UNIVERSIDADE FEDERAL DO RIO DE JANEIRO
INSTITUTO DE MATEMÁTICA
INSTITUTO TÉRCIO PACITTI DE APLICAÇÕES E PESQUISAS COMPUTACIONAIS

RAFAEL ALCEMAR DO NASCIMENTO

O IMPACTO DE UNIDADES DE SOFTWARE NÃO ESSENCIAIS E RELAÇÕES DE PRECEDÊNCIA FLEXÍVEIS SOBRE O VALOR DE PROJETOS DE SOFTWARE

RIO DE JANEIRO
2011

Rafael Alcemar do Nascimento

O IMPACTO DE UNIDADES DE SOFTWARE NÃO ESSENCIAIS E RELAÇÕES DE PRECEDÊNCIA FLEXÍVEIS SOBRE O VALOR DE PROJETOS DE SOFTWARE

Dissertação de Mestrado, apresentada ao Programa de Pós-Graduação em Informática, Instituto de Matemática, Instituto Tércio Pacitti de Aplicações e Pesquisas Computacionais, Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do titulo de Mestre em Informática.

Orientador: Prof. Eber Assis Schmitz, Ph.D.

Co-orientador: Prof. Antonio Juarez Sylvio Menezes de Alencar, D.Phil.

RIO DE JANEIRO
2011

N244 Nascimento, Rafael Alcemar do.
O Impacto de unidades de software não essenciais e relações de precedência flexíveis sobre o valor de projetos de software / Rafael Alcemar do Nascimento. -- 2011.
73 f.: il.

Dissertação (Mestrado em Informática) – Universidade Federal do Rio de Janeiro, Instituto Tércio Pacitti de Aplicações e Pesquisas Computacionais, Instituto de Matemática, 2011.

Orientador: Eber Assis Schmitz
Co-orientador: Antonio Juarez Sylvio Menezes de Alencar

1. Planejamento de Projetos - Teses. 2. Unidades de Software Não Essenciais - Teses. 3. Relações de Precedência Flexíveis - Teses. I. Eber Assis Schmitz (Orient.). II. Antonio Juarez Sylvio Menezes de Alencar (Co-orient.). III. Universidade Federal do Rio de Janeiro. Instituto de Matemática. Instituto Tércio Pacitti de Aplicações e Pesquisas Computacionais. IV. Título

CDD

Rafael Alcemar do Nascimento

O IMPACTO DE UNIDADES DE SOFTWARE NÃO ESSENCIAIS E RELAÇÕES DE PRECEDÊNCIA FLEXÍVEIS SOBRE O VALOR DE PROJETOS DE SOFTWARE

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Informática, Instituto Tércio Pacitti de Aplicações e Pesquisas Computacionais, Instituto de Matemática, Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do titulo de Mestre em Informática.

Aprovado em: Rio de Janeiro, 12 de Fevereiro de 2011.

Prof. Eber Assis Schmitz, Ph.D.
PPGI-NCE/IM - DCC - UFRJ (Orientador)

Prof. Antonio Juarez Sylvio Menezes de Alencar, D. Phil.
NCE - UFRJ (Co-orientador)

Prof. José Orlando Gomes, D.Sc.
PPGI-NCE/IM - DEI/POLI - UFRJ

Prof. Alexandre Luis Correa, D.Sc.
UNIRIO

Profa. Renata Mendes de Araujo, D.Sc.
UNIRIO

Dedicatória

À minha linda esposa, que soube com tanta paciência e amor me apoiar durante todo esse tempo ...

Agradecimentos

Primeiramente a Deus, que com seu grande amor me sustentou até aqui me enchendo de saúde e paz, para que este dia tão esperado pudesse chegar.

À minha esposa Verônica, por todo seu empenho e apoio durante todos os momentos que estamos juntos.

Aos meus pais que sempre com seu amor e conselhos me educaram e me ensinaram a não desanimar diante de quaisquer dificuldades.

Aos meus orientadores, Eber e Juarez, que não desanimaram em momento algum durante a minha jornada no mestrado. Que souberam me apoiar mesmo diante de várias limitações que eu tinha e que me ensinaram da melhor forma superar várias delas.

Aos professores Alexandre Correa e Fernando Manso que sempre me incentivaram e me ajudaram durante o tempo de mestrado.

Aos meus amigos do mestrado e em especial a Felipe Dias, Enio, Guga, Leandro, Bruno e tantos outros, que com suas contribuições me ajudaram e me ensinaram quando o meu conhecimento não era suficiente.

Resumo

NASCIMENTO, Rafael Alcemar do. **O impacto das unidades de software não essenciais e relações de precedência flexíveis sobre o valor de projetos de software**. Rio de Janeiro, 2011. Dissertação (Mestrado em Informática) – Instituto Tércio Pacitti de Aplicações e Pesquisas Computacionais, Instituto de Matemática, Universidade Federal do Rio de Janeiro, Rio de Janeiro, 2011.

Atualmente, empresas e investidores querem obter o máximo valor possível de um projeto de software. Assim, a priorização e modularização de software deve ser feita de maneira ampla, a fim de obter o máximo de valor de um projeto de software. Esta dissertação descobre o valor de Minimum Marketable Features Não essenciais (NMMFs) e de Elementos Arquiteturais Não essenciais (NAEs) para projetos de software, mostra ainda que o valor dessas unidades de software podem variar significativamente das unidades de software essenciais e discute o impacto da identificação prévia de NMMFs e NAEs no valor do software para o negócio e para a implantação das estratégias do negócio. Além disso, esta dissertação demonstra que a existência de relações de precedência flexíveis entre MMF e EAs pode ser explorada para aumentar ainda mais o valor do software.

Palavras-chave: Planejamento de Projetos, Unidades de Software Não Essenciais, Relações de Precedência Flexíveis, Maximização do VPL Financeiro, Gerência de Projetos de Software.

Abstract

NASCIMENTO, Rafael Alcemar do. **O impacto das unidades de software não essenciais e relações de precedência flexíveis sobre o valor de projetos de software**. Rio de Janeiro, 2011. Dissertação (Mestrado em Informática) – Instituto Tércio Pacitti de Aplicações e Pesquisas Computacionais, Instituto de Matemática, Universidade Federal do Rio de Janeiro, Rio de Janeiro, 2011.

Nowadays, companies and sponsors want to obtain the maximum possible value of a software project. Thus, the analysis of the modules to be built must be done of a large view to obtain the maximum value of the project. This dissertation uncovers the value of nonessential minimum marketable features modules (NMMF) and nonessential architectural elements (NAEs) to software projects, shows that the value-creation path of such self-contained software units may be quite different from that of essential software units, and discusses the impact of early NMMFs and NAEs identification on the value of software to business and the deployment of business strategies. Moreover, the paper demonstrates that the existence of flexible precedence relation among MMFs and AEs may also be exploited to further increase the value of software.

Keywords: Project Planning, Nonessential Software Unities, Flexible Precedence Relations, NPV maximization, Software Project Management.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

EA	Elemento Arquitetural
FC	Fluxo de Caixa
IFM	Incremental Funding Method
MMF	Minimal Marketable Feature
TI	Tecnologia da Informação
VPL	Valor Presente Líquido

SUMÁRIO

1 INTRODUÇÃO

1.1 Contextualização do problema

Apesar do papel de destaque que a tecnologia da informação(TI) desempenha no cenário corporativo, o financiamento de projetos de desenvolvimento de *software* no ambiente altamente competitivo no qual as empresas fazem negócios se tornou uma questão central tanto para gerentes quanto para profissionais de tecnologia (WU; SHI; GURBAXANI, 2007).

Em diversos mercados, os investidores têm buscado não somente melhores retornos financeiros como também períodos menores de investimento, tempo mais rápido de colocação dos produtos no mercado, menos riscos e maior capacidade de adaptação a novas condições mercadológicas, com rapidez e eficiência (PRIYA KURIEN; PURUSHOTTAM, 2004; CUSUMANO, 2004).

Como consequência, se torna cada vez mais difícil conseguir financiamento para projetos de desenvolvimento de *software* que não propiciem um valor claramente definido para o negócio (DENNE; CLELAND-HUANG, 2005).

Em sintonia com essas idéias, muitas propostas têm sido apresentadas para trazer disciplina financeira para o desenvolvimento de *software* (GREMBERGEN, 2001). Enquanto algumas propostas são amplamente baseadas em métricas de avaliação financeira de projetos, tais como o valor presente líquido, retorno do investimento, taxa interna de retorno e, mais recentemente, teoria de opções reais (BENAROCH; SHAH; JEFFERY, 2006; DEKLEVA, 2005), outras têm uma visão mais holística do desenvolvimento de *software* e defendem o uso de métricas baseadas em análise multivariada tais como *Strategy-to-Bottom-Line Value Chain*, *Multi-layer evaluation process* e *Information Economics* (BENSON; BUGNITZ; WALTON, 2004; MILIS; MERCKEN, 2004; ROSACKER; OLSON, 2008).

Contudo, todas essas tentativas falham em reconhecer que a priorização e a modularizacão de requisitos têm um papel fundamental na construção do valor do *software*. Enquanto os requisitos satisfeitos por uma unidade de *software* são cruciais para a determinação do seu valor, a ordem na qual essas unidades são implementadas indicam quão cedo esse valor pode ser apropriado.

Uma exceção notável é apresentada por Dennne e Cleland-Huang (DENNE; CLELAND-HUANG, 2005), que sugerem o uso de uma análise financeira abrangente com a finalidade de maximizar o valor de projetos de *software* compostos por *minimum marketablede features* (MMFs), ou seja, unidades de *software* contendo pequenos conjuntos de funcionalidades que têm valor para o negócio.

Em seu trabalho, Dennne e Cleland-Huang mostram que a ordem de implementação dos MMFs pode alterar substancialmente o valor destes projetos. Posteriormente, as idéias de Dennne e Cleland-Huang foram estendidas por Alencar *et al.* (ALENCAR; SCHMITZ; ABREU, 2008), que usam a técnica *branch & bound* para superar algumas limitações impostas pelo método proposto em (DENNE; CLELAND-HUANG, 2005).

Contudo, tanto Denne e Cleland-Huang (*op. cit.*), quanto Alencar *et al.* (*op. cit.*) falharam em não reconhecer que:

(a) nem sempre todas as unidades de *software* são essenciais ao desenvolvimento de um projeto de *software*;

(b) se uma unidade de *software* não for essencial a um projeto de *software*, o seu desenvolvimento pode ou não ser realizado pelo gerente de projetos durante o ciclo de vida do *software*;

(c) o valor de unidades de *software* não-essenciais pode variar de acordo com o conjunto de unidades que precederam o seu desenvolvimento e

(d) quando implementados, ao invés de criar o seu próprio fluxo de caixa, as unidades de *software* não-essenciais podem contribuir para o valor de um *software* influenciando de forma positiva o valor de unidades de *software* essenciais.

1.2 Objetivo do Trabalho

Esta dissertação é um passo à frente no preenchimento desta lacuna(isto é dos problemas citados no parágrafo anterior), revelando o valor de unidades de *software* não essenciais (que chamamos de NMMFs e NAEs, por razões que ficarão claras no Capítulo 2), cujo processo de criação de valor pode ser diferente das unidades de *software* tradicionais.

Além disso, esta dissertação mostra como a identificação prematura de NMMFs e NAEs durante o ciclo de vida do projeto pode afetar o valor final de projetos de *software*, beneficiando o desenvolvimento do *software* como um todo e ajudando a formatar as estratégias de negócio, já que a identificação prematura dessas unidades

aumenta o retorno sobre os investimentos realizados por uma organização.

Por fim, esta dissertação mostra como a combinação de relações de precedência flexíveis, NAEs e NMMFs pode ser usada para aumentar ainda mais o valor do *software*. Para isso, criamos uma variante do algoritmo *branch and bound*, capaz de indicar a melhor ordem de implementação de uma seqüência de unidades de *software* que contenham relações de precedência flexíveis e módulos não-essenciais.

