Universidade Federal Rural de Pernambuco
Departamento de Estatística e Informática

Uma Proposta de Modelagem para a Alocação de Requisitos em Times Ágeis Utilizando Programação Inteira Mista

Victor José Aguiar Teixeira de Melo França

Recife

Dezembro de 2015

Victor José Aguiar Teixeira de Melo França

Uma Proposta de Modelagem para a Alocação de Requisitos em Times Ágeis Utilizando Programação Inteira Mista

Orientadora: Silvana Bocanegra
Co-orientadora: Ana Cristina Rouiller

Monografia apresentada ao Curso Bacharelado em Sistemas de Informação da Universidade Federal Rural de Pernambuco, como requisito parcial para obtenção do título de Bacharel em Sistemas de Informação.

Recife

Dezembro de 2015

Ao meu avô.
À minha família.
Aos meus amigos.
À equipe SWQuality.
À todos que fazem a minha vida
valer a pena.

Agradecimentos

Agradeço primeiramente aos meus pais, irmãos e avós pelo apoio durante o curso e por aguentar os estresses que eu passava e contagiava todos, sempre me incentivando a seguir em frente.

Agradeço à professora Silvana Bocanegra, por todo empenho ao me orientar na conclusão deste trabalho, mesmo que às vezes eu tenha parecido meio perdido, era sua atenção e disponibilidade que me ajudavam a seguir em frente, sempre mostrando o melhor caminho. Foi um prazer tê-la como professora da primeira e última disciplina do curso.

Agradeço à professora Ana Rouiller, pela co-orientação, pelas oportunidades, por acreditar que eu sou capaz e ver em mim um potencial que nem eu sabia que tinha. Por sempre me incentivar a me apoiar profissional e pessoalmente, me trazendo sempre ao mundo real sem nunca deixar de sonhar em mudar o mundo.

E claro, à Jones, Giordano, Ceça, Thiago, Gabriel, Assad e todos os professores por quais passei, pois hoje tenho um pouquinho de cada um plantado em mim.

Obrigado à Equipe SWQuality, principalmente Renata, Sandro e Heron, que acreditaram no meu trabalho e me proporcionaram experiências incríveis que foram essenciais para a conclusão desta monografia. Também agradeço à Mariana Moura, amiga e companheira de trabalho, pela força, apoio e parceria durante o curso e principalmente neste momento de finalização, também obrigado Eça, Erick, Flaviano, Thays, Emeson, Delando e todos que de alguma forma me incentivaram a nunca desistir, ou mesmo àqueles que só perguntavam como estava meu trabalho.

Um agradecimento especial a Matheus Xavier, companheiro e amigo desde o primeiro período, me apoiando nas ideias mais loucas e sempre dando força para os meus objetivos.

Obrigado a DJVLM por sempre estarem comigo desde àquele momento que eu nem sabia o que fazer da vida, obrigado por serem meu escape. Valeu amigos do Intercâmbio, Daniela Pra, Émillie Dias e todos que me proporcionaram momentos incríveis. Obrigado Rafael Mendonça, por compartilhar tantos momentos e sonhos comigo, obrigado pela força.

Enfim, agradeço à todos os meus amigos, novos ou velhos que de alguma forma foram parte de mim durante todo o tempo da graduação. Obrigado por existirem na minha vida.

Resumo

Na Engenharia de *Software* identificamos problemas onde é necessário encontrar equilíbrio entre objetivos concorrentes e conflitantes. Estes problemas envolvem decisões sob um conjunto de inúmeras opções, e podem ser modelados como problemas de otimização. O NRP - *Next Release Problem*, é um dos problemas de Otimização em Engenharia de *Software*. Dado um conjunto de Clientes solicitando um conjunto de Requisitos cada, o NRP consiste em selecionar quais comporão o próximo *release*, respeitando o custo disponível e maximizando a satisfação dos clientes, considerando sua importância para a organização. Este trabalho tem como objetivo mapear características de empresas de desenvolvimento e manutenção de software que utilizam Scrum e realizar adaptações no modelo proposto inicialmente para adequá-lo a estas empresas. Foram usadas três funções objetivos distintas, a primeira com o objetivo de maximizar o percentual de satisfação dos clientes, tendo como parâmetro o *Business Value* dos requisitos, a segunda minimiza o tempo de desenvolvimento e a terceira minimiza o custo. As restrições envolvem capacidade da equipe, dependências entre os requisitos, entre outras. Como estudo de caso, foram selecionadas duas empresas de manutenção e evolução de software do estado da Bahia. Com os resultados obtidos, foi possível adequar o NRP à realidade destas organizações, focando no planejamento da próxima iteração de acordo com os objetivos estratégicos da empresa em atender seus clientes da melhor forma.

Palavras-chave: Engenharia de Software, Engenharia de Requisitos, Otimização, AIMMS, Scrum.

