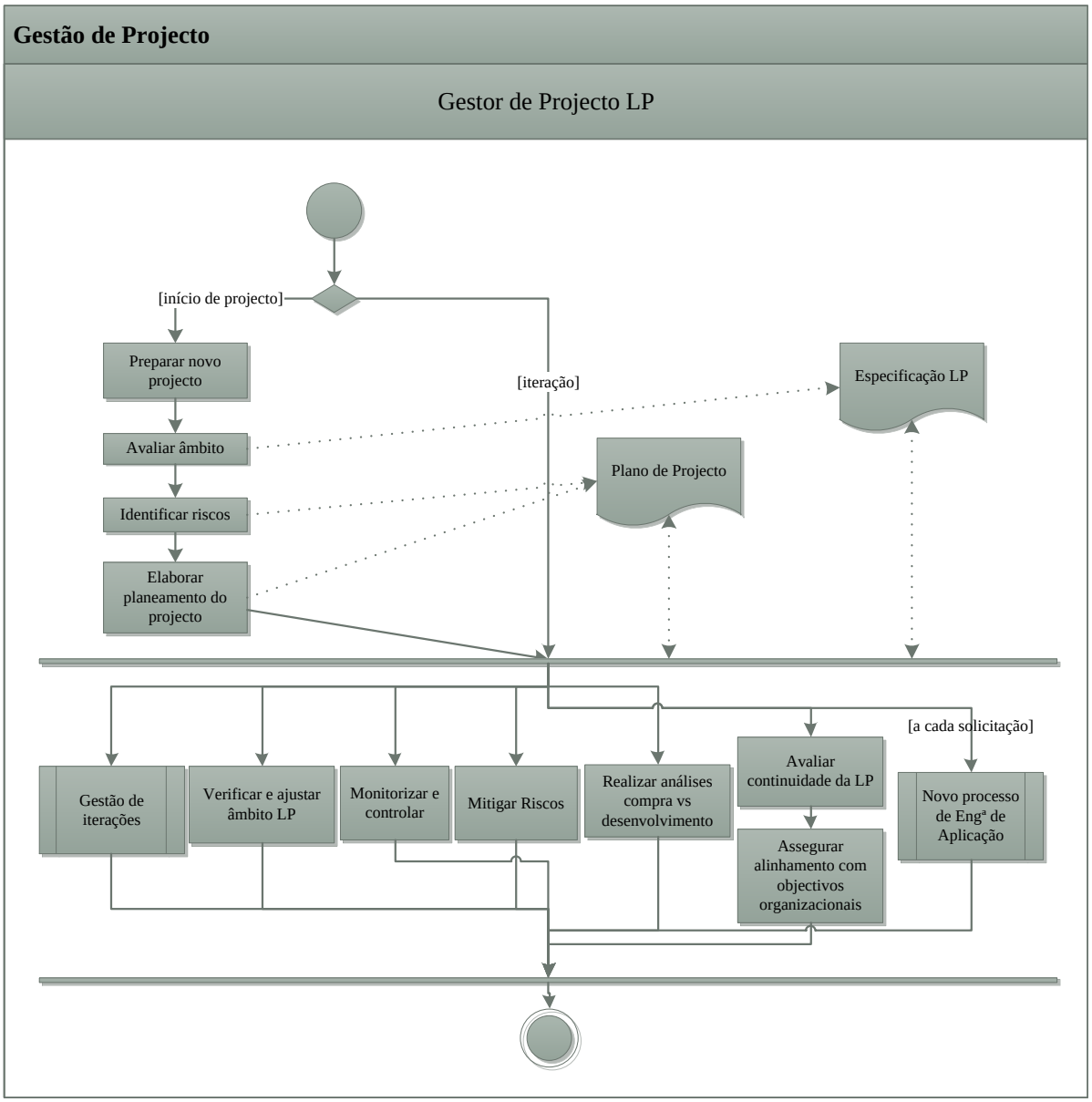

Figura 10-xiii - Gestão de Projecto

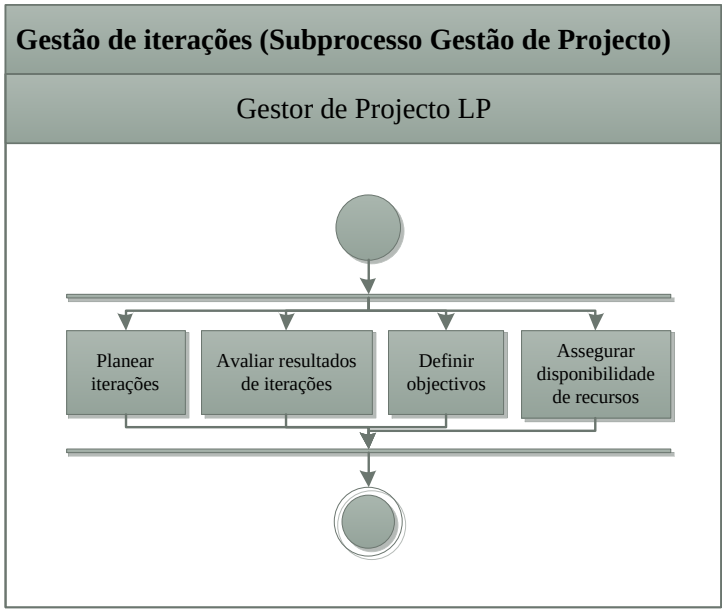

Figura 10-xiv - Gestão de iterações (Subprocesso Gestão de Projecto)