1.3 Contribuições

Os resultados parciais desta dissertação foram aceitos para publicação no:

- VII Simpósio Brasileiro de Sistemas de Informação (SBSI 2011), realizado na cidade de Salvador - Bahia, entre 23 a 25 de maio de 2011 sob o título *O impacto de unidades de software não essenciais e relações de precedência flexíveis sobre o valor de projetos de software*;

- 23^{rd} International Conference on Software Engineering and Knowledge Engineering (SEKE 2011), realizado na cidade de Miami Beach - USA, entre 7 e 9 de julho de 2011 sob o título *On the Impact of Nonessential Self-contained Software Units and Flexible Precedence Relation upon the Value of Software Projects* e

- Journal of Software (ISSN 1796-217X) sob o título *Unleashing the Potential Impact of Nonessential Self-contained Software Units and Flexible Precedence Relations upon the Value of Software.*

1.4 Organização da Dissertação

Esta dissertação está organizada da seguinte forma:

- Capítulo 2 apresenta uma revisão dos principais conceitos e métodos utilizados nesta dissertação.
- Capítulo 3 descreve o método utilizado nesta dissertação.
- Capítulo 4 introduz um exemplo inspirado no mundo real e que ajuda a entender o papel exercidos por NMMFs, NAEs e relações de precedência flexíveis no desenvolvimento de *software*.
- Capítulo 5 apresenta as conclusões desta dissertação.
- Apendice apresenta a ferramenta desenvolvida para demonstrar o método proposto.

2 ARCABOUÇO CONCEITUAL

Nesta capítulo apresentamos os principais conceitos utilizados no desenvolvimento desta dissertação, tais como: casos de uso, método de financiamento incremental (IFM), *Minimum Marketable Features Modules, Architectural Elements*, etc.

2.1 Caso de Uso

Caso de uso é uma linguagem de especificação gráfica e textual criada no final dos anos 80 por Ivar Jacobson, enquanto trabalhava para a Telefonaktiebolaget LM Ericsson na Suécia, que descreve como um conjunto de atores (conjunto de papéis que os usuários de casos de uso podem assumir quando interagem com o sistema) usa um sistema para alcançar um objetivo que tem valor para o negócio, e como o sistema ajuda esses atores a alcançar os seus objetivos (BITTNER; SPENCE, 2002). A Figura 2.1 apresenta um diagrama de caso de uso de um sistema de controle de empréstimos.

Na Figura 2.1 "Cliente" é um ator e as elipses são casos de uso, cujos significados são descritos na Tabela 2.1. A palavra "*include*" que aparece no diagrama do sistema

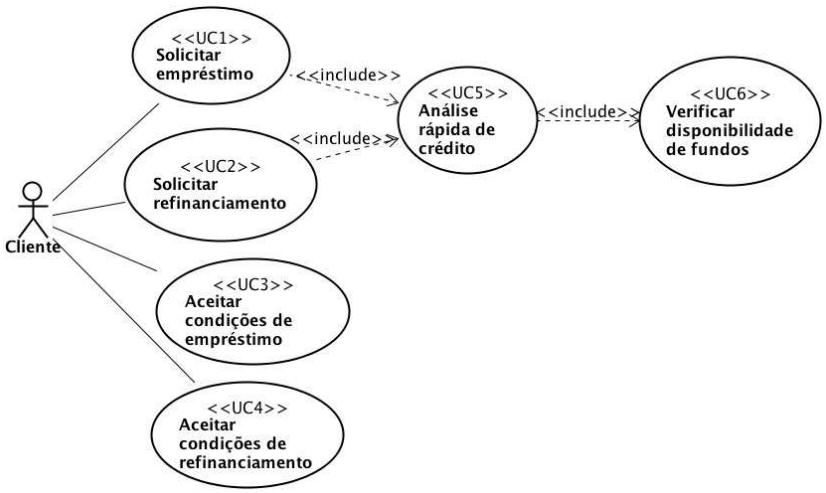

Figura 2.1: Um modelo de caso de uso de um sistema de controle de empréstimos.

de controle de empréstimos é usada para facilitar a modularização, frequentemente conectando partes comuns de dois ou mais casos de uso.

Tabela 2.1: Descrição de caso de uso

Caso de Uso		
Id	**Nome**	**Descrição**
UC_1	Solicitar empréstimo	Coleta os dados necessários para conceder um empréstimo a um cliente
UC_2	Solicitar refinanciamento	Coleta os dados necessários para conceder o refinanciamento de um empréstimo existente
UC_3	Aceitar condições de empréstimo	Permite que o cliente aceite ou recuse as condições do empréstimo propostas pelo cedente
UC_4	Aceitar condições de refinanciamento	Permite que o cliente aceite ou recuse as condições de refinanciamento propostas pelo cedente
UC_5	Análise rápida de crédito	Verifica a probabilidade de um cliente pagar um empréstimo de acordo com os valores e as datas das parcelas
UC_6	Verificar disponibilidade de fundos	Verifica se o cedente possui os fundos necessários para conceder um empréstimo a um determinado cliente

Os casos de uso são independentes de tecnologia. Assim, casos de uso não incluem detalhes sobre *interfaces*, uso de bancos de dados e design de telas. A palavra "*extends*" é eventualmente usada em diagramas de caso de uso para indicar que um comportamento opcional ou excepcional é adicionado em um caso de uso existente. Essa característica é usada em diagramas que aparecem posteriormente nesta dissertação.

Conforme são, os casos de uso podem ser aplicados em várias fases do processo de engenharia de requisitos, inclusive na fase inicial de levantamento de requisitos, modelagem de requisitos e validação de requisitos. Além disso, os casos de uso podem ser descritos em diferentes níveis de detalhe e em estilos diferentes. Todavia, independente de sua forma de expressão, sua característica mais importante permanece invariável - um caso de uso deve apresentar algum valor para o negócio (DENNEY, 2005). Este valor pode ser direto, se o caso de uso propiciar um retorno positivo dos investimentos realizados, ou indireto, se o caso de uso permitir que outras unidades propiciem retorno positivo.

2.2 Incremental Funding Method

De acordo com Denne e Cleland-Huang (DENNE; CLELAND-HUANG, 2004a), o *Incremental Funding Method*, ou IFM, é um método de análise financeira para o desenvolvimento de *software*. O IFM maximiza o retorno de investimento com a entrega de funcionalidades em pequenas unidades de *software* auto-contidas, que tém valor para o negócio e que são sequenciadas cuidadosamente para otimizar o retorno do investimento.

Essas pequenas unidades de *software* são chamadas de *Minimum Marketable Features*, ou simplesmente MMFs. MMFs são módulos que contém pequenos conjuntos

de funcionalidades que podem ser entregues de forma rápida e que criam valor para o negócio em uma ou mais das seguintes áreas:

- *Diferenciação competitiva* – A unidade de *software* habilita a criação de produtos ou serviços que são valorizados pelos clientes e que são diferentes de tudo que é oferecido no mercado;
- *Geração de lucro* – Embora a unidade de *software* não forneça nenhuma inovação que tenha valor para os clientes, ela aumenta o lucro propiciando o oferecimento de produtos de qualidade similar aos do mercado por preços melhores;
- *Redução de custos* – A unidade de *software* permite que a empresa economize dinheiro reduzindo os custos de execução de um ou mais processos;
- *Projeção da marca* – A construção da unidade de *software* permite reforçar na mente dos clientes atributos pelos quais a empresa deseja ser conhecida, tais como: tecnologicamente avançada, socialmente responsável, amiga do meio ambiente, etc.; e
- *Aumento da fidelidade dos clientes* – A unidade de *software* leva o cliente a comprar mais, com maior freqüência ou ambos.

Embora uma MMF seja uma unidade auto-contida, muitas vezes ela só pode ser desenvolvida depois que outras partes do projeto tiverem sido finalizadas. Estas partes do projeto podem ser outras MMFs ou a infra-estrutura arquitetural, ou seja, o conjunto de funcionalidades básicas que não oferecem valor direto aos clientes, mas que são necessárias às MMFs.

A própria infra-estrutura arquitetural pode ser decomposta em elementos autocontidos que podem ser entregues separadamente. Estes elementos, chamados de *ar-*

chitectural elements (AEs) ou elementos arquiteturais, permitem que a arquitetura seja entregue de acordo com a demanda, reduzindo ainda mais o investimento inicial necessário para se executar um projeto. Por exemplo, a biblioteca de interfaces gráficas que permite que os diversos módulos de um mesmo *software* sejam construídos com a mesma identidade visual, o conjunto de funções financeiras que auxiliam o funcionamento de um sistema de empréstimo, os módulos que calculam os impostos devidos na emissão de notas fiscais eletrônicas, etc.. Embora nenhuma dessas unidades de *software* traga diretamente retorno financeiro para o projeto, o bom senso nos indica que determinados módulos não deveriam ser desenvolvidos até que as unidades estejam disponíveis.

2.2.1 Relação Caso de Uso - MMFs e AEs

Observe que um caso de uso descreve conjuntos de funcionalidades que um sistema deve satisfazer. Se esse conjunto de funcionalidades gera valor para o negócio na forma de diferenciação competitiva, geração de lucro, redução de custos, projeção da marca ou aumento da fidelidade dos clientes, então a implementação desse caso dará origem a uma ou mais MMFs.

Por outro lado, se as funcionalidades descritas em um caso de uso não geram valor para o negócio, mas são necessárias para a especificação de outros casos de uso que geram valor, então estes casos de uso darão origem a um ou mais AEs.

Note que, de forma geral, um caso de uso pode gerar uma ou mais MMFs e AEs. Entretanto, no modelo de caso de uso descrito na Figura 2.1, cada caso de uso dá origem a exatamente uma MMF ou AE. Em conseqüência, deste ponto em diante, os casos de uso apresentados na Figura 2.1, serão referenciados como unidades de software, ou mesmo como MMFs ou AEs. As razões específicas pelas quais essas

unidades de *software* são MMFs e AEs são objeto de discussão no exemplo apresentado no Capítulo 4.

2.2.2 Fluxo de caixa de projetos de *software*

O *fluxo de caixa* de uma empresa refere-se às despesas realizadas e receitas auferidas durante um conjunto de períodos de tempo subjacentes, onde cada período possui a mesma duração. Seguindo a mesma linha de raciocínio, o fluxo de caixa de uma unidade de *software* refere-se às despesas e receitas geradas por aquela unidade durante um conjunto de períodos de tempo subjacentes e de igual tamanho. Esse conjunto de períodos é chamado de *janela de oportunidade* do projeto do qual aquela unidade faz parte. Hubbard (HUBBARD, 2007) mostra como os elementos de fluxo de caixa do projeto de *software* podem ser estimados de forma apropriada.

O fim da janela de oportunidade marca o instante de tempo no qual o *software* se torna obsoleto e será substituído por uma solução mais vantajosa. Em termos formais, uma janela de oportunidade J é um conjunto $\{p_1, p_2, ..., p_n\}$ de períodos subjacentes de mesma duração. A Tabela 2.2 apresenta os elementos de fluxo de caixa das unidades de *software* (MMFs e AEs) apresentados na Figura 2.1. Na Tabela 2.2, $J = \{1, 2, ..., 15\}$.

Observe que todas as MMFs geram ganhos após um investimento inicial, enquanto que o AE só apresenta custo no seu fluxo de caixa. Obviamente, o valor que uma MMF traz para o projeto é fortemente afetado pelo comportamento do seu fluxo de caixa no decorrer do tempo.

O fluxo de caixa de uma unidade de *software* v é dado por $fc(v)$ e o elemento do fluxo de v no período $p \in J$ é dado por $fc(v, p)$. Na Tabela 2.2, $fc(UC_1)$ no período

Tabela 2.2: Elementos de fluxo de caixa de unidades de *software*

Elementos de Fluxo de Caixa (US$ 1.000)					
Unid. de *Software*	**Período**				
Id	**1**	**2**	**3**	**4 to 14**	**15**
UC_1	-50	50	70	100	50
UC_2	-70	20	28	40	20
UC_3	-30	500	700	1,500	1,000
UC_4	-40	200	280	600	200
UC_5	-80	90	120	180	90
UC_6	-10	0	0	0	0

1, ou $fc(UC_1, 1)$, é -50, e $fc(UC_1)$ no período 2, ou $fc(UC_1, 2)$, é 50.

2.2.3 Fluxo de caixa descontado

Obviamente, o valor financeiro de uma MMF é a soma dos elementos do seu fluxo de caixa. Entretanto, uma vez que não é apropriado executar operações matemáticas sobre valores monetários em diferentes instantes de tempo sem considerar uma taxa de juros (FABOZZI; DAVIS; CHOUDHRY, 2006), é necessário calcular o seu fluxo de caixa descontado (FCD). Em termos formais, dado:

- um conjunto de elementos fc_1, fc_2, fc_3 ... fc_n de um mesmo fluxo de caixa,
- uma taxa de juros i, e
- um conjunto de períodos consecutivos p_1, p_2, p_3 ... p_n.