Abstract

In Software Engineering is possible identify problems where we need to find balance between competing and conflicting goals. These problems involve decisions under a set of many options and can be modeled as an optimization problem. The NRP - Next Release Problem, is one of optimization problems in software engineering. Given a set of customers requesting a set of requirements each, the NRP aims to select which will make the next release, respecting the available cost and maximizing customer satisfaction, considering its importance to the organization.This work aims to map characteristics of companies' development and maintenance of software using Scrum and make adaptations in the model proposed initially to adapt it to these companies. Three different objective functions were used, the first in order to maximize customer satisfaction percentage, having as parameter the Business Value of requirements, the second objective function minimizes development time and the third minimizes the cost. The restrictions involve team's ability, dependencies between requirements, etc. As a case study, two companies were selected maintenance and software development, from Bahia. With the results, it was possible to adapt the NRP to the reality of these organizations, focusing on the next iteration of planning in accordance with the strategic objectives of company serve its customers in the best way.

Keywords: *Software Engineering, Requirements Engineering, Optimization, AIMMS, Scrum.*

Sumário

Lista de Tabelas

Lista de Figuras

Lista de Algoritmos

Capítulo 1

Introdução

1.1 Visão Geral

A Engenharia de Software é uma disciplina de engenharia relacionada a todos aspectos que envolvem a produção de um software, desde as etapas iniciais de especificação do sistema até a etapa de manutenção do produto já em utilização [19].

Atualmente, uma forte tendência do mercado é a adoção de metodologias que tenham como objetivo que as necessidades do cliente sejam atendidas de forma rápida, recebendo *feedbacks* sobre os produtos entregues durante todo o processo do desenvolvimento. Esse modelo, também chamado de Modelo Incremental, faz uso de pequenas entregas com conjuntos de funcionalidades desenvolvidas, que são chamadas de *releases*. Apesar de ter se tornado popular atualmente por conta das Metodologias Ágeis[1], essa técnica de desenvolvimento já é utilizada desde a década de 1970 [14].

O Modelo Incremental é basicamente um processo onde a especificação, o projeto, a implementação e o teste são realizados em fases intercaladas. Assim, uma série de *releases* são definidas e a cada entrega, um incremento do software deverá ser disponibilizado. A Figura 1.1[2] ilustra os incrementos do modelo. Ao final dos incrementos necessários, haverá a entrega do produto como um todo.

[1] http://www.manifestoagil.com.br/

[2] Fonte: https://infomarcio.wordpress.com/2010/10/30/

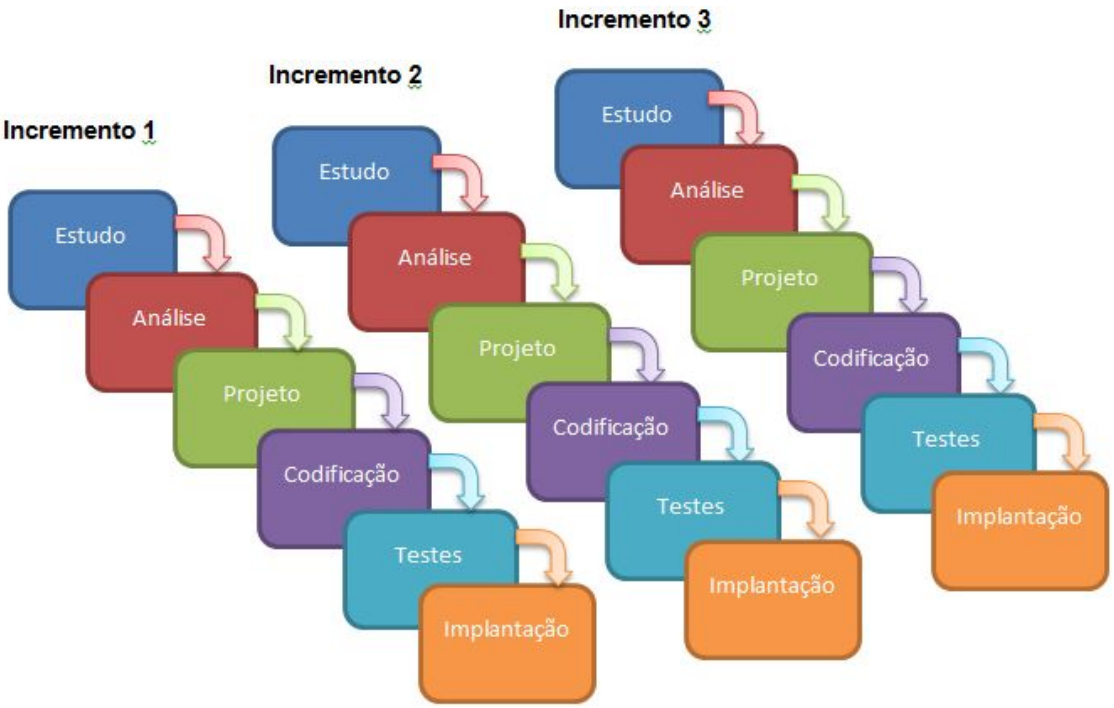

Figura 1.1: Modelo Incremental.

Ao final de cada ciclo uma nova versão do sistema é entregue, o que aumenta a satisfação do cliente no que se refere ao prazo. Com esta versão é possível que o cliente forneça *feedbacks* frequentes à equipe de desenvolvimento e consequentemente isso reduz a possibilidade de insatisfação em relação às funcionalidades desenvolvidas e entregues.

O método iterativo traz muitas vantagens para o desenvolvimento de grandes sistemas de software, mas também incorpora uma maior complexidade no gerenciamento das atividades por parte dos desenvolvedores e líderes de projetos. Ao final de cada iteração, as novas funcionalidades implementadas devem ser acopladas ao subproduto que o cliente está utilizando, gerando um novo subproduto.