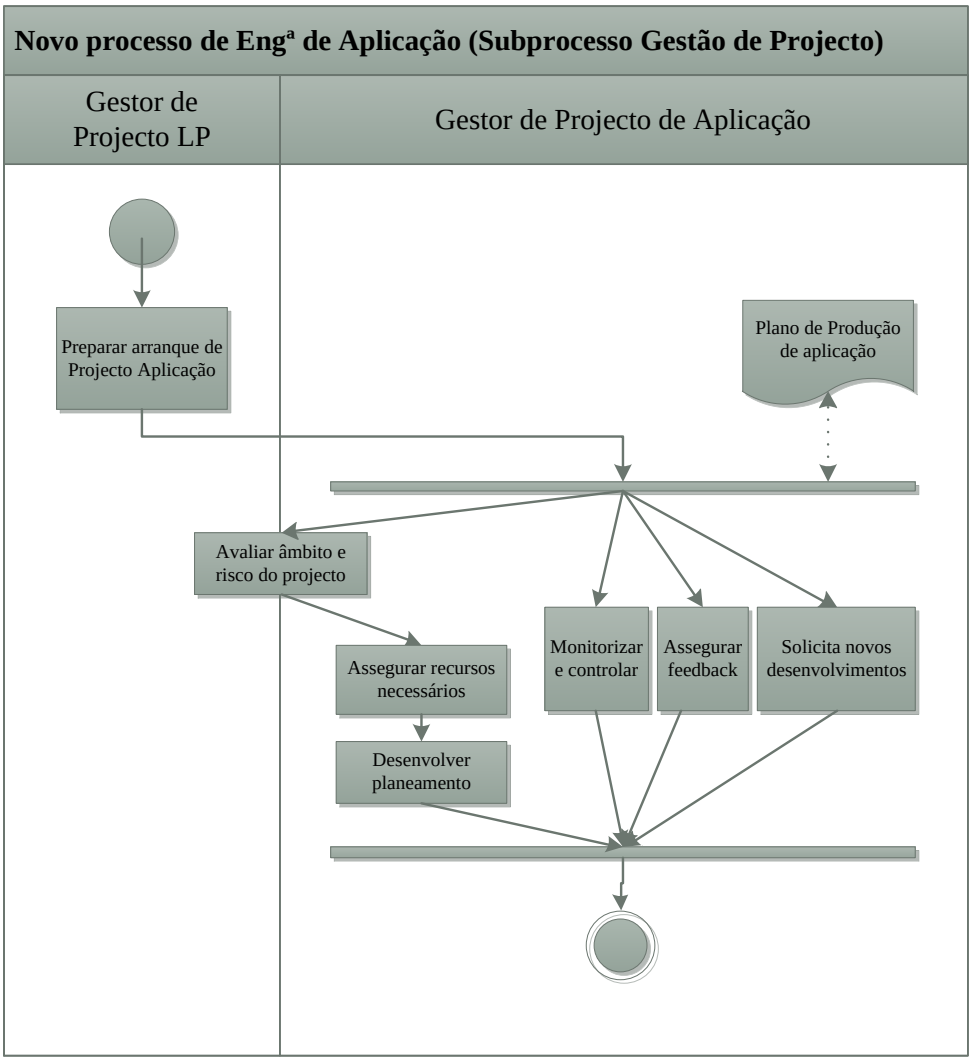

Figura 10-xv - Novo processo de Engª de Aplicação (Subprocesso Gestão de Projecto)