O fluxo de caixa descontado é determinado pela seguinte fórmula:

$$\frac{fc_1}{(1+i)^1}, \frac{fc_2}{(1+i)^2}, \frac{fc_3}{(1+i)^3}, \ldots, \frac{fc_n}{(1+i)^n}$$

Considerando o fluxo de caixa apresentado na Tabela 2.2 e uma taxa de juros de 2%, o fluxo de caixa descontado da unidade de *software* UC1 é obtido da seguinte forma:

$$\frac{-50}{(1+\frac{2}{100})^1} = -49, \frac{50}{(1+\frac{2}{100})^2} = 48, \frac{70}{(1+\frac{2}{100})^3} = 66, \cdots, \frac{50}{(1+\frac{2}{100})^{15}} = 37$$

Assim sendo, a Tabela 2.3 mostra o fluxo de caixa descontado de todas as unidades de *software* apresentadas na Tabela 2.2, considerando que todas as unidades de *software* pudessem ser desenvolvidas no primeiro período. Para facilitar o entendimento, os valores apresentados na Tabela 2.3 foram arredondados para o valor inteiro mais próximo. Os demais valores apresentados nesta dissertação seguem a mesma convenção.

Tabela 2.3: Fluxo de caixa descontado de cada unidade de *software* caso pudesse ser desenvolvida no primeiro período

Elementos de Fluxo de Caixa (US$ 1.000)						
Unid. de *Software*	**Período**					
Id	**1**	**2**	**3**	**4**	**...**	**15**
UC_1	-49	48	66	92	...	37
UC_2	-69	19	26	37	...	15
UC_3	-29	481	660	1386	...	743
UC_4	-39	192	264	554	...	149
UC_5	-78	87	113	166	...	67
UC_6	-10	0	0	0	...	0

2.2.4 Valor presente líquido

Valor presente líquido, ou VPL, representa a soma do fluxo de caixa descontado (DENNE; CLELAND-HUANG, 2004b).

Em termos formais, o VPL de uma unidade de *software* v, cujo desenvolvimento

inicia no período $t \in P$, considerando uma taxa de juros contante, é dado por

$$vpl(v,t) = \sum_{j=t}^{n} \frac{fc(v, j-t+1)}{(1+i)^j},$$

Onde i é a taxa de juros e n é o último período da janela de oportunidade J.

Como exemplo, observamos que se o desenvolvimento do MMF UC_1 começasse no período 1, a uma taxa de juros i de 2% por período, então seu VPL será

$$vpl(UC_1, 1) = \frac{-50}{(1+\frac{2}{100})^1} + \frac{50}{(1+\frac{2}{100})^2} + \frac{70}{(1+\frac{2}{100})^3} + \ldots + \frac{50}{(1+\frac{2}{100})^{15}}$$

$$= \text{-49 + 48 + 66 + } \ldots \text{+ 37}$$

$$= \$987 \text{ mil}$$

No entanto, se observarmos a Tabela 2.4 , caso o desenvolvimento de UC_1 começasse no período 2, então seu VPL seria diferente, ou seja

$$vpl(UC_1, 2) = \frac{-50}{(1+\frac{2}{100})^2} + \frac{50}{(1+\frac{2}{100})^3} + \frac{70}{(1+\frac{2}{100})^4} + \ldots + \frac{100}{(1+\frac{2}{100})^{15}}$$

$$= \text{-48 + 47 + 65 + } \ldots \text{+ 74}$$

$$= \$894 \text{ mil}$$

A Tabela 2.5 mostra o VPL de cada unidade de *software* apresentada na Figura 2.2 considerando o período no qual cada uma é desenvolvida. Como existem seis unidades para serem desenvolvidas e cada unidade é desenvolvida em exatamente um

Tabela 2.4: Fluxo de caixa descontado de cada unidade de *software* caso pudesse ser desenvolvida no segundo período

Elementos de Fluxo de Caixa Descontado(US$ 1.000)							
Unid. de *Software*	**Período**						
Id	**1**	**2**	**3**	**4**	**5**	**...**	**15**
UC_1	0	-48	47	65	91	...	74
UC_2	0	-67	19	26	36	...	30
UC_3	0	-29	471	647	1359	...	1115
UC_4	0	-38	188	259	543	...	446
UC_5	0	-77	85	111	163	...	134
UC_6	0	-10	0	0	0	...	0

período de tempo, a Tabela 2.5 apresenta o VPL dessas unidades nos seis primeiros períodos de tempo do projeto de desenvolvimento do sistema de controle de empréstimos consignados.

Tabela 2.5: Valor presente líquido de unidades de *software* de acordo com o período de início do seu desenvolvimento

Valor Presente Líquido (US$ 1.000)						
Unid. de *Software*	**Período**					
Id	**1**	**2**	**3**	**4**	**5**	**6**
UC_1	1.045	987	894	802	712	623
UC_2	368	346	309	274	239	204
UC_3	16.001	14.944	13.537	12.157	10.804	9.478
UC_4	6.221	5.950	5.388	4.836	4.296	3.766
UC_5	1.885	1.781	1.613	1.447	1.285	1.126
UC_6	-10	-10	-10	-9	-9	-9

2.2.5 A relação de dependência

Uma vez que a construção de uma MMFs ou AEs pode depender da construção de outras unidades de *software*, é importante explicitar a relação de dependência que existe entre eles. Relações de precedência entre MMFs e AEs podem ser represen-

tadas por um grafo direcionado acíclico que restringe o desenvolvimento do projeto. Uma introdução à teoria de grafos pode ser encontrada em (GROSS; YELLEN, 2005).

Podemos observar um desses grafos na Figura 2.2. No grafo, as MMFs são representadas pelas unidades de *software* UC_1, UC_2, UC_3, UC_4 e UC_5; enquanto que os AEs são representados pela unidade UC_6.

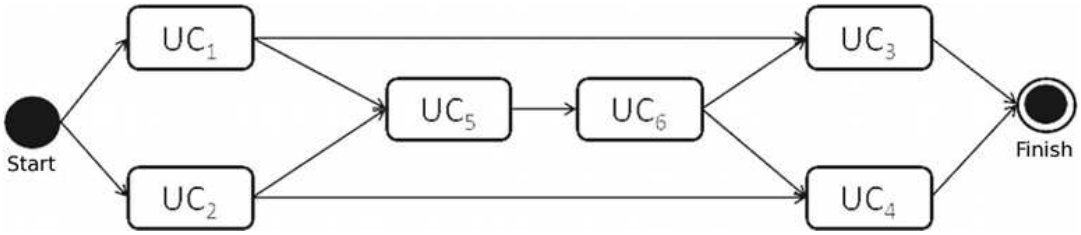

Figura 2.2: Diagrama de dependência das unidades de *software* do sistema de controle de empréstimos.

Tabela 2.6: Opções de planejamento

Opções de Agendamento	**Periodo**					
	1	**2**	**3**	**4**	**5**	**6**
1	UC_1	UC_2	UC_5	UC_6	UC_3	UC_4
2	UC_1	UC_2	UC_5	UC_6	UC_4	UC_3
3	UC_2	UC_1	UC_5	UC_6	UC_3	UC_4
4	UC_2	UC_1	UC_5	UC_6	UC_4	UC_3

Uma seta de UC_1 para UC_2, isto é, $UC_1 \rightarrow UC_2$, indica que o desenvolvimento do módulo UC_1 deve ser concluído antes que o de UC_2 comece. O nó inicial (*Start*) e o final (*Finish*) servem apenas para marcar o começo e o término do projeto, respectivamente, e por isso, possuem duração e custo de desenvolvimento iguais a zero. A Tabela 2.6 mostra todas as possíveis seqüências de desenvolvimento para o sistema de controle de empréstimos, considerando que:

a) Cada unidade de *software* demora somente um período para ser desenvolvida;

b) Somente uma unidade de *software* pode ser desenvolvida de cada vez;

c) A primeira unidade de *software* é desenvolvida no período 1;

d) Não existe atraso entre o fim do desenvolvimento de uma unidade de *software* e o começo do próximo;

Em termos formais, para o IFM, um grafo de precedência é uma estrutura matemática $G(V_G, E_G)$, na qual:

- $V_G = \{v_1, v_2, \cdots, v_n\}$ é um conjunto de MMFs e AEs, onde cada uma das unidades do *software* possui tempo de desenvolvimento $t(v_i)$ e um fluxo de caixa associado $fc(v_i)$ avaliado para uma janela de oportunidade J, e
- E_G é um conjunto de pares ordenados, tal que se $(v_1, v_2) \in E_G$, então o desenvolvimento de v_2 depende do desenvolvimento de v_1.
- Se (v_1, v_2) e $(v_3, v_2) \in E_G$, então $v_1 = v_3$. Isto significa que cada unidade de *software* só pode ter um predecessor.

No grafo apresentado na Figura 2.2, $V_G = \{UC_1, UC_2, UC_3, UC_4, UC_5, UC_6\}$ e $E_G = \{(UC_1, UC_5), (UC_2, UC_5), (UC_5, UC_6), (UC_1, UC_3), (UC_6, UC_3), (UC_2, UC_4), (UC_6, UC_4)\}$. Note que os módulos UC_5, UC_3 e UC_4 possuem mais de uma dependência e, por isso, o grafo apresentado não atende à última cláusula do IFM. Casos de dependências múltiplas iguais a esse não são resolvidos pelo IFM.

O grafo de precedência apresentado na Figura 2.2 indica que apenas o UC_1 e o UC_2 podem ser desenvolvidos no primeiro período. Uma vez que neste exemplo, cada MMF precisa de exatamente um período para ser desenvolvido, o UC_5 não pode ser

desenvolvido até o terceiro período. Além disso, cada sequência em particular de MMF produz o seu próprio VPL.

Por exemplo, a sequência

$$UC_1 \rightarrow UC_2 \rightarrow UC_5 \rightarrow UC_6 \rightarrow UC_3 \rightarrow UC_4,$$

produz um VPL de \$17.564 mil. Já a seqüência

$$UC_2 \rightarrow UC_1 \rightarrow UC_5 \rightarrow UC_6 \rightarrow UC_3 \rightarrow UC_4,$$

produz um VPL de \$17.601 mil.

2.2.6 Limitações do IFM

Para encontrar um resultado em tempo polinomial, os algoritmos propostos pelo IFM se valem das seguintes limitações (DENNE; CLELAND-HUANG, 2004b):

- Cada unidade de *software* do projeto possui no máximo 1 predecessor;
- O método não trata de relações de precedências flexíveis;
- O método não considera a existência de unidades de *software* não essenciais (isto é, aquelas que não necessariamente precisam ser desenvolvidas durante o ciclo de vida do projeto de *software* - ver Seção 3.2);
- Não garantem que a solução gerada seja ótima.

O método proposto nesta dissertação supera essas limitações e fornece a solução ótima para projetos que utilizam dependências flexíveis e contam com unidades de *software* não essenciais. Para tanto faz uso de uma categoria de algoritmos chamada *Branch & Bound*.

2.3 Sobre o algoritmo Branch & Bound

De acordo com (LIBERTI, 2003) os algoritmos Branch & Bound são muito bem sucedidos e frequentemente aplicados a problemas de otimização não linear.

2.3.1 Como o Branch & Bound funciona

Dividir para conquistar é o conceito básico por trás deste método. Como o tamanho e a complexidade do problema original dificultam a solução direta, divide-se o problema sucessivamente em subproblemas cada vez menores até que apresentem tamanho e complexidade passíveis de conquista (solução).

A divisão (*branch*) é feita particionando-se o conjunto de soluções válidas em subconjuntos cada vez menores. A conquista é feita, em parte, calculando-se um limite (*bound*) de quão boa a melhor solução do subconjunto pode vir a ser. Subconjuntos são descartados quando seus limites indicam que não existe a possibilidade deles conterem uma solução ótima para o problema original.

Esta estratégia nos leva a um algoritmo composto por dois passos, que geram uma árvore de busca para encontrar a solução ótima. Enquanto o passo do *branch* é responsável por fazer com que a árvore cresça, o passo do *bound* é responsável por limitar esse crescimento (HILLIER; LIEBERMAN, 2001).

A decisão de podar a árvore utiliza duas funções denominadas ¨limite superior" e "limite inferior". A função limite superior de N, ou $ls(N)$, é uma estimativa otimista que indica o valor máximo da melhor solução que pode ser encontrada a partir de um nó candidato N da árvore de busca. A função limite inferior $li(N)$ faz justamente o oposto, isto é, estima o valor mínimo da melhor solução que pode ser encontrada a

partir de um nó candidato N.