Como o sistema será entregue ao longo de vários *releases*, torna-se claro que, no início de cada ciclo, deve ser decidido o que será incluído nesta entrega. Contudo, definir quais são os requisitos que farão parte do próximo *release* é uma tarefa difícil, podendo ser considerada um problema NP-Difícil, ou seja, não é conhecido nenhum algoritmo de complexidade polinomial para soluciona-lo, principalmente devido à possibilidade de existirem vários clientes envolvidos no projeto e do orçamento limitado previsto para o *release*. É neste contexto que surge o Problema do Próximo *Release* (*Next Release Problem - NRP*) [3] que tem como objetivo selecionar os requisitos que farão parte do próximo *release*, sem extrapolar o orçamento

máximo definido para ele e que garanta maximizar a satisfação dos clientes.

1.2 Objetivos do Trabalho

Este trabalho tem como principais objetivos:

- Entender o problema da próxima release sob a perspectiva de um problema de otimização;
- Mapear as especificidades de empresas que utilizam *scrum* como metodologia de desenvolvimento de Software;
- Coletar informações nestas empresas de software para propor o desenvolvimento das novas restrições que serão acrescentados ao modelo básico do problema da próxima release;
- Propor modelos, usando novas funções objetivo e restrições, que representem melhor a realidade destas empresas;
- Utilizar a ferramenta AIMMS para modelagem matemática dos problemas e em seguida resolver tais modelos usando dados reais de empresas de desenvolvimento de software que utilizam *scrum*;
- Propor uma interface que auxilie o gerente de projetos / produto ao definir o subconjunto de requisitos para o próximo *release*.

1.3 Justificativa

Um problema bastante comum para as empresas que trabalham com evolução e manutenção de produtos de *software* desenvolvidos para vários clientes é priorizar o que será implementado na próxima *release*. Isto porque diversos fatores podem influenciar nessa escolha, onde alguns deles são fatores humanos, tornando-a bastante complicada quando tratá-se de avaliar manualmente cada um deles e muitas vezes esta atividade é negligenciada pelas organizações, já que é despendido um tempo considerável para que uma boa solução seja encontrada.

No caso de empresas que utilizam o *Scrum* como processo de desenvolvimento, alguns pontos vindouros da metodologia são identificados e dificultam ainda mais esta escolha, como por exemplo:

- Time-box da *sprint* como prazo de entrega de um conjunto de requisitos;
- Depências entre requisitos;
- Vários times de desenvolvimento para alocação dos requisitos;
- Estimativa do requisito é dada em pontos;
- Valor de negócio do requisito não é levado em consideração no modelo tradicional.

Portando, há a necessidade de inserir algumas adequações ao modelo básico para representar a realidade das empresas.

1.4 Organização do Documento

- No Capítulo 2 - O Problema do Próximo *Release*, é descrita a formulação tradicional do Problema da Próxima *Release* e é apresentada uma revisão da literatura sobre o assunto
- No Capítulo 3 - Proposta para Empresas de Software que Utilizam Scrum, são apresentadas especificadades do processo de desenvolvimento e manutenção de *software* de algumas empresas que utilizam a metodologia. Em seguida, são apresentadas as adequações propostas para o modelo.
- No Capítulo 4 - Estudo de Caso, é apresentada a ferramenta AIMMS para modelagem e solução do problema e são realizadas algumas simulações do modelo proposto com dados de duas empresas de manutenção e evolução de software. São realizadas análises com a variação do tamanho da *sprint* e seu reflexo no custo e satisfação.
- No Capítulo 5 - Conclusão, são descritas as conclusões e trabalhos futuros. Finalizando encontram-se as Referências Bibliográficas e os Anexos. Nos anexos estão descritos os modelos na linguagem de modelagem do AIMMS.

Capítulo 2

O Problema do Próximo Release

A Engenharia de Software Baseada em Busca, ou *Search Based Software Engineering* (SBSE), é o termo criado por [11] para denominar a aplicação de técnicas de busca em Engenharia de Software. É possível encontrar alguns trabalhos anteriores[3] na área de Pesquisa Operacional na Engenharia de Software, principalmente na área de testes, mas a partir deste artigo [11] é que a comunidade da Engenharia de Software realmente começou a perceber a utilidade dos algoritmos de otimização em seus problemas.

Para aplicar uma técnica de otimização a um problema de Engenharia de Software, se faz necessária a adaptação, ou modelagem, do problema. Um modelo pode ser visto como a representação aproximada de algum problema real utilizando uma determinada linguagem (matemática, lógica, geográfica, física e etc). Modelos que utilizam a linguagem matemática são denominados modelos matemáticos e podem ser representados por um conjunto de equações e/ou expressões. Um modelo de otimização é um modelo matemático que é obtido quando houver a necessidade de maximizar ou minimizar uma função objetivo sujeita a um conjunto de restrições (equações ou inequações) que limitam as variáveis dessa função. A função objetivo, ou função de fitness, é responsável por medir a qualidade de uma possível solução. Basicamente atribui um valor numérico para determinada solução, fazendo com que várias soluções candidatas possam ser comparadas entre si. A modelagem do problema é um dos fatores mais importante na SBSE, sendo considerado o mais importante por vários autores [12]. Com a SBSE tomando forma e se concretizando como uma linha de pesquisa dentro da Engenharia de Software, alguns problemas já são considerados "clássicos" da área.