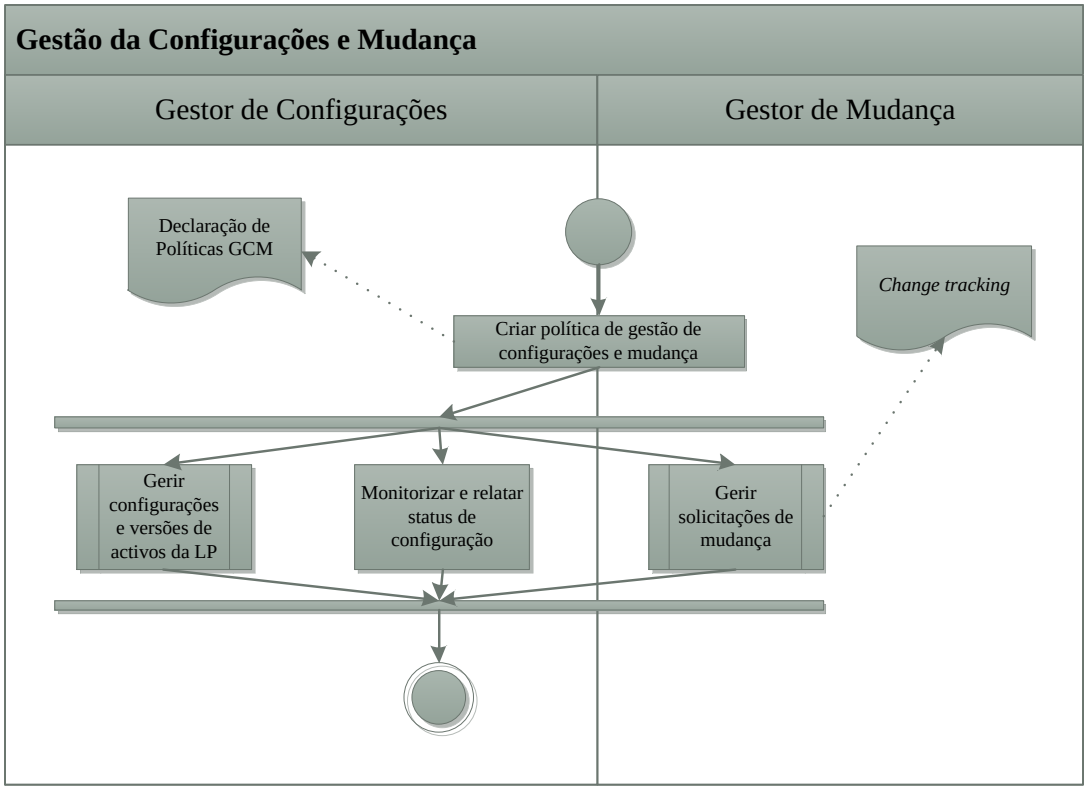

Figura 10-xvi - Gestão da Configurações e Mudança

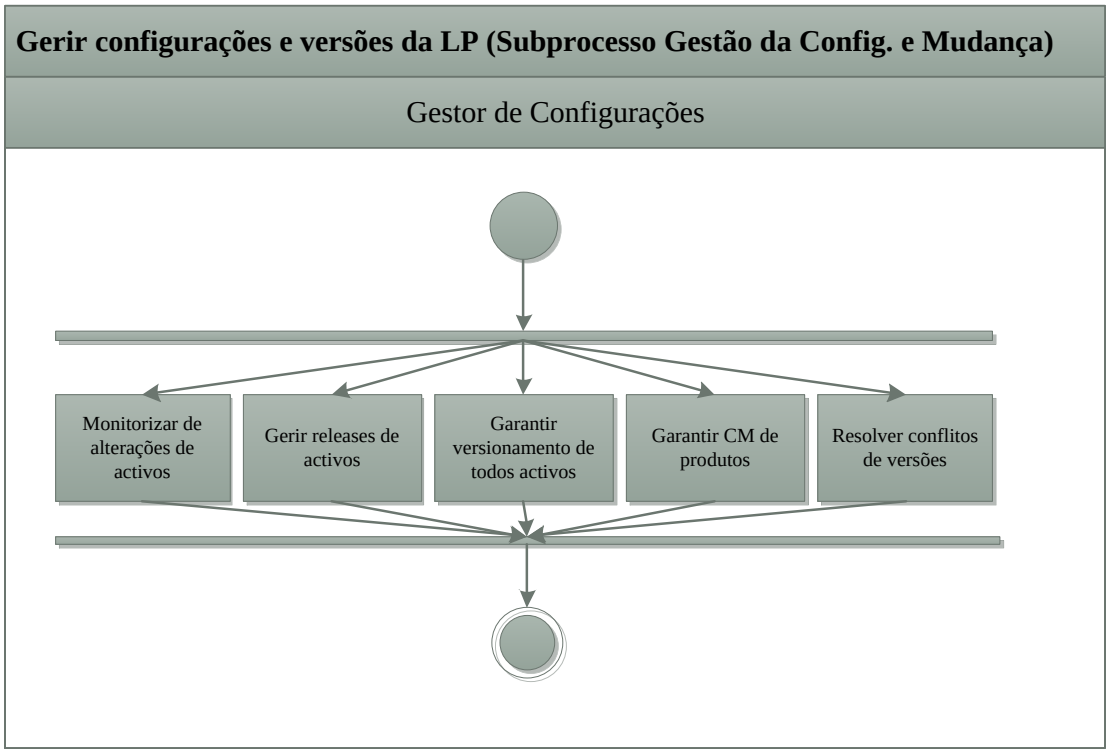

Figura 10-xvii - Gerir configurações e versões da LP (Subprocesso Gestão da Config. e Mudança)