Um nó só pode ser descartado quando o seu limite superior for menor que o maior limite inferior já encontrado pela busca. Neste caso, o ramo da árvore que contém esse nó nunca produzirá a solução ótima, pois existe outro ramo que contém uma solução melhor.

2.3.2 Classificação do método

Segundo (DEMEULEMEESTER; HERROELEN, 2002), geralmente é feita uma distinção entre duas estratégias de algoritmos *Branch & Bound*. Essa dicotomia se baseia na escolha do próximo nó onde será realizado o *branch*.

A primeira estratégia é a *depth-first*, ou *backtracking*, que seleciona um dos nós entre os que foram criados no estágio anterior. No entanto, se o último estágio tiver sido totalmente verificado, a busca retorna um nível acima, até encontrar um nó que não tenha sido totalmente explorado.

A segunda estratégia é tradicionalmente conhecida como *best-first*, *frontier search* ou *skiptracking*. Ela sempre seleciona o nó que apresenta o melhor valor de limite superior a cada ramificação da árvore de busca.

As principais vantagens e desvantagens de ambas as estratégias, de acordo com (DEMEULEMEESTER; HERROELEN, 2002; ZHANG, 2000), são apresentadas na Tabela 2.7.

A estratégia que o método proposto nesta dissertação utiliza na seleção de um nó para a realização do *branch* é a *Depth-First*. Um estudo feito pelo próprio (ZHANG,

Tabela 2.7: Comparativo entre as estratégias *Branch & Bound*

#	**Depth First**	**Best First**
1	Requer menos memória	Requer mais memória
2	Encontra uma solução viável, e geralmente boa, quase que imediatamente e melhora esse resultado ao longo da busca	Não fornece soluções válidas no decorrer da busca e a solução ótima só é obtida no final da busca
3	Em geral, quando sua execução é interrompida, é capaz de ainda assim fornecer uma boa solução para o problema	Em geral, nenhuma solução será obtida caso sua execução seja interrompida
4	Visitas aos nós geralmente são mais rápidas	Visitas aos nós costumam ser mais demoradas
5	Em geral, visita um número muito maior de nós	Em geral, visita um número menor de nós

2000) mostrou que a estratégia *Depth-First* se comporta melhor do que a *Best-First* em diversos cenários.

3 O PROBLEMA MAX-NPV COM UNIDADES DE SOFTWARE OPCIONAIS E PRECEDÊNCIAS FLEXÍVEIS

O problema determinístico MAX-NPV se refere à organização das atividades de um projeto dentro uma janela de oportunidade, a fim de maximizar o VPL de um projeto. O projeto é representado por um grafo de precedência, onde o conjunto de nós do grafo representa as unidades de *software* e o conjunto de arcos representa a restrição de precedência fim-inicio com um intervalo entre as atividades igual a zero (DEMEULEMEESTER; HERROELEN, 1992).

Dado um grafo de precedência $G(V_G, E_G)$ de um projeto P e uma unica equipe de desenvolvimento, uma seqüência de desenvolvimento $S = v_1, v_2 \cdots, v_n$ é válida se:

- $v_i \in V_G \Longleftrightarrow v_i \in S$;
- $v_i, v_j \in S \wedge i \neq j \Rightarrow v_i \neq v_j$;
- $(v_i, v_j) \in E_G \wedge S[k] = v_i \wedge S[l] = v_j \Rightarrow k < l$;

onde S[n] retorna o n-ésimo elemento da seqüencia S.

O problema MAX-NPV consiste em encontrar uma seqüência válida S tal que *VPL(S)* é máximo.

3.1 Precedências flexíveis

A relação de precedência flexível introduzida nesta dissertação nesta seção, é uma forma de representar dependências entre unidades de *software* de um projeto de *software*, onde nem todos os predecessores de uma unidade precisam estar desenvolvidos, para que esta unidade seja construída. Através do exemplo apresentado nesta dissertação, verificamos que uma unidade de *software* pode ter predecessores opcionais, que não são obrigatórios para a sua construção.

Essa descoberta gerou a necessidade de criar uma nova notação para esta relação. A nova notação utiliza o símbolo "+" (ou inclusivo) no grafo de precedência do projeto. Essa notação fornece um apoio importante na decisão da melhor ordenação do projeto, já que cria novas alternativas para que o gerente do projeto decida pela melhor seqüência. Essa representação também já existe em algumas outras linguagens de modelagem de *software* tais como *Unified Modelling Language* (UML) (BOOCH; RUMBAUGH; JACOBSON, 2005) e Graphical Evaluation and Review Technique (GERT), que apresenta inclusive outras formas de precedência flex'ivel.

Observe que na Figura 3.1, para que a unidade UC_5 seja construída, basta que o desenvolvimento de um de seus predecessores esteja concluído.

3.2 Unidades de *software* essencias e não essenciais

Se um projeto de *software* tem seus objetivos claramente estabelecidos, as unidades de *software* essencias são aquelas que se forem removidas do projeto diminuem as chances desses objetivos serem alcançados. Já as unidades de *software* não essencias, introduzidas nesta seção, são fundamentalmente as unidades que se forem removidas do projeto não comprometem as chances destes objetivos serem alcançados. O conceito de unidades não essenciais é análogo ao conceito de *feature* opcional comumente utilizado na modelagem de dominio de linhas de produto de *software* (POHL; BöCKLE; LINDEN, 2010).

Se o objetivo de um projeto de *software* pode ser expresso em sua totalidade através de indicadores financeiros, então as unidades de *software* essencias são o menor conjunto de unidades de *software* que permitem que as metas estabelecidas para cada um desses indicadores sejam alcançadas.

Conforme apresentado na seção 2.2, unidades de *software* são classificadas como *Minimum Marketable Feature* (MMFs) ou *Architectural Elements* (AEs).

Um bom exemplo é aquele que nos ensina e nos mostra mais do que aquilo que sabíamos. O exemplo do sistema de controle de empréstimos consignado nos per-

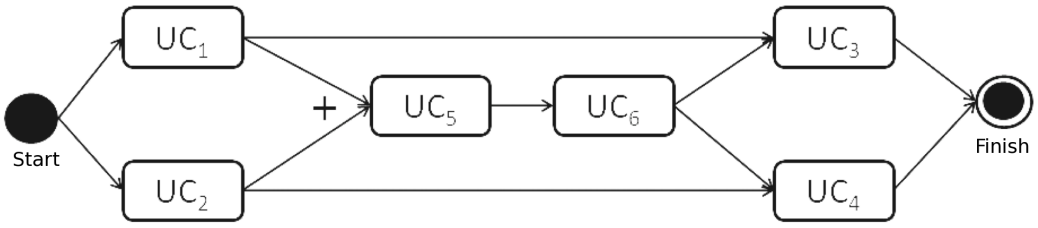

Figura 3.1: Diagrama de precedência do projeto de sistema de empréstimo consignado.

mitiu conhecer um novo comportamento das unidades de *software*. Essas unidades mostraram que quando são criadas, podem não gerar receita alguma por si só, e por isso deveriam ser considerados como AEs. No entanto, como sua implementação influencia diretamente o fluxo de caixa de outras unidades de *software*, elas podem ser considerados unidades do tipo NAEs (*Non Essential* AEs ou AEs não essenciais).

Em outra situação, uma MMF não essencial pode, ao entrar em execução, não só criar seu próprio fluxo de caixa como também influenciar o fluxo de caixa de uma MMF impactada por sua construção. Estas unidades são chamadas de NMMFs (Non Essential MMFs ou MMF não essenciais). O Capítulo 4 apresenta exemplos de NAEs e NMMFs.

3.3 Algoritmo BnBFlex

A seguir, apresentamos um conjunto de variáveis, funções e passos utilizados no algoritmo *Branch and Bound Flexível*, ou somente BnbFlex, baseado no algoritmo *Branch and Bound*, que descobre a seqüência de implementação de um projeto, composto por unidades de software essenciais e não essenciais, que provê o máximo retorno financeiro (MAX NPV) ao investidor.

3.3.1 Variáveis utilizadas

FINISH: Nó final da árvore.

limiteInferiorNo: Limite inferior do nó;

limiteSuperiorNo: Limite superior do nó.

maiorLimiteInferiorArvore: Valor utilizado pela função *Bound* para descartar nós que não têm a solução procurada;

noCandidato: Nó da árvore analisado a cada passagem do algoritmo;

nosCandidatos: Conjunto de nós com os possíveis sucessores de um determinado nó;

nosImpactantes: Conjunto de nós predecessores que produzem impacto financeiro sobre um determinado nó;

S: sequencia solução que retorna o MAX NPV;

vplMelhorSequencia: VPL da sequencia solução;

seqCorrente: sequencia temporária que representa o escalonamento dos nós de um determinado caminho da árvore de busca;

START: Nó inicial da árvore.

3.3.2 Funções Auxiliares

A seguir são apresentadas funções que são utilizadas com frequência no algoritmo desenvolvido para o método proposto nesta dissertação.

AtualizaNosCandidatos: Atualiza o conjunto de nós candidatos.

Branch: Escalona o conjunto de nós candidatos de um determinado nó obtidos através da função *ObtemNosCandidatos*.

Bound: descarta nó cuja solução é pior do que a que já existe e atualiza conjunto de nós candidatos da árvore de busca.

LimiteInferior: calcula o valor do limite inferior de um nó.

LimiteSuperior: calcula o valor do limite superior de um nó .

ObterMaiorLimiteInferior: Retorna o maior limite inferior da árvore.

ObterSequenciaCompleta: Retorna uma seqüência completa de um nó candidato.

ObterVPLSeq: calcula o VPL de uma determinada sequencia.

ObterNosImpactantes: Obtém o conjunto de nós predecessores impactantes para um determinado nó.

ObtemNosCandidatos: retorna os nós candidatos para o próximo nível da árvore de busca, considerando as restrições de precedência da unidade de *software* apresentada no grafo de precedência do projeto.

PredecessoresAtendidos: verifica se todos os predecessores obrigatórios de uma unidade de *software* foram atendidos, ou seja, se todos os seus predecessores já foram escalonados na árvore.

RecalculaVPL: recalcula o VPL de um nó da árvore impactado por uma unidade não essencial escalonada anteriormente na árvore.

3.3.3 Passos do algoritmo

Antes da aplicação do algoritmo, é necessário particionar o produto em unidades de *software* e classificá-las em unidade de *software* essenciais e não essenciais.

1. Avaliar *maiorLimiteInferiorArvore* aplicando a função *ObterMaiorLimiteInferior* para a árvore de busca;
2. $noCandidato \leftarrow START$;
3. *nosCandidatos* $\leftarrow$ *ObterNosCandidatos(noCandidato)*;
4. $S \leftarrow$ *ObterSequenciaCompleta(nosCandidatos[0])*.
5. *vplMelhorSequencia* $\leftarrow$ *ObterVPLSeq(S)*;
6. *AtualizaNosCandidatos(nosCandidatos)*;
7. Enquanto *nosCandidatos* $\neq \emptyset$:

   7.1 Para cada nó candidato:

   noCandidato $\leftarrow$ *nosCandidatos[i]*.

   nosImpactantes $\leftarrow$ *obterNosImpactantes(noCandidato)*.

   Se *nosImpactantes* $\neq \emptyset$ então

   - somar os impactos exercidos sobre o nó
   - atualizar o fluxo de caixa do nó;
   - recalcular o VPL do nó.

   limiteInferiorNo $\leftarrow$ *LimiteInferior(noCandidato)*;

   limiteSuperiorNo $\leftarrow$ *LimiteSuperior(noCandidato)*;

Se o *limiteSuperiorNo* < *maiorLimiteInferiorArvore* então
Bound(noCandidato).
Senão
Branch(noCandidato)
seqCorrente ← *seqCorrente* + *noCandidato*.
Se o *limiteInferiorNo* > *maiorLimiteInferiorArvore* então
maiorLimiteInferiorArvore ← *limiteInferiorNo*;
Se *ObterNPVSeq(seqCorrente)* > *vplMelhorSequencia* então
vplMelhorSequencia ← *seqCorrente*.
S ← *seqCorrente*.

7.2 *AtualizaNosCandidatos(nosCandidatos)*;

8 Retorna a sequencia S como solução.

3.3.4 A Função de Limite Superior

A função de limite superior retorna uma estimativa otimista do maior VPL que ser obtido a partir de um determinado nó da árvore de busca. Esta estimativa é otida pela soma do VPL das unidades de *software* já executadas com o maior VPL das unidades ainda não executadas.