[3] http://crestweb.cs.ucl.ac.uk/resources/sbse_repository/

Dentre os quais podem ser citados a seleção e priorização de casos de teste [8], a otimização de estimativas de custo [12] e problemas relacionados à engenharia de requisitos, como o Problema do Próximo Release [3], que é o foco deste trabalho.

2.1 Formulação Tradicional

O problema é originado da dificuldade enfrentada pelas equipes de desenvolvimento para selecionar o conjunto de requisitos que irá compor a *release* seguinte, levando em consideração que estes requisitos são demandados por diferentes clientes, com graus de importância variados para o negócio. Cada requisito possui um custo de desenvolvimento e o conjunto selecionado deve satisfazer uma limitação de custos pré-definida. Logo, o problema é selecionar um conjunto ideal de requisitos que satisfaça os clientes da melhor maneira e respeite a limitação de custos da organização.

Este problema foi denominado como *Next Release Problem* (NRP) ou Problema do Próximo Release. A formulação originial do NRP foi apresentada como um problema de otimização monoobjetivo (uma única função objetivo): maximizar a satisfação dos patrocinadores do projeto de software. Neste sentido, o esforço para desenvolvimento dos requisitos desejados para a próxima *release* possui um limite superior e é definido pelos patrocinadores de acordo com os recursos disponíveis na organização. Com isto, o objetivo é encontrar o subconjunto de requisitos que melhor atenda às expectativas dos patrocinadores. A seguir, a formalização do NRP de acordo com o trabalho de [3].

Seja R o conjunto de todos os requisitos de software a serem desenvolvidos por uma equipe de desenvolvimento. Cada cliente, i, terá um conjunto de requisitos, $R_i \subseteq R$ e um valor $v_i \in \mathbb{Z}^+$ que mensura a importância do cliente para a organização.

Cada $r \in R$ possui um custo associado, $c_r \in \mathbb{Z}^+$, que é o custo de implementação do requisito r. E a empresa possui uma limitação de custo de implementação L, que é o limite de capacidade de desenvolvimento da equipe.

Associado ao conjunto R está um grafo $G = (R, E)$, direcionado e acíclico, onde R é o conjunto de vértices, E é o conjunto de arestas e $(r, r') \in E(G)$ se, e somente se, r é um pré-requisito de r'. G não é apenas acíclico, ele também é transitivo, desde que $(r, r') \in E(G)$ & $(r', r'') \in E(G) \rightarrow (r, r'') \in E(G)$.

Se a empresa decidir satisfazer todos requisitos do cliente i, ela não deverá desenvolver apenas R_i, mas também

$$parents(R_i) = \{r \in R | (r, r') \in E(G), r' \in R\}.$$

Assumindo que a empresa possui n clientes, o desafio é encontrar um subconjunto de clientes, $S \subseteq \{1, 2, ..., n\}$ cujos requisitos serão satisfeitos.

Este problema pode ser formulado como um problema de otimização, no qual temos:

1. **Variáveis de decisão:**

   $$y_i = \begin{cases} 1 & \text{se o cliente i é atendido} \\ 0 & \text{caso contrário} \end{cases}, \forall i \in S.$$

   $$x_j = \begin{cases} 1 & \text{se o requisito j é implementado} \\ 0 & \text{caso contrário} \end{cases}, \forall j \in R.$$

2. **Função objetivo:**

   $$max \sum_{i \in S} v_i y_i,$$

   onde v_i representa a importância do cliente i para a empresa.

3. **Restrições:**

   3.1. *Limitação de custo de implementação* - Diversos fatores podem influenciar esta limitação dentro de uma empresa. Alguns exemplos são prazos e recursos limitados. Dessa forma, temos

   $$\sum_{j \in R} c_j x_j \leq L,$$

   onde c_j é o custo de implementação do requisito r_j e L é a limitação de custo de implementação da empresa.

   3.2. *Dependência entre requisitos* - A dependência entre os requisitos pode ser representada pela seguinte equação:

$$x_r \geq x_{r'}, \forall (r, r') \in E(G),$$

isso significa que um dado requisito r' só pode ser implementado se seu pré-requisito r também for.

3.3. *Garantia que se o cliente é atendido, todos os requisitos solicitados por ele devem ser implementados.*

$$x_j \geq y_i, \forall i, j, \text{ tal que } r_j \text{ é requisito do cliente } i.$$

Sintetizando, tem-se:

$$max \sum_{i \in S} v_i y_i,$$

$$\text{sujeito a } \begin{cases} \sum_{j \in R} c_j x_j \leq L & \\ x_r \geq x_{r'}, & \forall (r, r') \in E(G). \\ x_j \geq y_i, & \forall i,j, \text{ tal que } r_j \text{ é requisito do cliente } i. \end{cases}$$

onde v_i representa o valor do cliente i, c_j representa o custo de implementação do requisito r_j e L é a limitação de custo de implementação da equipe de desenvolvimento.