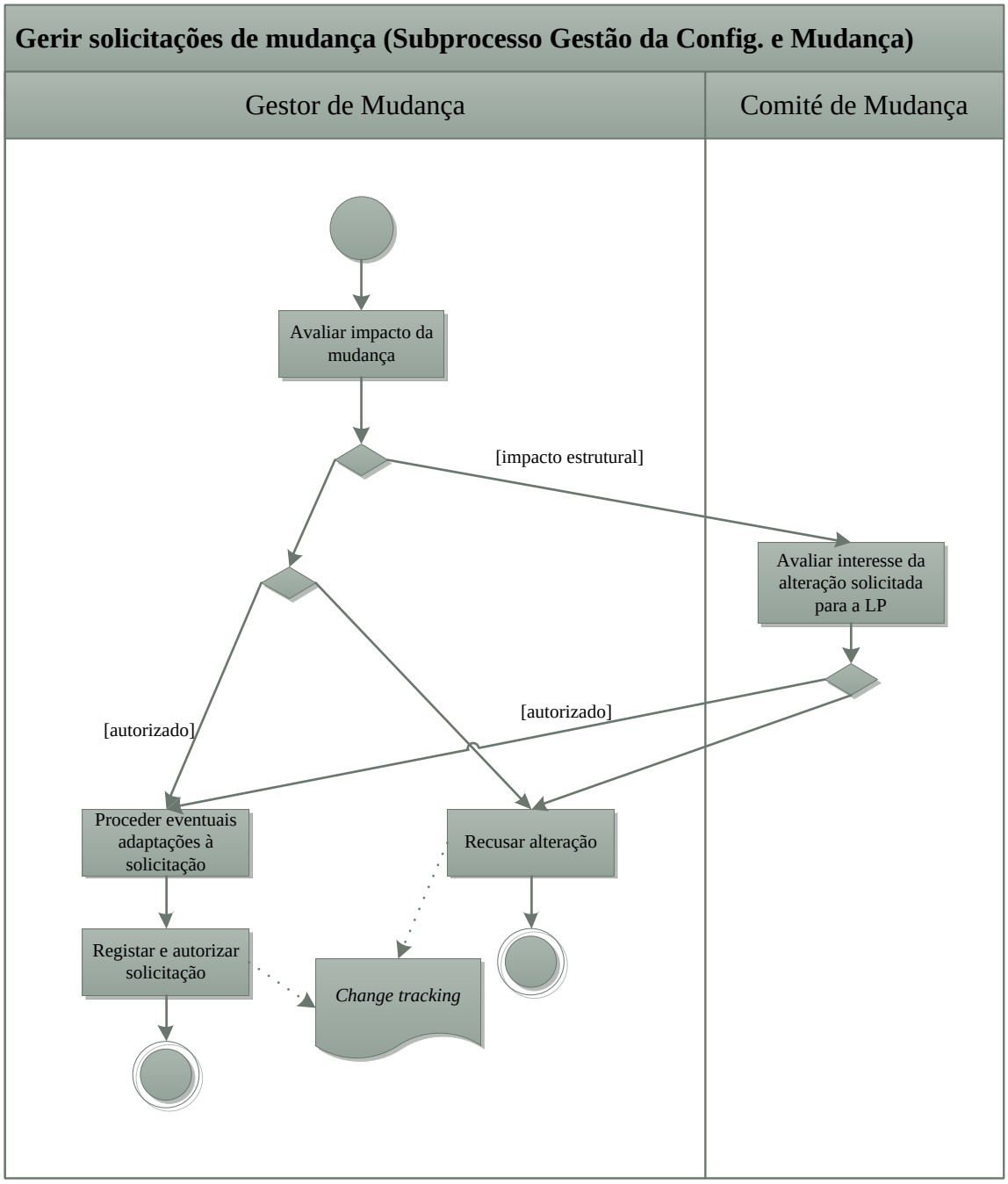

Figura 10-xviii - Gerir solicitações de mudança (Subprocesso Gestão da Config. e Mudança)