A função de limite superior verifica se o nó da árvore corresponde a uma unidade de *software* impactada por uma ou mais unidades não essenciais. Se a unidade for impactada, a função verifica se estas unidades não essenciais já foram escalonadas. Se essas unidades não essenciais já foram escalonadas na árvore de busca, o VPL da unidade correspondente ao nó atual é atualizado considerando o impacto que as unidades não essenciais exercem sobre ela.

3.3.5 A Função de Limite Inferior

A função de limite inferior retorna uma estimativa pessimista do menor VPL que se pode encontrar nos escalonamentos gerados a partir de um determinado nó da árvore de busca. Para isso, soma-se o VPL dos nós já escalonados ao menor VPL dos nós candidatos da árvore que ainda estão disponíveis.

Assim como a função de limite superior, a função de limite inferior soma o VPL das unidades de *software* já escalonadas, verifica se o nó da árvore corresponde a uma unidade impactada por uma ou mais unidades não essenciais. Se a unidade é impactada, a função verifica se estas unidades não essenciais já foram escalonadas na árvore de busca. Se já foram escalonadas, o VPL da unidade é atualizado considerando o impacto que as unidades não essenciais exercem sobre ela.

3.3.6 A Função de *Branch*

Retorna o conjunto de nós candidatos de um determinado nó da arvore de busca.

3.3.7 A Função de *Bound*

Analisa o limite superior de cada nó escalonado na árvore e descarta os nós cujo limite superior é menor que o maior limite inferior armazenado durante a construção da árvore.

4 EXEMPLO DE APLICAÇÃO

De acordo com Seneca (4 BC - 65 AD), o filósofo romano:

"O aprendizado através de normas e regras é longo, através do exemplo é curto e eficaz".

Em conseqüência, as considerações apresentadas nesta dissertação são introduzidas passo a passo com a ajuda de um exemplo inspirado no mundo real relacionado ao desenvolvimento de um aplicativo para dispositivo móvel[1].

4.1 Informações de Contexto

Empréstimos consignados são empréstimos de utilidade geral, de baixo risco e baixa taxa de juros, concedidos a funcionários qualificados de empresas associadas a uma instituição financeira, nos quais os pagamentos são feitos em parcelas deduzidas de seus salários (PETERSON, 2008).

[1] O exemplo foi inspirado em um projeto desenvolvido por uma grande instituição financeira sul-americana, que solicitou que seu nome não fosse divulgado.

Neste sentido, considere uma instituição financeira internacional, tal como CITIBANK, BARCLAYS, HSBC, ABN AMRO, UBS e muitas outras que fazem empréstimos consignados a seus clientes. Para o propósito desta dissertação, esta organização será chamada de LOANS "R"US, or $L_R U$.

Assim que um pedido de um empréstimo é recebido, a $L_R U$ calcula a probabilidade do cliente pagar o empréstimo solicitado de acordo com valores e datas pré-estabelecidas. Em seguida, a $L_R U$ verifica se há fundos suficientes para conceder o empréstimo que o cliente está solicitando, já que legalmente as instituições financeiras não podem conceder empréstimos acima de um determinado limite, de modo a preservar sua saúde financeira. Finalmente, são apresentadas ao cliente as condições nas quais um empréstimo consignado pode ser concedido, se houver. Neste ponto, há a opção de aceitar ou recusar a oferta do empréstimo.

As instituições financeiras que oferecem empréstimo consignado também tendem a oferecer serviços de refinanciamento para contratos de empréstimos existentes como forma de aliviar a pressão financeira de clientes atuais e futuros. A $L_R U$ não é exceção a essa regra. Como a operação de refinanciamento pode envolver uma terceira instituição, frequentemente essa operação é chamada de "compra de dívida".

Para oferecer uma resposta competitiva adequada aos movimentos recentes da concorrência no mercado de empréstimos consignado e desenvolver ainda mais seus negócios de empréstimos, a $L_R U$ decidiu criar um novo sistema de empréstimo consignado para dispositivos móveis baseado na internet.

A empresa acredita que, se agir rapidamente, além de aumentar o seu faturamento consideravelmente, esse sistema pode redefinir favoravelmente o cenário competitivo dos negócios de empréstimos consignado. A Figura 2.1 apresentou um modelo contendo os casos de uso que formam o novo sistema de empréstimo consignado.

4.2 Determinando Como Cada Unidade de *Sofware* Pode Gerar Receita

A Tabela 2.1 descreve o significado e o tipo de cada unidade de *software* apresentado na Figura 2.1. Observa-se que todos os cinco primeiros casos de uso listados na tabela são unidades de *software* autônomas e que estas unidades geram receita para a L_RU da seguinte forma:

- UC_1 *e* UC_2 - pergunta a cada cliente que solicita um empréstimo consignado se concorda em receber ocasionalmente ofertas de novos produtos da L_RU e de seus parceiros comerciais.

  Novas campanhas de marketing podem ser aplicadas aos clientes que concordam, gerando novas oportunidades de vendas para a L_RU e receita na forma de taxas devidas ao uso da base de dados de clientes da L_RU por empresas parceiras;

- UC_3 *e* UC_4 - assim que um cliente aceita uma oferta de empréstimo ou refinanciamento, é gerada receita na forma de juros;

- UC_5 - os resultados do caso de uso "Análise rápida de crédito" são usados para enriquecer a base de dados de clientes da L_RU.

  As informações de análise de crédito são particularmente importantes para campanhas de marketing que dependem da saúde financeira dos clientes, especialmente as campanhas que permitem que o cliente pague por um produto ou serviço em parcelas. As empresas associadas à L_RU pagam uma taxa especial para acessar a base de dados enriquecida.

Portanto os casos de uso UC_1, UC_2, $\cdots$, UC_5 são, na verdade, MMFs. Por outro lado, UC_6, "Verificar a disponibilidade de fundos" é o único AE entre os casos de uso

na Figura 2.1, já que apresenta um serviço essencial ao comportamento desejado do sistema. No entanto, nem os clientes nem as empresas associadas estão dispostas a pagar por este serviço, uma vez que UC_6 não gera receita alguma.

4.3 Planejando a Implementação do Sistema

Embora a equipe da L_RU responsável pela execução do projeto de construção do sistema de empréstimo consignado tenha considerado inicialmente adotar o diagrama de precedência da Figura 2.2, logo perceberam que o comportamento da unidade UC_5 é independente do tipo de empréstimo que está sendo solicitado: novo empréstimo ou refinanciamento. Portanto, o desenvolvimento da unidade UC_5 pode começar quando o desenvolvimento das unidades UC_1 ou UC_2 estiver completo, ou mesmo quando o desenvolvimento de ambas estiver completo.

Considerando que essa nova visão das dependências requeridas por UC_5 é, na verdade, uma opção real, e que as opções poderem mudar substancialmente o valor de projetos de *software* (FICHMAN; KEIL; TIWANA, 2005), a equipe de projeto da L_RU decidiu alterar o diagrama de precedência apresentado na Figura 2.2. A Figura 3.1 mostra o diagrama de precedência atualizado para o projeto de construção do sistema de empréstimo consignado.

Observa-se que o requisito de precedência flexível do UC_5 é apropriadamente sinalizado pela presença do símbolo "+" (ou inclusivo) no diagrama. Também é importante observar que, de acordo com as condições atuais de mercado, as unidades apresentadas na Figura 3.1 formam o conjunto de unidades de *software* que são essenciais para o desenvolvimento do sistema de empréstimo consignado, ou seja este é o menor conjunto de unidades autônomas que tem alguma chance de apresentar retorno a longo prazo ao investimento a ser feito pela L_RU em seu desenvolvimento

(HIBBS; JEWETT; SULLIVAN, 2009).

Note que, neste caso, as condições de mercado foram determinadas através de pesquisa de mercado encomendada pela equipe de analistas de negócios da L_RU (BURNS; BUSH, 2009).

A Tabela 4.1 indica todas as sequências possíveis de desenvolvimento do sistema de empréstimo consignado. Existem ao todo quatorze possibilidades. É importante mencionar que como as unidades UC_1, UC_2, $\cdots$, UC_6 são todas essenciais ao comportamento desejado do sistema, todas devem ser desenvolvidas.

Tabela 4.1: Opções de Planejamento

Opções de Planejamento	**Período**					
	1	**2**	**3**	**4**	**5**	**6**
1	UC_1	UC_2	UC_5	UC_6	UC_3	UC_4
2	UC_1	UC_2	UC_5	UC_6	UC_4	UC_3
3	UC_1	UC_5	UC_2	UC_6	UC_3	UC_4
4	UC_1	UC_5	UC_2	UC_6	UC_4	UC_3
5	UC_1	UC_5	UC_6	UC_2	UC_3	UC_4
6	UC_1	UC_5	UC_6	UC_2	UC_4	UC_3
7	UC_1	UC_5	UC_6	UC_3	UC_2	UC_4
8	UC_2	UC_1	UC_5	UC_6	UC_3	UC_4
9	UC_2	UC_1	UC_5	UC_6	UC_4	UC_3
10	UC_2	UC_5	UC_1	UC_6	UC_3	UC_4
11	UC_2	UC_5	UC_1	UC_6	UC_4	UC_3
12	UC_2	UC_5	UC_6	UC_1	UC_3	UC_4
13	UC_2	UC_5	UC_6	UC_1	UC_4	UC_3
14	UC_2	UC_5	UC_6	UC_4	UC_1	UC_3

4.4 Avaliação de Diferentes Opções de Planejamento

A Tabela 4.2 mostra os elementos de fluxo de caixa não-descontados de cada MMF e AE no modelo apresentado na Figura 3.1 conforme estimado pela equipe de projeto

da $L_R U$.

Uma vez que a unidade UC_5 gera receita de acordo com o número de clientes existentes na base de dados de clientes da $L_R U$ (vide Seção 4.3), seus elementos de fluxo de caixa variam de acordo com o contexto no qual UC_5 é desenvolvido, já que as relações de dependência entre UC_5 e seus predecessores são flexíveis. Quanto maior o número de clientes na base de dados, maior a receita gerada por UC_5. Portanto

- Se UC_1 preceder o desenvolvimento de UC_5, o que é indicado por $UC_1 \rightarrow UC_5$ na Tabela 4.2, então UC_5 produz US$ 60 mil no segundo período, US$ 90 mil no terceiro período, US$ 130 mil no quarto período, e assim sucessivamente;
- Se for UC_2 que preceder o desenvolvimento de UC_5, ou seja, $UC_2 \rightarrow UC_5$, então o UC_5 produz valores menores em cada período. Produz US$ 20 mil no segundo período, US$ 30 mil no terceiro período, US$ 50 mil no quarto período, e assim sucessivamente;
- Contudo, se tanto UC_1 quanto UC_2 precederem o desenvolvimento de UC_5, ou seja, (UC_1,UC_2) $\rightarrow UC_5$, então o valor produzido por UC_5 em cada período atinge os seus valores mais altos. UC_5 produz US$ 90 mil no segundo período, US$ 120 mil no terceiro período, US$ 180 no quarto período, e assim sucessivamente.

Em todas as circunstâncias, o investimento necessário pelo UC_5 permanece o mesmo, isto é US$ 80 mil.

A Tabela 4.3 relaciona as opções de planejamento apresentadas na Tabela 4.1 com os seus respectivos VPLs. Por exemplo, na sétima opção, UC_1 é desenvolvido primeiramente. Então, UC_5, UC_6, UC_3, UC_2 e UC_4 são desenvolvidas no segundo, terceiro,

Tabela 4.2: Elementos de fluxo de caixa não-descontados das unidades de *sofware*

Elementos de Fluxo de Caixa (US$ 1.000)					
Unid. de *Software*	**Período**				
	1	**2**	**3**	**4 to 14**	**15**
UC_1	-50	50	70	100	50
UC_2	-70	20	28	40	20
UC_3	-30	500	700	1,500	1,000
UC_4	-40	200	280	600	200
$UC_1 \rightarrow UC_5$	-80	60	90	130	60
$UC_2 \rightarrow UC_5$	-80	20	30	50	20
$(UC_1,UC_2) \rightarrow UC_5$	-80	90	120	180	90
UC_6	-10	0	0	0	0

quarto, quinto e sexto períodos, respectivamente, produzindo um VPL de US$18.460 mil, que é o VPL mais alto entre as opções de planejamento e, como resultado, a escolha lógica da equipe de projetos da L_RU.