Um exemplo simples para facilitar o entendimento pode ser observado a seguir:

$R = \{r_1, r_2, ..., r_7\}$,
$c_1 = 10, c_2 = 6, c_3 = 7, c_4 = 1, c_5 = 4, c_6 = 6, c_7 = 1$,
$R_1 = \{r_6\}, R_2 = \{r_6, r_7\}, R_3 = \{r_5\}$,
$E = \{(1,3), (1,4), (1,6), (1,7), (2,5), (2,7), (3,6), (4,7), (5,7)\}$,
$\hat{R_1} = \{r_1, r_3, r_6\}, \hat{R_2} = \{r_1, r_2, r_3, r_4, r_5, r_6, r_7\}, \hat{R_3} = \{r_2, r_5\}$,
$v_1 = 50, v_2 = 60, v_3 = 70$.

Uma representação gráfica da estrutura deste problema é apresentada na Figura 2.1. Expandindo a formulação do problema temos que nosso objetivo é maximizar

$50y_1 + 60y_2 + 70y_3$,

Sujeito a

- Limitação de custo de implementação:

  $10x_1 + 6x_2 + 7x_3 + 1x_4 + 4x_5 + 6x_6 + 1x_7 \leq L$.

- Dependência entre requisitos:

  $x_1 \geq x_3$,
  $x_1 \geq x_4$,
  $x_2 \geq x_5$,
  $x_3 \geq x_6$,
  $x_4 \geq x_7$,
  $x_5 \geq x_7$,
  $x_j \in \{0,1\}, y_i \in \{0,1\}$.

- Garantia que se o cliente é atendido, todos os requisitos solicitados por ele devem ser implementados

  $x_6 \geq y_1$,
  $x_6 \geq y_2$,
  $x_7 \geq y_2$,
  $x_5 \geq y_3$,
  $x_j \in \{0,1\}, y_i \in \{0,1\}$.

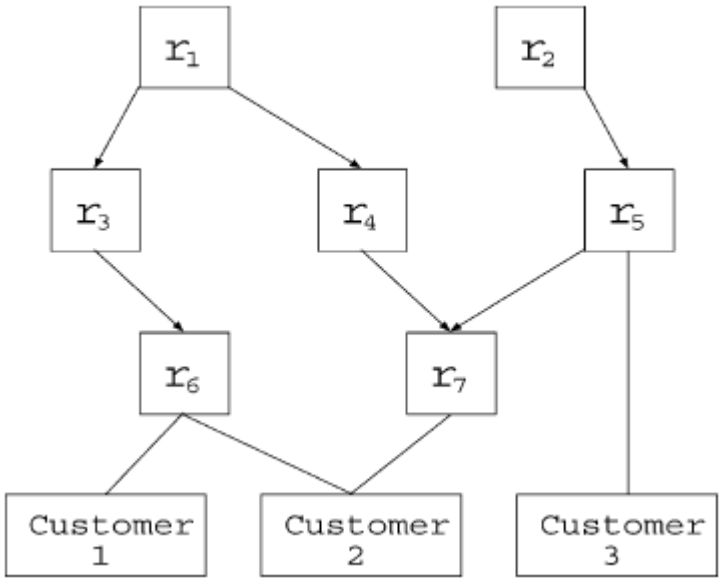

Figura 2.1: Representação gráfica do exemplo do Problema do Próximo Release.
Fonte: MA, MOURA. Algoritmos de Otimização para a Solução do Próximo Release, 2014

Em um caso especial da formulação do problema, onde a única restrição é a limitação de custo de implementação, o problema é básico e pode ser formulado como o Problema da Mochila, que pode ser enunciado da seguinte maneira:

"Um viajante levará para sua viagem apenas uma mochila, de capacidade limitada, a qual ele deverá preencher com objetos que lhe serão úteis para a viagem. Cada objeto ocupa uma certa capacidade da mochila e possui um determinado valor de importância para o viajante. O problema é decidir quais objetos deverão ser colocados na mochila de tal forma que o valor de importância seja maximizado."

O Problema da Mochila possui diferentes abordagens de formulação. Para o problema aqui tratado, a abordagem utilizada será a 0/1 unidimensional, que considera que a mochila possui apenas uma dimensão e assume que cada objeto possui apenas um peso e um valor pertencentes ao conjunto dos números naturais. Para cada um dos n objetos j, $x_j = 1$ se este objeto estiver na mochila e caso não esteja, $x_j = 0$. p_j é o valor do objeto j, w_j seu peso e W é a capacidade total da mochila. A solução é encontrar um vetor $x(x_1, x_2, ..., x_n)$ que maximize

$$\sum_{j=1}^{n} p_j x_j (p_j \in \mathbb{N}^*).$$

sujeito a

$$\sum_{j=1}^{n} w_j x_j \leq W (w_j, W \in \mathbb{N}^*),$$

onde $x_j \in \{0, 1\}, 1 \leq j \leq n$.

Se as variáveis do problema pudessem assumir valores reais, ele seria um problema de Programação Linear e uma solução ótima poderia ser encontrada em um tempo razoável, já que problemas de Programação Linear podem ser resolvidos em tempo polinomial. Como as variáveis são inteiras, podendo apenas assumir valores binários, temos um caso de Programação Linear Inteira que pode ser classificado como NP-Difícil [5], ou seja, não é conhecido nenhum algoritmo de complexidade polinomial para solucioná-lo.