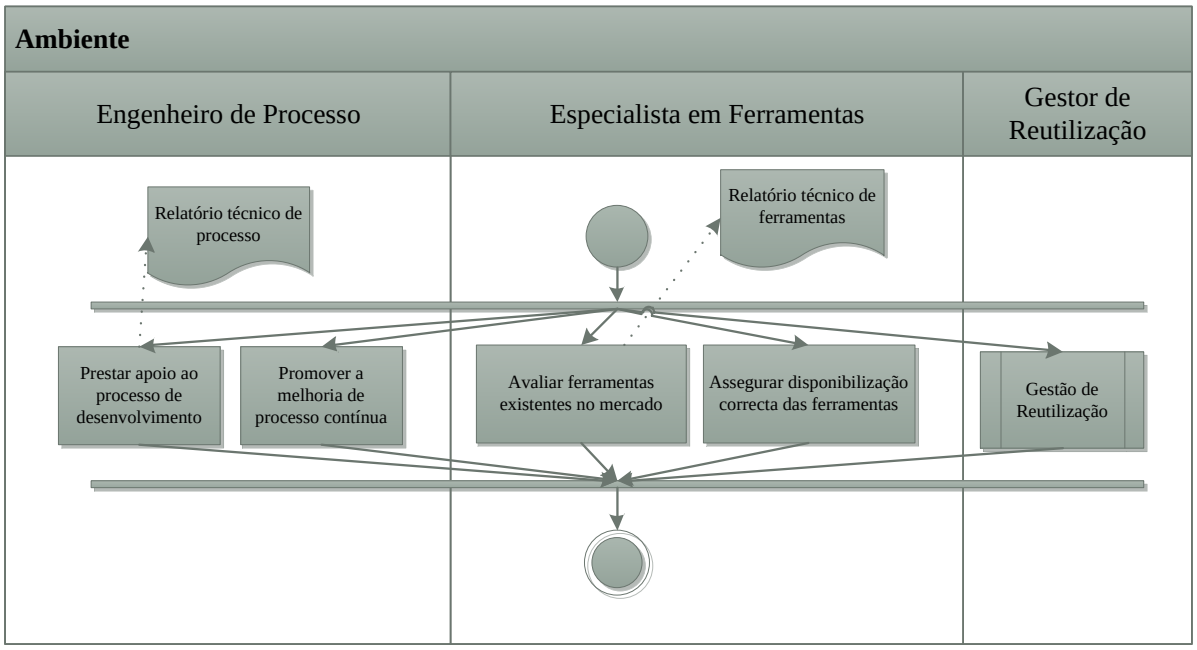

Figura 10-xix - Ambiente

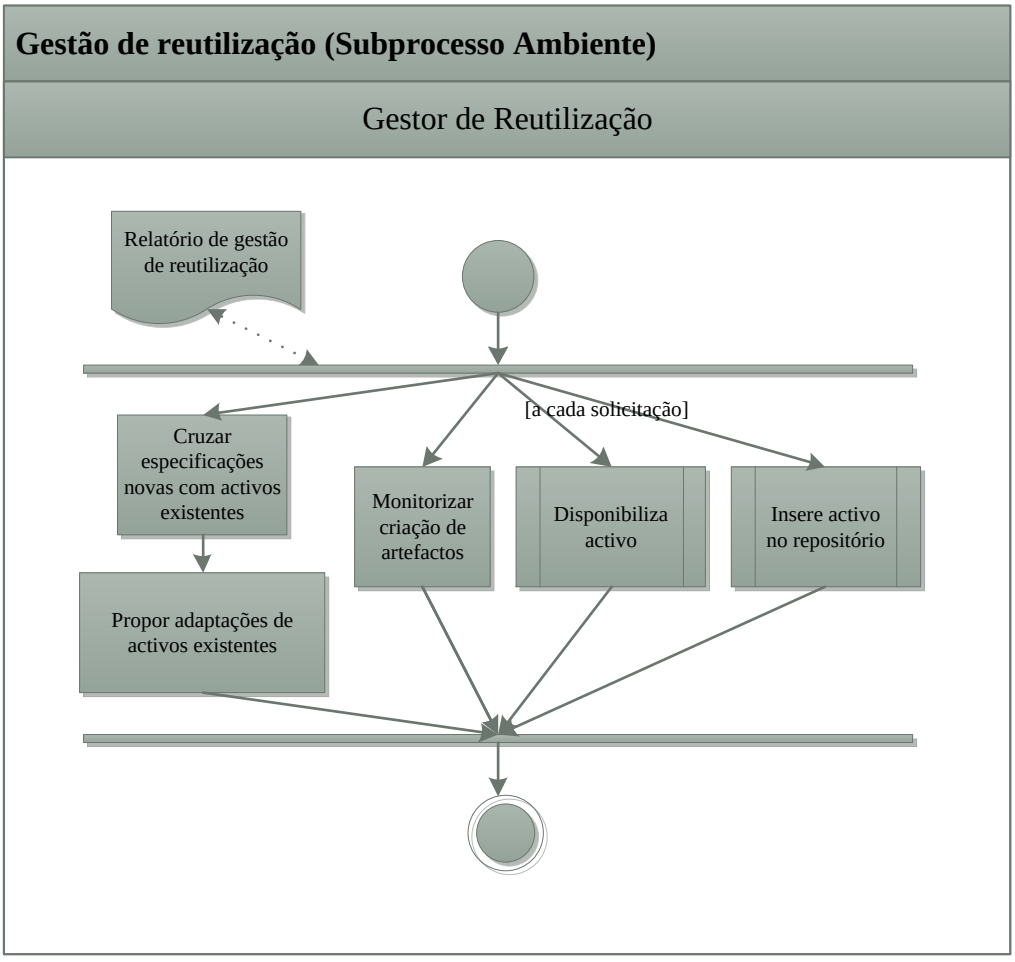

Figura 10-xx - Gestão de reutilização (Subprocesso Ambiente)

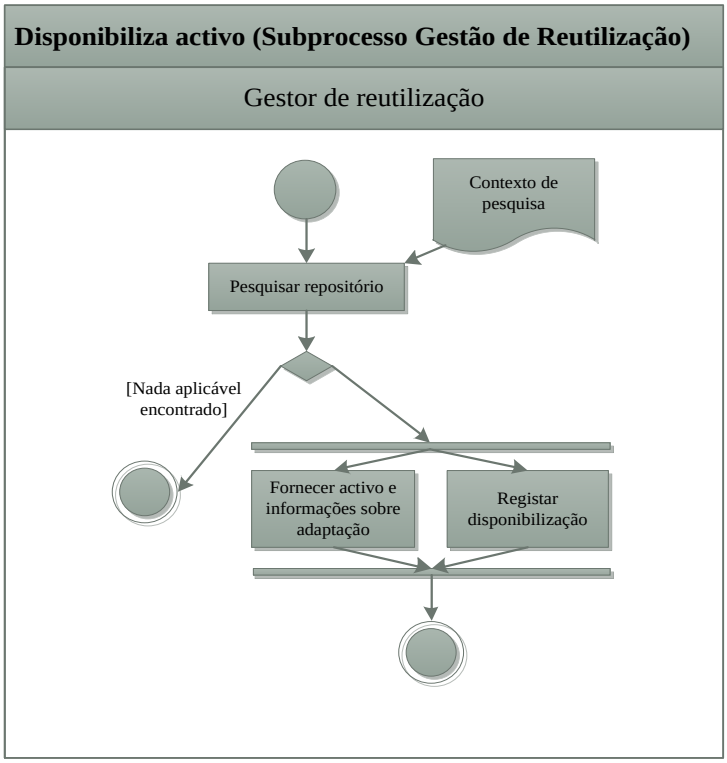

Figura 10-xxi - Disponibiliza activo (Subprocesso Gestão de Reutilização)

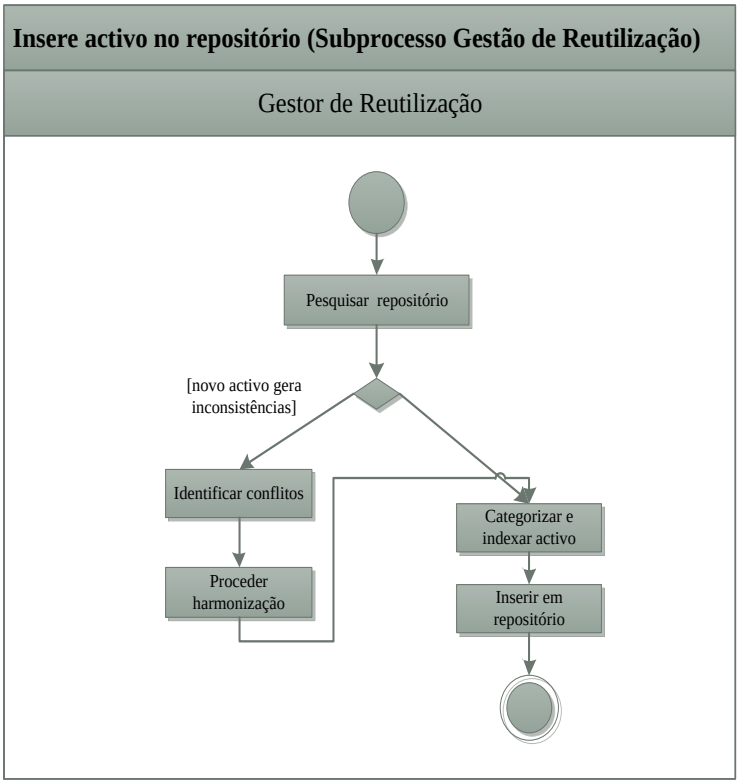

Figura 10-xxii - Insere activo no repositório (Subprocesso Gestão de Reutilização)