Obviamente, este VPL só pode ser alcançado porque a L_RU administra os seus negócios de empréstimos com o apoio de processos de negócios confiáveis, que podem ser usados temporariamente para apoiar o desenvolvimento do sistema de empréstimo consignado.

Por exemplo, quando o desenvolvimento de UC_1 ou UC_2 estiver completo, os clientes não precisarão esperar pelo desenvolvimento das unidades de *software* restantes para serem informados sobre as condições nas quais pode ser concedido um empréstimo. Neste ponto, os processos de empréstimos da L_RU fornecem uma resposta adequada. Isso se mantém para qualquer combinação de unidades de *software* previamente desenvolvidas.

Tabela 4.3: VPL da Opção de Planejamento

Opção de Planejamento	**VPL (US$ 1,000)**
1	17.564
2	16.769
3	17.198
4	16.403
5	17.712
6	16.346
7	18.460
8	17.601
9	16.806
10	16.345
11	15.550
12	16.160
13	15.364
14	15.814

4.5 Lidando com Unidades de *Software* Não-Essenciais

O bom senso e a experiência mostram que há duas formas principais de se perder dinheiro no mercado de empréstimos: emprestar dinheiro àqueles que não pagarão os seus empréstimos e deixar de conceder empréstimos àqueles que pagarão (SIDDIQI, 2005).

Portanto, antes de fazer uma opção de planejamento, em especial com base em um modelo que considere apenas unidades de *software* essenciais, a equipe de projetos da L_RU decidiu examinar o efeito das unidades não essenciais que lidam com estes dois aspectos do negócio de empréstimos.

A Figura 4.1 mostra um novo modelo de casos de uso que contém as unidades de *software* do sistema de empréstimo consignado juntamente com as unidades de *software* não essenciais identificadas pela equipe de projetos.

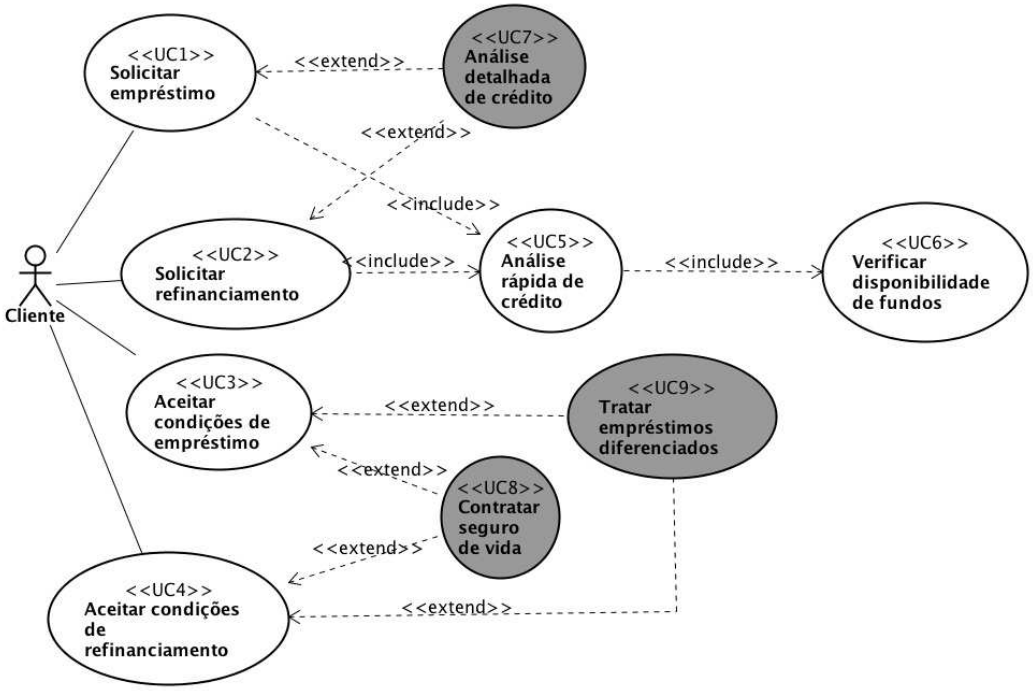

Figura 4.1: Modelo contendo unidades de *software* essenciais e não essenciais.

A Tabela 4.4 descreve as unidades de *software* não essenciais apresentadas na Figura 4.1. Há somente uma MMF não essencial (NMMF) entre as unidades de *software* não essenciais, ou seja, UC_7, "Análise detalhada de crédito", que enriquece ainda mais a base de dados de clientes da L_RU com informações que abrem oportunidades para novas campanhas de marketing, tanto da L_RU quanto de empresas associadas.

Como a receita gerada pela unidade UC_7 depende do número de clientes aprovados, é razoável afirmar que quanto mais clientes tiverem seu crédito analisado, maior será a receita gerada.

Assim sendo, a receita gerada por UC_7 é influenciada diretamente pelo desenvolvimento das unidades UC_1,"Solicitar empréstimo", e UC_2, "Solicitar refinanciamento", assim como UC_5, "Análise rápida de crédito". A Tabela 4.5 apresenta os elementos de fluxo de caixa de UC_7. A receita gerada por esta unidade de *software* em particular se dá através das empresas associadas à L_RU que pagam uma taxa especial para acessar a base de dados de clientes enriquecida.

Tabela 4.4: Descrições das unidades de *software* não essenciais

Unidade de *software*		
Id	**Nome**	**Descrição**
UC_7	Análise detalhada de crédito	Executa uma análise de crédito mais detalhada em pedidos que foram rejeitados pela "Análise rápida de crédito", a fim de conceder o empréstimo a estes clientes, caso isso seja possível
UC_8	Contratar seguro de vida	Para conceder os empréstimos, inclui o custo de um seguro de vida em pedidos de empréstimo rejeitados pela "Análise rápida de crédito" devido a baixa expectativa de vida desses clientes
UC_9	Lidar com empréstimos aprovados	Determina a data mais provável na qual a L_RU pode conceder um pedido de empréstimo ou refinanciamento que foi rejeitado por "Verificar disponibilidade de fundos" devido a uma falta de fundos temporária

Surpreendentemente, esta não é a única forma em que UC_7 contribui para o valor do projeto de construção do sistema de empréstimo consignado. Se implementada, a unidade "Análise detalhada de crédito" aumenta o número de pedidos de empréstimos e refinanciamentos aprovados. Quanto mais pedidos de empréstimos e refinanciamentos forem aprovados, maior o retorno produzido pelas unidades "Aceitar condições de empréstimo" e "Aceitar condições de refinanciamento". Portanto, a unidade de *software* "Análise detalhada de crédito" também contribui para o aumento da receita gerada por essas duas outras unidades de *software*.

Tabela 4.5: Elementos de fluxo de caixa de UC_7

Elementos de Fluxo de Caixa (US$ 1.000)					
Unidade de *Software* Id	**Período**				
	1	**2**	**3**	**4 to 9**	**10**
$UC_1 \rightarrow UC_7$	-90	65	75	90	65
$UC_2 \rightarrow UC_7$	-90	25	30	35	25
$(UC_1,UC_2) \rightarrow UC_7$	-90	90	100	130	90

Por outro lado, "Lidar com empréstimos aprovados" e "Contratar seguro de vida" são unidades autônomas cujos serviços nem os clientes nem as empresas associadas desejam pagar. Portanto, são, de fato, elementos de arquitetura não essenciais, ou NAEs, para abreviar. Apesar destas duas unidades não gerarem receita por si mesmas, contribuem para o valor do sistema de empréstimo consignado influenciando a receita gerada por outras unidades.

De modo semelhante a "Análise detalhada de crédito", "Lidar com empréstimos aprovados" e "Contratar seguro de vida" aumentam o número de pedidos de empréstimos e refinanciamentos aprovados. Como resultado, aumentam a receita gerada por "Aceitar condições de empréstimo" e "Aceitar condições de refinanciamento". A Tabela 4.6 apresenta o impacto estimado do desenvolvimento de unidades de *software* não essenciais sobre a receita gerada por unidades de *software* essenciais.

Tabela 4.6: O impacto estimado de unidades de *software* não essenciais sobre a receita gerada por unidades de *software* essenciais

		Unidade de *Software* Essencial	
		UC_3 (Aceitar condições de empréstimo	**UC_4** (Aceitar condições de refinanciamento)
Unidade de *Software* Não-Essencial	**UC_7** (Efetuar análise detalhada de crédito)	+20%	+15%
	UC_8 (Contratar seguro de vida)	+15%	+10%
	UC_9 (Lidar com empréstimos aprovados)	+25%	+20%

Por exemplo, de acordo com as informações exibidas na Tabela 4.6, a partir do momento no qual "Análise de crédito detalhada" é implementado, a receita gerada por "Aceitar condições de empréstimo" e "Aceitar condições de refinanciamento" aumenta em 20% e 15%, respectivamente.

4.6 Replanejando a Implemantação do Software

A Figura 4.2 apresenta o novo diagrama de precedência do sistema de empréstimo consignado. Este diagrama leva em consideração a existência de unidades de *software* essenciais e não essenciais. A linha tracejada ao redor da identificação da unidade tal como UC_7 ou UC_8 indica uma unidade de *software* não essencial, que pode ser desenvolvida ou não.

Contudo, se um caso de uso não essencial for desenvolvido, as dependências que este cria devem ser satisfeitas. Por exemplo, se nem UC_7 nem UC_8 forem desenvolvidas,

Tabela 4.7: Novas Opções de Planejamento

#	**Período**								
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
1	UC_1	UC_2	UC_5	UC_6	UC_3	UC_4			
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
81	UC_2	UC_1	UC_5	UC_7	UC_8	UC_6	UC_3	UC_4	
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
144	UC_2	UC_5	UC_7	UC_9	UC_6	UC_1	UC_8	UC_4	UC_3

o desenvolvimento de UC_6 pode começar imediatamente após o desenvolvimento do UC_5 estar completo. Porém, se UC_7 for desenvolvida e UC_8 não for, o desenvolvimento de UC_6 só pode começar quando o desenvolvimento de UC_5 e UC_7 estiver completo, e assim por diante. A Tabela 4.7 apresenta as opções de planejamento do sistema de empréstimo consignado, levando em consideração a existência de casos de uso essenciais e não essenciais. Há ao todo 144 diferentes opções de planejamento.

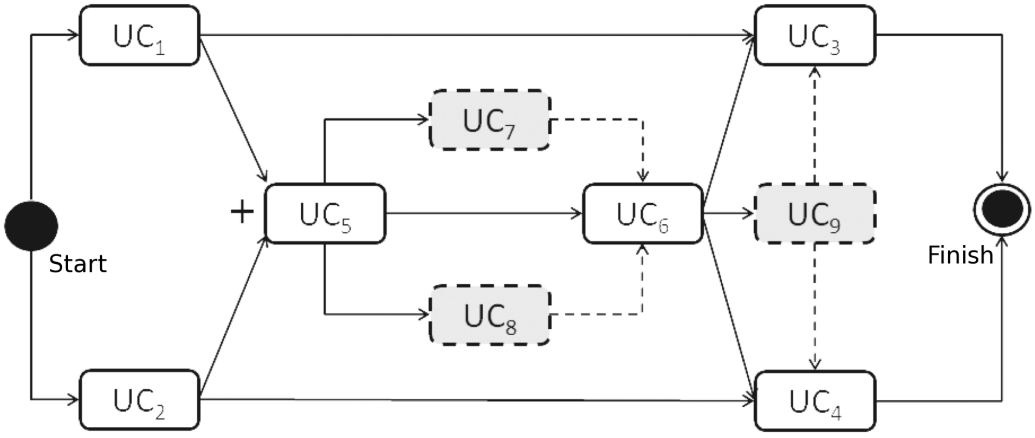

Figura 4.2: Um diagrama de precedência contendo unidades de *software* essenciais e não essenciais.

4.7 Qual Opção de Planejamento Apresenta o VPL Mais Alto?

A Tabela 4.8 exibe os elementos de fluxo de caixa não-descontados de cada MMF, AE, NMMF e NAE apresentados na Figura 4.2, conforme estimado pela equipe de projetos da L_RU.