2.2 Estado da Arte

O primeiro trabalho sobre o NRP foi o trabalho desenvolvido por Bagnall *et al* [3], onde é apresentada a primeira formulação do problema, que possui como objetivo maximizar a satisfação do cliente atendendo os requisitos de maior importância para eles na *release*. Para validar a proposta, os autores realizam testes com métodos exatos, gulosos e heurísticos como, por exemplo, algoritmo GRASP (*Greedy Randomized Adaptive Search Procedure*), *Hill Climbing* e *Simulated Annealing*. Os autores utilizaram cinco modelos de sistemas para avaliação dos algoritmos, e concluíram que para sistemas que envolvem poucos requisitos e patrocinadores, um algoritmo de otimização exata é suficiente. No entanto, para problemas maiores, o *Simulated Annealing* se mostrou mais eficiente ao encontrar os melhores resultados e em um tempo de execução satisfatório.

Considerando essa modelagem de seleção de clientes, uma abordagem utilizando uma adaptação da metaheurística *Greedy Randomized Adaptive Search Procedure* (GRASP), chamada de GRASP Reativo, é mostrada em [15]. Constata-se que esse algoritmo é adequado ao problema quando é comparado com a *Simulated Annealing*, Algorítimo Genético e GRASP tradicional.

Ainda neste contexto, [7] faz um estudo comparativo entre Algoritmo Genético, *Simulated Annealing* e o Colônia de Formigas aplicado ao NRP. Nesse estudo, os autores mostram que o Colônia de Formigas consegue encontrar soluções com o máximo de satisfação do cliente no limite do orçamento estipulado para o *release*. Já o Algoritmo Genético mostra uma performance parecida com o Colônia de Formigas mas com uma grande variabilidade na satisfação e por fim, o *Simulated Annealing* encontra os piores resultados tanto no limite de orçamento máximo do release quanto na satisfação dos clientes.

Souza e Ferreira [9] fizeram um estudo comparativo entre Colônia de Formigas, Algoritmo Genético e *Simulated Annealing* aplicado ao NRP com o uso de interdependência entre requisitos. Neste problema, as interdependências como, por exemplo, custo (selecionar um requisito altera o custo de outro requisito), valor (selecionar um requisito altera valor de outro requisito) e precedência (selecionar um requisito somente se outro for selecionado) foram incorporados na escolha do conjunto de requisitos. As melhores soluções foram obtidas com o Colônia de Formigas quando comparadas com as soluções encontradas com outras metaheurísticas, mas com um tempo em média 28% maior.

Em [13] é apresentado um algoritmo de espinha dorsal (tradução livre para *Backbone Algorithm*) para resolução de grandes instâncias do NRP, como seleção de clientes. Essa técnica reduz constantemente a instância através da detecção de partes comuns de vários ótimos locais. Essas partes comuns formam a espinha dorsal da solução final. Após um certo grau de diminuição, a instância reduzida é solucionada utilizando programação linear. Resultados são comparados com a LMSA (variação do *Simulated Annealing*), algoritmo este que vinha encontrando os melhores resultados até então.

Considerando a importância das capacidades humanas de intuição e negociação, [17] faz um estudo do processo de planejamento de *release* e propõe uma abordagem híbrida de planejamento que integra a inteligência computacional com o conhecimento e as experiências humanas, oferecendo flexibilidade no número de *releases* a serem planejadas. A solução computacional adotada por eles é baseada em resolver uma sequência de problemas de Programação Linear sem condições inteiras, combinada com heurísticas para acelerar o processo e gerar mais de uma boa solução.

Em [20] o problema foi formulado de forma multiobjetiva, com o propósito de maximizar a satisfação do cliente e minimizar o custo de implementação dos requisitos. Para resolver o problema, os autores realizaram experimentos com algumas metaheurísticas multiobjetivas das quais o NSGA-II (Nondominated Sorting Genetic Algorithm II) [6] superou as outras técnicas em problemas de maior escala.

No artigo [4], é proposta uma solução para o problema em sua versão multiobjetiva, utilizando a abordagem evolucionária multiobjetiva NSGA II [6] e comparando-a com outros algoritmos evolucionários por meio de métricas de performance.

Por fim, Paixão e Araújo [2] propõem a utilização de um Algoritmo Genético Interativo com aprendizado de máquina como forma de substituir o usuário no processo de avaliação das soluções geradas para o Problema do Próximo Release. Na abordagem o usuário é convidado a avaliar as soluções geradas fornecendo ao algoritmo uma nota que representa o quão satisfeito ele está com a solução gerada. Como forma de validar a abordagem os autores utilizaram simuladores responsáveis por representar o perfil de avaliação de possíveis usuários do sistema. Os resultados mostram que o algoritmo conseguiu gerar soluções que são cerca de 70% similar quando comparado com as soluções alvos utilizadas nos simuladores.

Capítulo 3

Proposta para Empresas de Software que Utilizam Scrum

Para este trabalho, além de levantamento e estudo de material referente à a medologia Scrum e as especificidades das empresas que a utilizam, que servirá de entrada para a adequação do NRP original, serão analisadas os processos de um grupo com 7 empresas de manutenção e evolução de *software*.