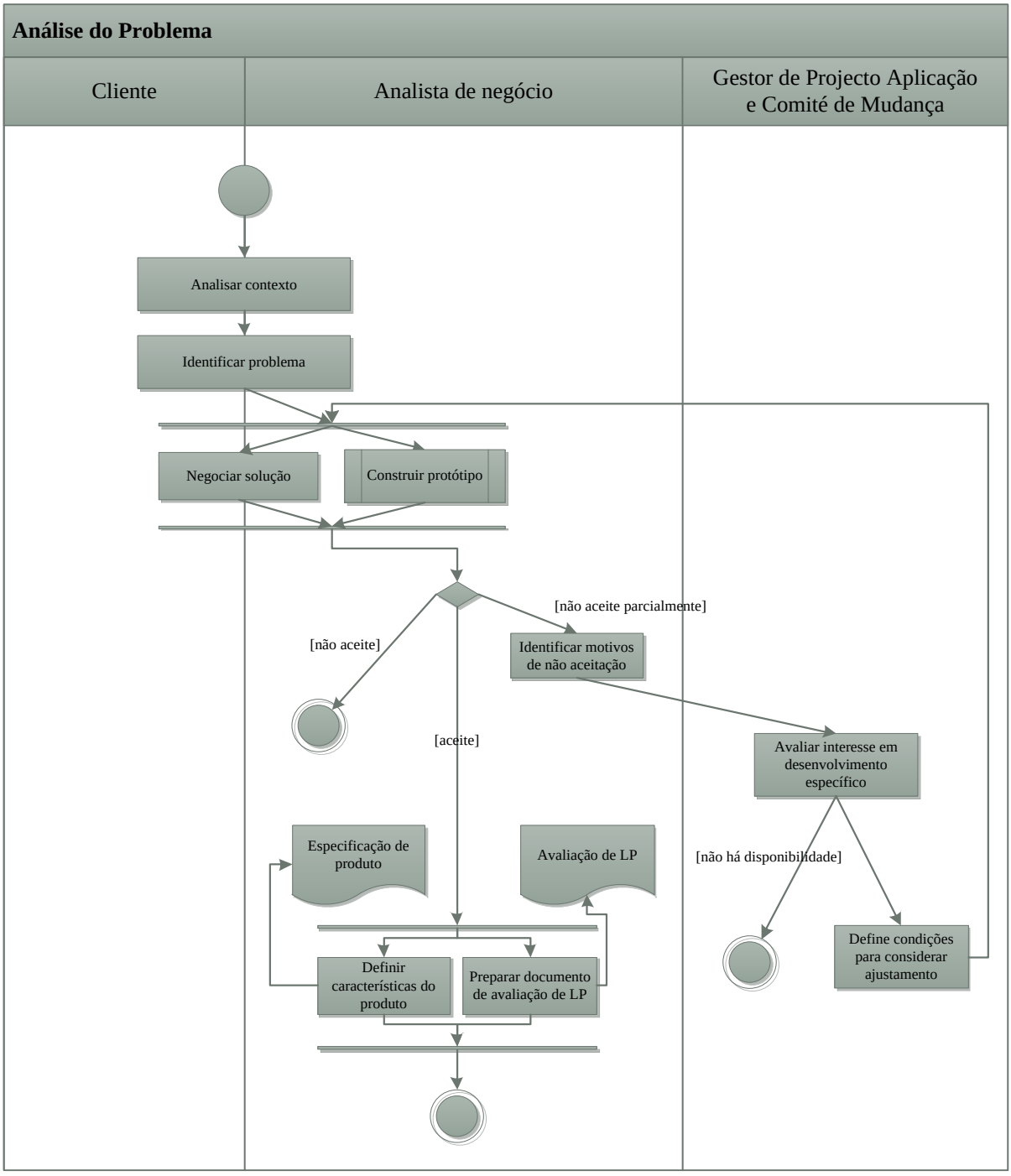

Figura 10-xxiii – Análise do Problema

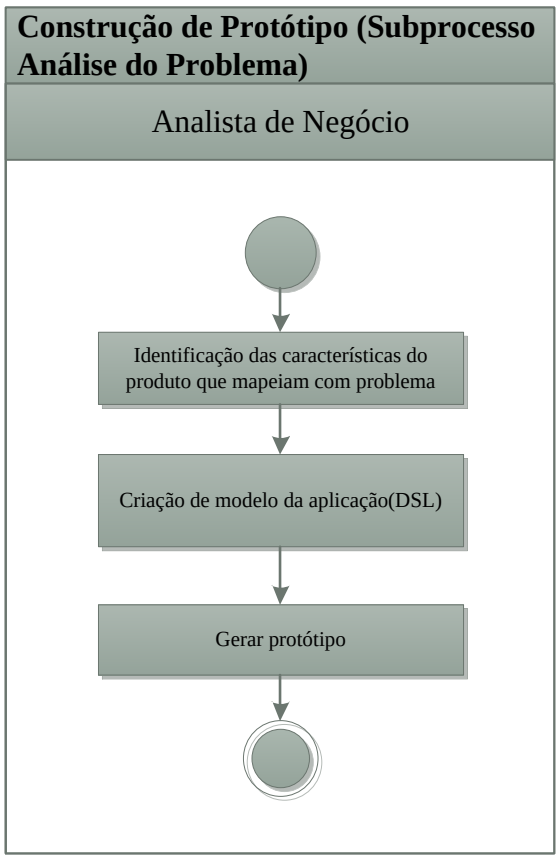

Figura 10-xxiv - Construção de Protótipo (Subprocesso Análise do Problema)