Tabela 4.8: Novos elementos de fluxo de caixa da unidade de *software*

Novos Elementos de Fluxo de Caixa (US$ 1.000)					
Unid. de *Software*	**Período**				
Id	**1**	**2**	**3**	**4 to 9**	**10**
UC_1	-50	50	70	100	50
UC_2	-70	20	28	40	20
UC_3	-30	500	700	1,500	1,000
$UC_7 \rightarrow UC_3$	-30	600	840	1,800	1,200
$UC_8 \rightarrow UC_3$	-30	625	875	1,875	1,250
$UC_9 \rightarrow UC_3$	-30	575	805	1,725	1,150
$(UC_7,UC_8) \rightarrow UC_3$	-30	725	1,015	2,175	1,450
⋮	⋮	⋮	⋮	⋮	⋮
UC_9	-10	0	0	0	0

A Tabela 4.9 exibe o VPL de cada MMF, AE, NMMF e NAE apresentado na Figura 4.2 de acordo com o período no qual é desenvolvido, levando em consideração uma taxa de juros de 2% por período. Observa-se que o número de períodos que se leva para desenvolver todo o sistema aumentou de seis para nove períodos, devido à possível introdução de MMFs e AEs não essenciais (vide Tabela 4.7).

Entre todas as opções de planejamento apresentadas na Tabela 4.7, a opção mais lógica para a L_RU é desenvolver as unidades de *software* que compõem o sistema de empréstimo consignado na seguinte ordem:

$$UC_2 \rightarrow UC_1 \rightarrow UC_5 \rightarrow UC_7 \rightarrow UC_8 \rightarrow UC_6 \rightarrow UC_3 \rightarrow UC_4,$$

Tabela 4.9: VPLs de unidades de *software* essenciais e não essenciais

Valor Presente Líquido (US$ 1.000)					
Unid. de *Software*	**Período**				
Id	**1**	**2**	**3**	$\cdots$	**10**
UC_1	1,045	987	894	$\cdots$	369
UC_2	368	346	309	$\cdots$	105
UC_3	16,001	14,944	13,537	$\cdots$	5,653
$UC_7 \rightarrow UC_3$	19,207	17,939	16,250	$\cdots$	6,788
$C_8 \rightarrow UC_3$	20,009	18,688	16,928	$\cdots$	7,072
$UC_9 \rightarrow UC_3$	18,406	17,190	15,572	$\cdots$	6,504
$(UC_7,UC_8) \rightarrow UC_3$	23,215	21,683	19,641	$\cdots$	8,208
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
UC_9	-10	-10	-10	$\vdots$	-9

que produz um VPL de US$ 19.651.

Observe que esta opção de planejamento não contempla o desenvolvimento do UC_9, "Lidar com empréstimo aprovado". De acordo com as estimativas geradas pela equipe de projetos da L_RU, o custo de desenvolvimento do UC_9 não compensa o seu impacto positivo sobre a receita gerada pelos UC_3 e UC_4.

5 DISCUSSÃO E CONCLUSÃO

5.1 Discussão

No início desta dissertação, nos propusemos a mostrar como a identificação precoce de NMMFs (MMFs não essenciais) e NAEs (elementos arquiteturais não essenciais) pode afetar o valor final de projetos de *software* e que o processo de criação de valor de NMMFs e NAEs pode ser bem diferente do processo de criação de valor de MMFs e AEs. A seguir, discutimos as principais questões que nortearam este trabalho e suas implicações para diferentes dimensões de negócio e do gerenciamento de projetos de *software*.

5.1.1 Como o valor do sistema de controle de empréstimo consignado da L_RU foi afetado pela presença de NMMFs, NAEs e relações de precedência flexíveis?

A Tabela 5.1 compara os valores das diferentes alternativas consideradas pela equipe de projetos da L_RU para o desenvolvimento do sistema de controle de empréstimo consignado. Como resultado, é seguro afirmar que a introdução de relações de

precedência flexíveis e a presença de NMMFs e NAEs entre as unidades de software essenciais pode aumentar o valor de negócio de projetos de software.

Tabela 5.1: Alternativas consideradas pela $L_R U$ para o desenvolvimento do sistema de controle de empréstimo consignado

#	Referência		Flexibilidade	NPV da Escolha Lógica (US$ 1.000)
	Modelo do Caso de Uso	Diagrama de Precedência		
1	Figura 2.1	Figura 2.2	Nenhuma	US$17.564
2	Figura 2.1	Figura 3.1	A relação de precedência de dependência entre os UC_1, UC_2 e UC_5 é flexibilizada	US$18.460
3	Figura 4.1	Figura 4.2	A relação de precedência de dependência entre os UC_1, UC_2 e UC_5 é flexibilizada e os NMMFs e NAEs são apresentados	US$19.651

5.1.2 Como os resultados apresentados nesta dissertação ajudam as organizações a tirarem um maior proveito de seus projetos de *software*?

Apesar da considerável quantidade de pesquisa disponível na literatura sobre o impacto da flexibilidade na avaliação, seleção e no gerenciamento de projetos de *software* (BENAROCH; SHAH; JEFFERY, 2006; WU; ONG, 2008; FICHMAN; KEIL; TIWANA, 2005), surpreendentemente, a introdução de flexibilidade através de unidades de *software* não essenciais e relações de precedência flexíveis é pouco explorada. O objetivo desta dissertação é preencher essa lacuna.

Nas últimas décadas, indústria e academia testemunharam a tecnologia da informação passar do papel de uma simples ferramenta capaz de acelerar processos para se tornar peça fundamental na estratégia de negócios de organizações de todos os tipos

e tamanhos. Os adventos que favoreceram esta mudança de papel não se restrigem à introdução de tecnologias mais novas e mais poderosas. Envolve a globalização da economia mundial e o número crescente de consumidores altamente exigentes, que se acostumaram a receber melhores produtos e serviços por preços reduzidos. Neste ambiente competitivo, onde há uma quantidade enorme de fornecedores para cada produto e serviço, pouquíssimas empresas estão numa posição que lhes permitam ser tecnologicamente ineficientes.

A discussão apresentada nesta dissertação ajuda as organizações a tirar o máximo proveito de seus projetos de software e, como resultado, aumentar o nível de eficiência no uso da tencologia da informação. Em outras palavras, tendo consciência do valor potencial de NMMFs e relações de dependência flexíveis, as empresas podem seguir as idéias de Denne e Cleland-Huang (DENNE; CLELAND-HUANG, 2005) e desenvolver projetos maiores e mais complexos a partir de um investimento relativamente menor.

5.1.3 Quais são as principais limitações do método proposto nesta dissertação?

Este trabalho de dissertação se baseia nos seguintes pressupostos:

- *Equipe de desenvolvimento* - que apenas uma equipe de desenvolvimento está disponível para implementar as diversas unidades de *software* que fazem parte de um projeto. Portanto somente uma unidade de *software* pode ser desenvolvida em cada período;
- *Recursos* - que todos os recursos que uma unidade necessita para o seu desenvolvimento (sejam eles financeiros ou não) estão disponíveis quando as unidades de *software* começam a ser desenvolvidas;

- *Unidades Essenciais* - que todas as unidades essenciais serão necessariamente desenvolvidas;

- *Delay* - nenhum *delay* (espaço) de tempo decorre entre o desenvolvimento de uma unidade de *software* e o desenvolvimento da unidade seguinte;

- *Estimativas* - os valores referentes as estimativas de duração e custo do esforço de desenvolvimento das diversas unidades de *software* são fixos, apesar destes valores só virem a ser conhecidos com precisão no futuro.

5.1.4 As expressões "MMF e AEs não essenciais" são adequadas ao escopo desta dissertação?

Quando os objetivos de um projeto não podem ser expressos ou não serão expressas em termos financeiros, então as expressões "MMF e AEs não essenciais" descrevem com clareza as unidades que podem ser opcionalmente implementadas sem que isso diminua as chances do projeto alcançar seus objetivos.

Entretanto, se o projeto em questão tem objetivos financeiros bem definidos e estes objetivos envolvem a maximização do retorno sobre os investimentos realizados, então o termo "unidade não essencial" deixa de caracterizar claramente unidades que podem ser retiradas do projeto sem que seus objetivos sejam afetados, já que a presença destas unidades pode refletir positivamente sobre o retorno dos investimentos realizados, conforme mostrado ao longo da dissertação.

Neste caso, unidades não essenciais são aqueles que ao serem implementados não maximizam o retorno dos investimentos. Segundo (DENNE; CLELAND-HUANG, 2004b) trata-se na verdade de unidades que não trazem valor para o negócio, ou seja: não propiciam diferenciação competitiva, não provêem geração de lucro, não per-

mitem a redução de custos, não projetam a marca da organização e não influenciam no aumento da fidelidade dos clientes. Portanto, em se tratando de maximização do retorno financeiros as unidades de *software* podem ser divididas entre financeiramente essenciais e financeiramente não essenciais. Sendo que não existe razão lógica para que essas últimas sejam implementadas.

5.2 Trabalhos Futuros

Os conhecimentos produzidos a partir deste trabalho representam o ponto de partida na investigação de novos benefícios que as relações de precedência flexíveis e unidades de *software* não essenciais podem criar.

Neste sentido, além de se desenvolverem esforços para superar as limitações descritas na Seção 5.1.3, com o objetivo de expandir a usabilidade do método apresentado nesta dissertação, destacamos as seguintes oportunidades para futuros esforços de pesquisa:

- *Elicitação de MMFs e AEs* - apesar de tudo que já foi escrito na literatura sobre *minimum marketable features*, a identificação de MMFs e AEs ainda carece de método (DENNE; CLELAND-HUANG, 2005).
- *O alinhamento entre as diferentes unidades de software e a estratégia organizacional* - apesar do alinhamento e esforços com a estratégia organizacional serem uma das formas mais eficientes de obter retorno dos investimentos realizados por uma organização, até o momento nada foi dito na literatura sobre como considerar a possível contribuição de MMFs e AEs (assim como de suas contrapartes não essenciais) para a implementação da estratégia organizacional no escalonamento do desenvolvimento das diversas unidades de softwarte (AK-

PAN, 2007).

- *Carteira de Projetos* - NMMFs e NAEs não ocorrem somente em projetos de *software* isolados, elas, na verdade, ocorrem dentro de carteiras de projetos, podendo, muitas vezes serem comuns a vários projetos. É possível também, que unidades não essenciais em um projeto sejam essenciais para o desenvolvimento de outros projetos. Neste caso, um método que abrangesse o estudo de unidades não essenciais e suas relações dentro de um portfolio de projetos viria a aumentar em muito a abrangência do método apresentado nesta dissertação.

- *Diferentes formas de flexibilização* - no bojo desta dissertação consideramos apenas a possibilidade de deixar de implementar ou não certas unidades de software e que as relações de dependências entre diferentes unidades poderia ser flexível. Entretanto, no mundo real, as organizações passam por situações nas quais a forma mais lucrativa de se conduzir um projeto é a de adiar a implementação de certas unidades por algum tempo até que os recursos disponíveis ou as condições de mercado se modifiquem. A adição deste tipo de flexibilidade pode mudar consideravelmente o valor de projetos de *software*, principalmente em situações de grande incerteza (FAVARO, 2002).

5.2.1 Conclusões

Nesta dissertação apresentamos um método capaz de avaliar o impacto de unidades de *softwares* não essenciais, assim como relações de dependência flexíveis, no valor total de um projeto de *software*.

No decorrer do texto, mostramos que a forma como unidades de *software* não essenciais, na forma de MMFs e elementos arquiteturais não essenciais, criam valor para um projeto pode ser bem diferente da forma de criação de valor das unidades essenciais.

Além disso, a implementação de unidades não essenciais, quando bem planejada, pode aumentar significativamente o valor de um projeto ao mesmo tempo em que diminui a necessidade de capital e o risco.

Uma ferramenta automatizada foi desenvolvida para auxiliar o gerente de projeto obter o melhor sequenciamento possível para a implementação das unidades de *software* em projetos que contenham unidades essenciais e não essenciais e é apresentada no Apêndice.

APÊNDICE

Nesta seção apresentaremos a ferramenta que implementa o método BnBFlex. A tecnologia utilzada para desenvolvimento foi Java 1.6 para Web (JEE) utilizando o servidor de aplicações GlassFish.

5.3 Propriedades do Projeto

A tela incial solicita as propriedades do projeto. Nesta tela são configuradas algumas propriedades do projeto.