3.1 O Problema das Empresas

As empresas supra citadas são organizações que atuam no ramo de evolução e manutenção de *software*, tendo como principal característica a demanda contínua de solicitações por parte dos seus clientes para correções ou melhorias nos seus produtos de *software*.

O processo de desenvolvimento destas organizações é iterativo e incremental, utilizando conceitos da metodologia ágil Scrum [18]. Esta metodologia é baseada no conceito de "dividir para conquistar" e consiste na repetição de ciclos de PDCA (método iterativo de gestão usada para melhoria contínua de processos e produtos cuja sigla vem do inglês: *Plan - Do - Check - Act*) [1] de curta duração, com o intuito de facilitar o gerenciamento do trabalho ao dividí-lo em partes menores, trazer respostas mais rápidas para o cliente em termos de incrementos de *software* funcionando, assim como receber frequentemente seu *feedback*, favorecendo a

adaptação à mudanças, como mostrado na Figura 3.1[4].

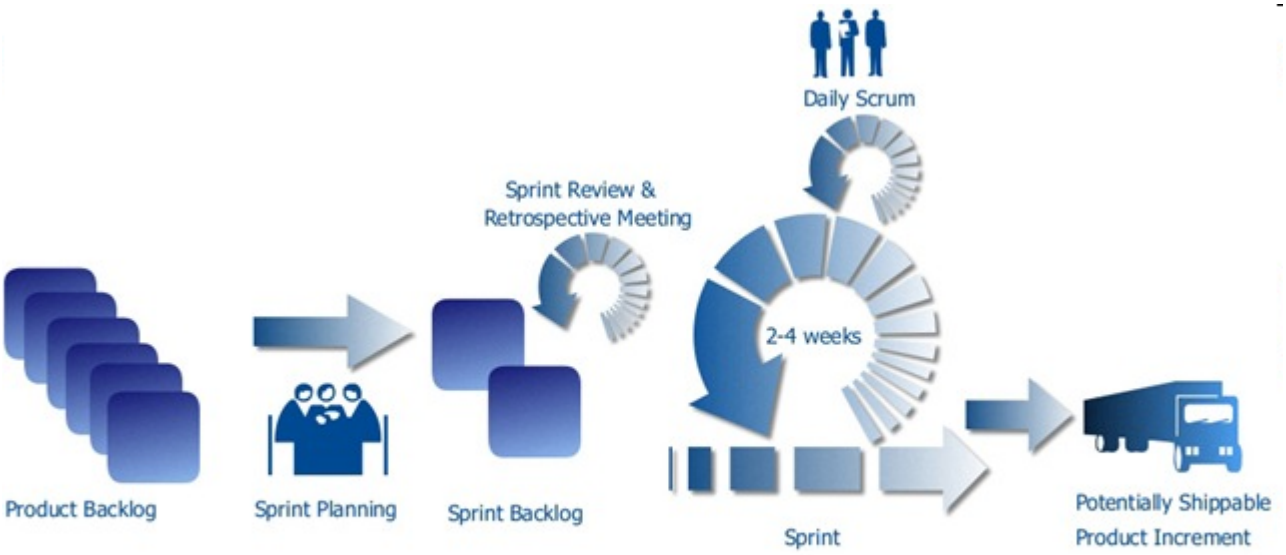

Figura 3.1: Processo do Scrum.

Com o uso do Scrum é possível a identificação do Problema do Próximo *Release*, pois esta metodologia divide a gestão do desenvolvimento total em partes menores e de mesma duração, conhecidas como *sprints*, que representam um ciclo de trabalho. A cada *sprint* um conjunto de requisitos é implementado, tendo como resultado um incremento do produto que está sendo desenvolvido. Os requisitos devem ser selecionados da melhor maneira para compor um destes ciclos de trabalho do SCRUM, de tal forma que o valor da satisfação dos clientes atendidos seja maximizada e as depências dos requsitos sejam atendidas.

Com a utilização deste processo, surge a necessidade de distribuir entre as *sprints* os requisitos a serem desenvolvidos em cada iteração, contudo, esta distribuição não é trivial. O custo de desenvolvimento por *sprint* é limitado devido ao tempo de duração do ciclo, além de que, as empresas possuem vários clientes demandando ao mesmo tempo, portanto, faz-se necessário priorizar e escolher da melhor forma conjuntos de requisitos, que caibam nestas limitações, desenvolvendo em uma certa ordem o necessário para manter os clientes satisfeitos levando em consideração a importância de cada um deles, e as decisões estratégicas das organizações.

[4]Fonte: http://www.codeproject.com/Articles/551161/Agile-Testing-Scrum-and-eXtreme-Programming

Dentre os 12 princípios do manifesto ágil[5], temos "Garantir a satisfação do cliente, entregando rápida e continuamente *software* funcional"e "Colaboração com clientes mais do que negociação de contratos", ou seja, satisfazer os clientes é um dos pilares do manifesto ágil, e assim das metodologias vindouras dele. Com isso o cliente tem participação importantíssima para a priorização dos requisitos. No Scrum, cada requisito possui um atributo chamado *Business Value* ou BV, que é o grau de importância que o requisito representa para o cliente ou para o *Product Owner*, que é o seu representante.

É de extrema importância que o tamanho dos requisitos e os seus valores de importância estejam devidamente preenchidos para que a priorização dos requsitos se dê de formar adequada para atender os *stakeholders*.