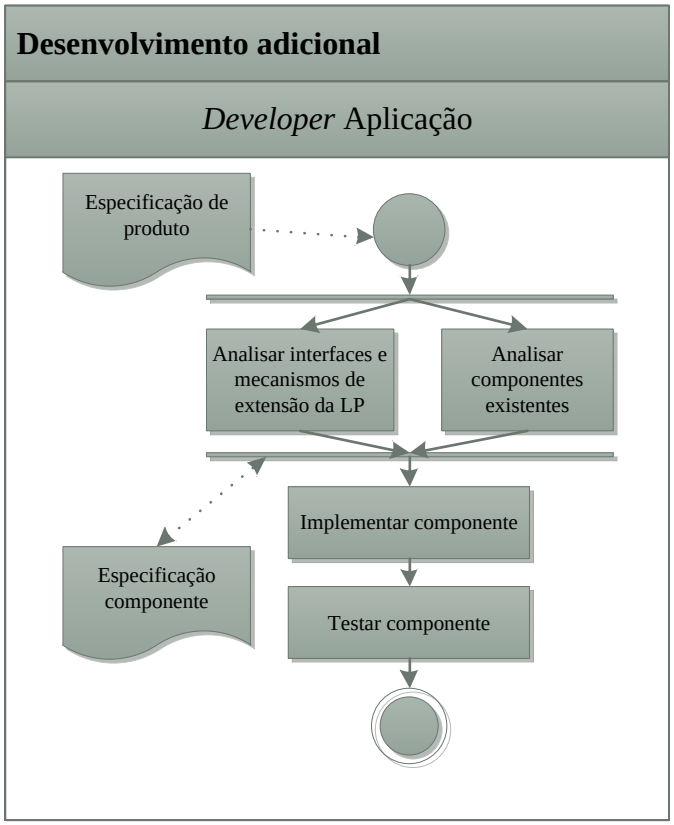

Figura 10-xxv – Desenvolvimento adicional

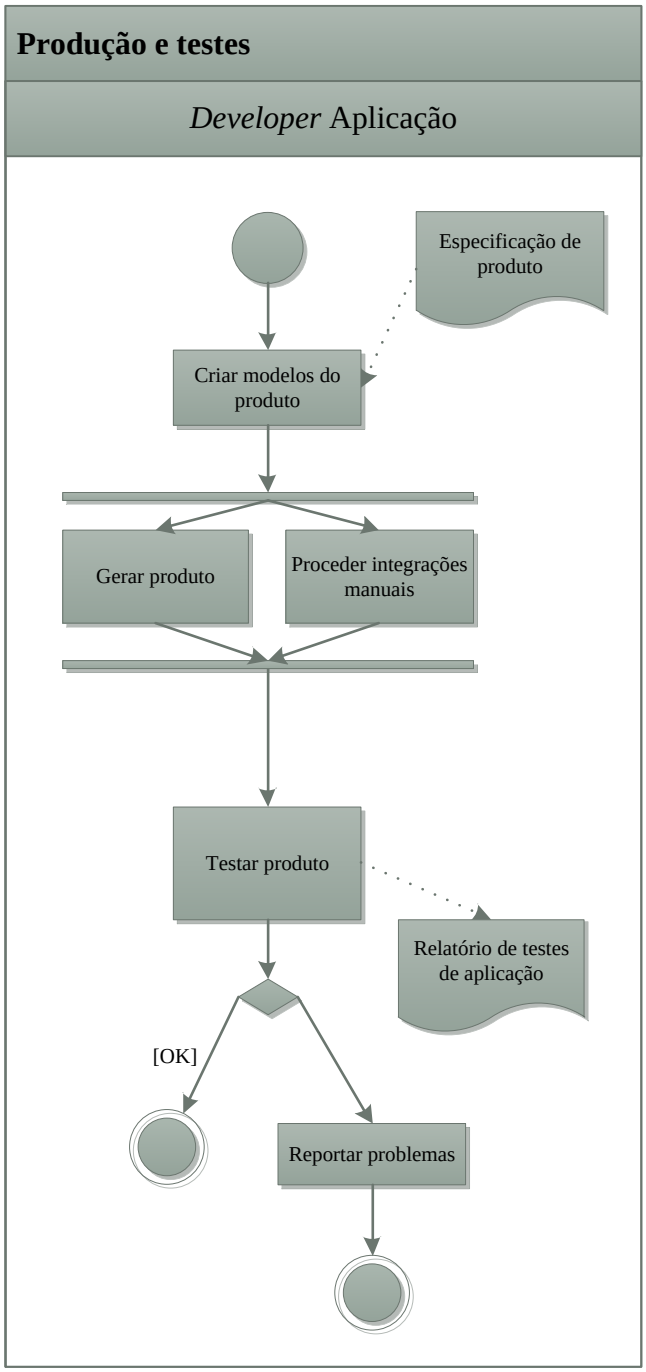

Figura 10-xxvi – Produção e Testes