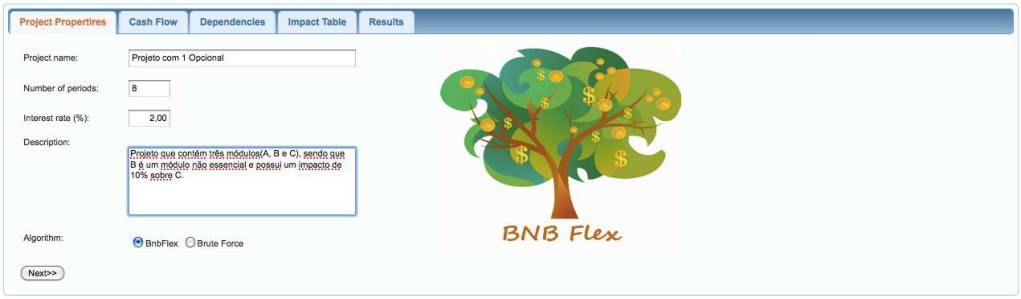

Figura 5.1: Propriedades do projeto.

As informações solicitadas são:

- Project name - Nome do projeto que será simulado.
- Number of periods - Tempo de vida ou janela de oportunidade do projeto.
- Interest rate - Taxa de desconto utilizada no projeto.
- Description - Uma breve descrição do projeto.
- Algorithm - Algoritmo utilizado para a simulação.

5.4 Fluxo de Caixa

A segunda tela permite que as unidades de *software* sejam informadas com os seus respectivos fluxos de caixa. Além disso, ainda é possível informar se o módulo é ou não essencial para o projeto.

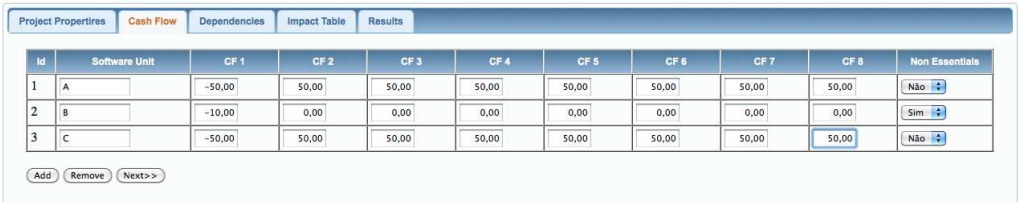

Figura 5.2: Fluxo de Caixa do projeto.

Os botões de "Adicionar"e "Remover"permitem que sejam adicionados ou removidos dinamicamente novas unidades de *software* com seus respectivos fluxos de caixa.

5.5 Relações de dependência

A terceira tela solicita que a relação de dependência seja informada para cada uma das unidades de *software* do projeto.

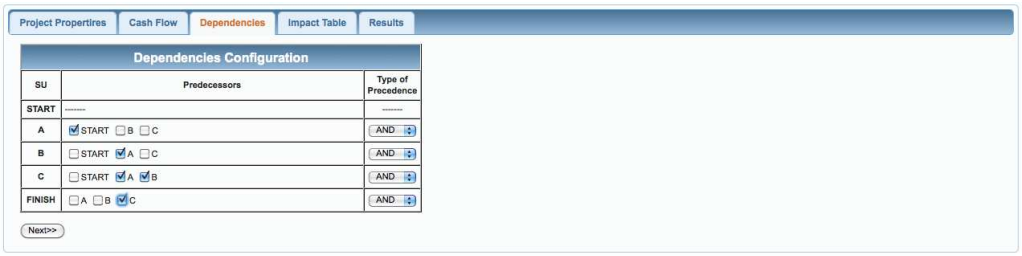

Figura 5.3: Relação de Dependência do projeto.

5.6 Tabela de Impacto

A quarta tela solicita que sejam informados os percentuais de impacto exercidos pelas unidades de *software* não essenciais sobre as unidades de *software* essenciais.

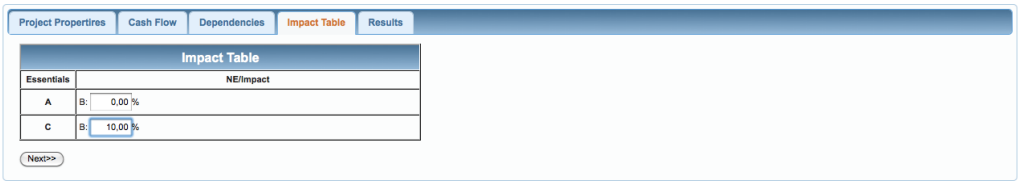

Figura 5.4: Tabela de impacto das unidades de *software* não essenciais sobre as unidades de *software* essenciais do projeto.

5.7 Resultado Final

A quinta e última tela lista todas as informações passadas anteriormente, calcula os VPL's de cada unidade de *software* e por fim constrói a árvore do projeto utilizando o método BnBFlex para dar a opção que maximiza o VPL do projeto.

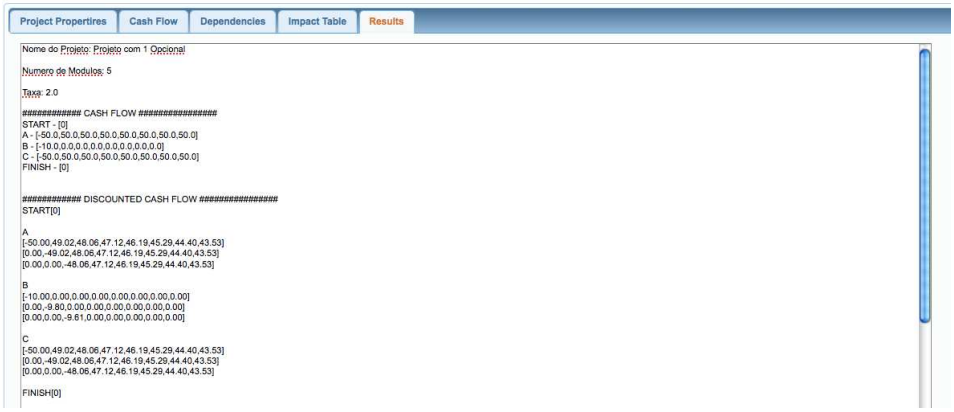

Figura 5.5: Resultado do método - Parte 1.

Figura 5.6: Resultado do método - Parte 2.

REFERÊNCIAS

AKPAN, E. O. **Strategic Alignment**: the business imperative for leading organizations. 3^{rd}.ed. Mustang, OK, USA: Tate Publishing & Enterprises, 2007.

ALENCAR, A. J.; SCHMITZ, E. A.; ABREU Ênio Pires de. Maximizing the Business Value of Software Projects: a branch & bound approach. In: INTERNATIONAL CONFERENCE ON ENTERPRISE INFORMATION SYSTEMS., 10., 2008, Barcelona, Spain. **Anais...** INSTICC, 2008. v.ISAS-2, p.162–169.

BENAROCH, M.; SHAH, S.; JEFFERY, M. On the Valuation of Multistage Information Technology Investments Embedding Nested Real Options. **Journal of Management Information Systems**, [S.l.], v.23, n.1, p.239–261, Summer 2006.

BENSON, R. J.; BUGNITZ, T.; WALTON, B. **From Business Strategy to IT Action**: right decisions for a better bottom line. Hoboken, NJ, USA: Wiley, 2004.

BITTNER, K.; SPENCE, I. **Use Case Modeling**. Old Tappan, NJ, USA: Addison-Wesley, 2002.

BOOCH, G.; RUMBAUGH, J.; JACOBSON, I. **The Unified Modeling Language User Guide**. 2^{nd}.ed. Old Tappan, NJ, USA: Addison-Wesley, 2005.

BURNS, A. C.; BUSH, R. F. **Marketing Research**. 6^{nd}.ed. Saddle River, NJ, USA: Prentice Hall, 2009.

CUSUMANO, M. A. **The Business of Software**: what every manager, programmer, and entrepreneur must know to thrive and survive in good times and bad. 40 Main St, Suite 301, Florence, MA, USA: Free Press, 2004.

DEKLEVA, S. Justifying Investments in IT. **Journal of Information Technology Management**, [S.l.], v.XVI, n.3, p.1–8, 2005.

DEMEULEMEESTER, E.; HERROELEN, W. A Branch-and-Bound Procedure for the Multiple Resource-Constrained Project Scheduling Problem. **Management Sciene**, [S.l.], v.38, n.12, p.1803–1818, 1992.

DEMEULEMEESTER, E. L.; HERROELEN, W. S. **Project Scheduling**: a research handbook. New York, NY, USA: New York, NY, 2002.

DENNE, M.; CLELAND-HUANG, J. The Incremental Funding Method: data-driven software development. **IEEE Software**, Institute of Electrical and Electronics Engineers, v.21, n.3, p.39– 47, 2004.

DENNE, M.; CLELAND-HUANG, J. **Software by Numbers**. Redwood Shores, CA, USA: Sun Microsystems Press, 2004.

DENNE, M.; CLELAND-HUANG, J. Financially Informed Requirements Prioritization. In: INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING (ICSE), 27., 2005, St. Louis, Missouri, USA. **Proceedings...** ACM Press, 2005. p.710–711.

DENNEY, R. **Succeeding with Use Cases**: working smart to deliver quality. Old Tappan, NJ, USA: Addison-Wesley, 2005.

FABOZZI, F. J.; DAVIS, H. A.; CHOUDHRY, M. **Introduction to Structured Finance**. San Francisco, USA: John Wiley, 2006.

SUCCI, G.; MARCHESI, M.; WILLIAMS, L.; WELLS, J. D. (Ed.). **Extreme Programming Perspectives**. Boston, MA, USA: Addison-Wesley Longman Publishing Co, 2002. p.503–552.

FICHMAN, R. G.; KEIL, M.; TIWANA, A. Beyond Valuation: real options thinking in IT project management. **California Management Review**, [S.l.], v.47, n.2, p.74–100, 2005.

GREMBERGEN, W. V. **Information Technology Evaluation Methods and Management**. Chocolate Ave., Hershey, PA, USA: Idea Group Publishing, 2001.

GROSS, J. L.; YELLEN, J. **Graph Theory and Its Applications**. 2^{nd}.ed. Boca Raton, FL, USA: Chapman & Hall and CRC, 2005.

HIBBS, C.; JEWETT, S.; SULLIVAN, M. **The Art of Lean Software Development**: a practical and incremental approach. Sebastopol, CA, USA: O'Reilly Media, 2009.

HILLIER, F. S.; LIEBERMAN, G. J. **Introduction to operations research**. 7^{th}.ed. New York, NY, USA: McGraw-Hill, 2001.

HUBBARD, D. W. **How to Measure Anything**: finding the value of "intangibles" in business. Hoboken, NJ, USA: John Wiley, 2007.

LIBERTI, L. **Optimization and Optimal Control**. Hackensack, NJ, USA: World Scientific, 2003. p.165–174. (Computers and Operations Research, v.1).

MILIS, K.; MERCKEN, R. The use of the balanced scorecard for the evaluation of Information and Communication Technology projects. **International Journal of Project Management**, [S.l.], v.22, n.2, p.87–97, February 2004.

PETERSON, C. L. Usury Law, Payday Loans, and Statutory Sleight of Hand: salience distortion of american credit pricing limits. **Minnesota Law Review**, [S.l.], v.92, n.4, April 2008.

POHL, K.; BöCKLE, G.; LINDEN, F. J. van der. **Software Product Line Engineering**: foundations, principles and techniques. New York, NY, USA: Springer, 2010.

PRIYA KURIEN, W. R.; PURUSHOTTAM, V. S. The case for re-examining IT effectiveness. **Journal of Business Strategy**, [S.l.], v.25, n.2, p.29–36, 2004.

ROSACKER, K. M.; OLSON, D. L. An empirical assessment of IT project selection and evaluation methods in state government. **Project Management Journal**, [S.l.], v.29, n.1, p.49–58, February 2008.

SIDDIQI, N. **Credit Risk Scorecards**: developing and implementing intelligent credit scoring. San Francisco, USA: John Wiley, 2005.

WU, L.-C.; ONG, C.-S. Management of information technology investment: a framework based on a real options and mean variance theory perspective. **Technovation**, [S.l.], v.28, n.3, p.122–134, 2008.

WU, S. D.; SHI, C.; GURBAXANI, V. Investigating the Risk-Return Relationship of Information Technology Investment: firm-level empirical analysis. **Management science**, [S.l.], v.53, n.12, p.1829–1842, 2007.

ZHANG, W. Depth-first branch-and-bound versus local search: a case study. In: IN PROC. 17TH NATIONAL CONF. ON ARTIFICIAL INTELLIGENCE (AAAI-2000, 2000. **Anais...** [S.l.: s.n.], 2000. p.930–935.