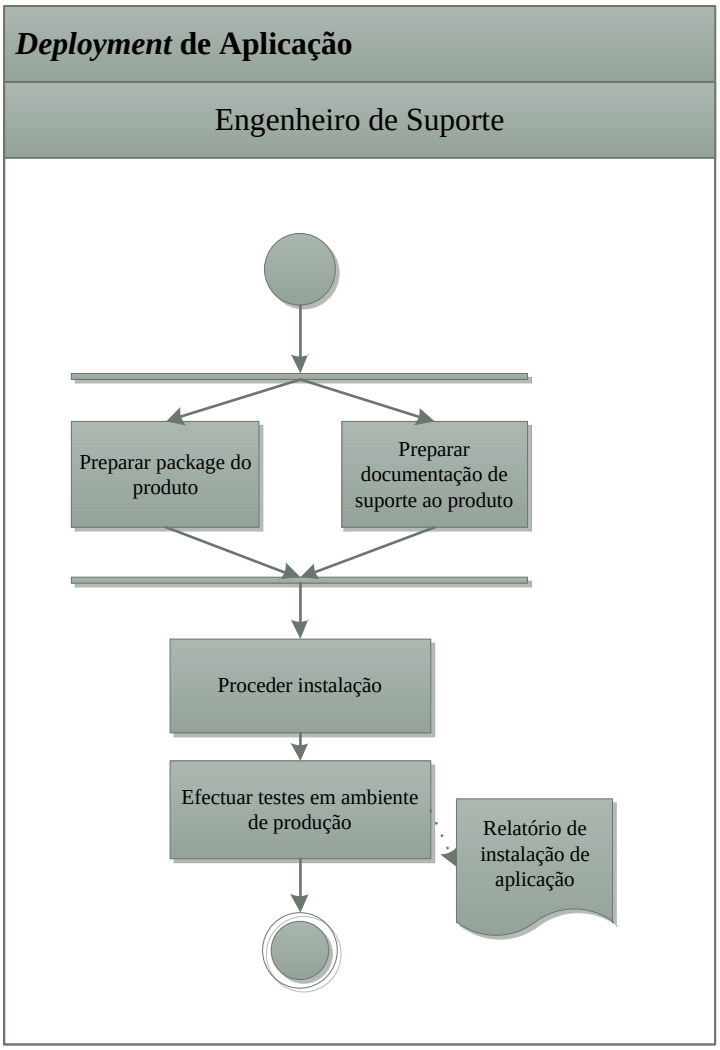

Figura 10-xxvii – Deployment de Aplicação

11 ANEXO – META-MODELO DSL (ITCORP)

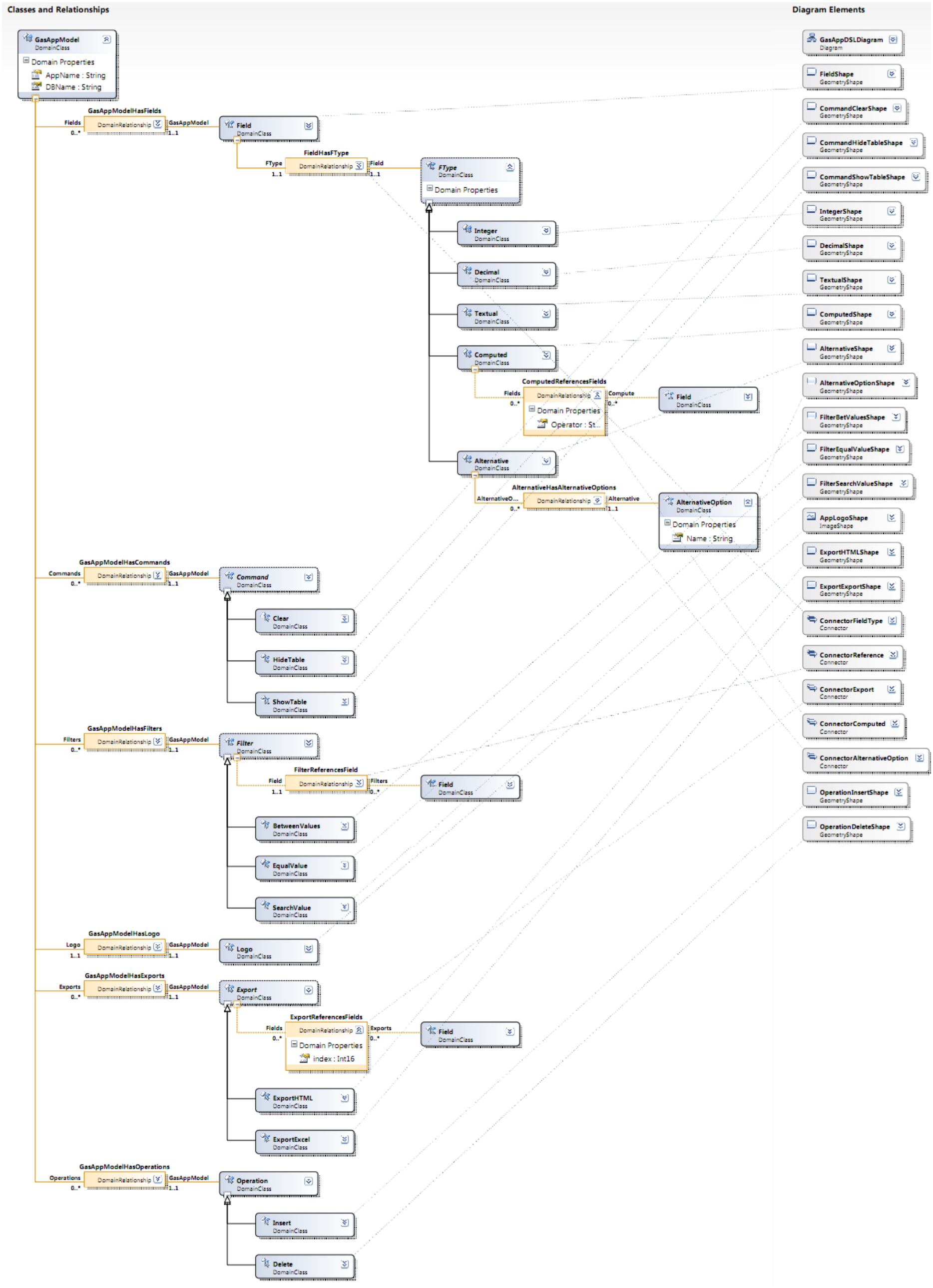

Figura 11-i – Meta-modelo DSL (ITCorp)