Por fim, para alguns *softwares*, são mapeados as depências entre os requisitos, que podem ser:

Tipo de Dependência	**Descrição**
Precedência	O requisito r é selecionado antes do requisito r'.
And	Quando um requisito r é selecionado, o requisito r' também tem que ser escolhido.
XOr	Os requisitos r e r' são conflitantes entre si. Somente um entre r e r' pode ser selecionado.

Tabela 3.1: Dependências entre requsitios.

Para a adaptação do Problema do Próximo Release à realidade de empresas que utilizam o Scrum como metodologia de desenvolvimento, é preciso que seja feita a especificação de alguns termos utilizados na metodologia [16].

- ***Backlog*:** É o conjunto de todos os requisitos que a empresa possui para serem desenvolvidos englobando novas funcionalidades e defeitos ou bugs.
- ***Sprint*:** É cada um dos ciclos nos quais o desenvolvimento total é subdividido. Uma *sprint* é tida como a unidade básica de desenvolvimento do Scrum, geralmente com duração entre uma à quatro semanas, cujo objetivo é dividir o desenvolvimento e gerenciamento total em partes menores (iterações) ao final das quais são gerados incrementos dos produtos.

[5]http://www.manifestoagil.com.br/

- **Time:** Equipe de desenvolvimento.
- ***Time-box* da *sprint*:** É a limitação de tempo da *sprint*. Cada *sprint* deve começar e terminar dentro de um *time-box*. O *time-box* estabelece um padrão de desenvolvimento e serve para ajudar os gerentes a extrair medidas que podem auxiliá-los no monitoramento e controle do projeto, como por exemplo, a quantidade média de pontos que a equipe consegue desenvolver dentro de uma *sprint*.
- **Histórias:** No Scrum, são equivalentes aos requisitos.
- **Capacidade:** É o tamanho máximo de pontos que o time consegue entregar durante o tempo pré-estabelecido da *sprint*.
- **Tamanho:** É o tamanho de cada requisito, cuja unidade básica é o ponto, dada pela estimativa com o uso da técnica *Planning Poker*[6].
- ***Planning Poker*:** Técnica de estimativa do tamanho dos requisitos na qual, em vez de realizar estimativas em horas, a equipe estima uma pontuação para cada requisito, que leva em consideração a complexidade de implementação e o tamanho do requisito, é baseada no valor relativo de uma referência, também chamada de caso base. Por exemplo, o time decide que a referência para a estimativa será a implementação de um cadastro, que possui tamanho de 2 pontos. Os outros requisitos são estimados com base nesta referência, para mais, se forem julgados maiores ou para menos, caso contrário. Cada membro da equipe sugere uma pontuação para o requisito e caso haja divergências, a equipe discute até que todos cheguem em um consenso. A estimativa pode variar entre (0, 0.5, 1, 2, 3, 5, 8, 13, 20, 40, 100).
- **Valor de Negócio:** *O Business Value*, ou Valor de Negócio, é um número dado pelo cliente ou *Product Owner* ao requisito levando em questão a relevância do requisito para ele.

[6]https://www.planningpoker.com/

Dessa forma, adequando o estudo de caso ao Problema do Próximo *Release*, temos:

- Um *backlog* de histórias a serem desenvolvidas pelo time.
- Um conjunto de clientes que possuem diferentes valores de importância para a organização. Esses valores são baseados no percentual de contribuição destes clientes no faturamento da empresa ou decisões estratégicas para a organização.
- Uma limitação de pontos a serem desenvolvidos dada pelo time-box da *sprint*.
- Cada história tem um tamanho medido em pontos do *Planning Poker*.
- Cada cliente possui um subconjunto de histórias que deseja que sejam implementadas.
- Cada história possui um valor de negócio dado por cada cliente com interesse nela.

Com base nessas definições, temos as seguintes relações entre o problema original e a abordagem utilizada nas Empresas:

Problema original	**Estudo de caso**
Equipe de desenvolvimento.	Time.
Release.	*Sprint*.
Conjunto de requisitos.	*Backlog*.
Requisito.	História.
Custo de implementação do requisito.	Tamanho do requisito.
Custo máximo disponível por time.	Capacidade do Time.
Limitação de Custo para a Empresa .	Soma das Capacidades dos Times de desenvolvimento.

Tabela 3.2: Associação dos elementos do problema original aos do estudo de caso.

Além das especificidades apontadas à na Tabela 3.2, estas empresas, assim como muitas que utiliziam o Scrum como metodologia de desenvolvimento, possuem um ou mais times de desenvolvedores para alocarem as histórias do *backlog*. Cada time possui sua capacidade especifíca baseada na sua velocidade (pontos que consegue entregar no *time-box* da *sprint*).

3.2 Adequação do Modelo

Realizando as alterações necessárias apresentadas na seção anterior e na Tabela 3.2, temos que para o modelo proposto os índices i serão utilizados para representar os requisitos ou histórias, j para os clientes e t para os times. A notação utilizada na modelagem está descrita na Tabela 3.3.

Conjuntos	**Descrição**
Requisitos	Conjunto de requisitos a serem desenvolvidos, $i = \{1,2,...,n\}$.
Clientes	Conjunto de clientes com valor para a empresa, $j = \{1,2,...,m\}$.
Times	Conjunto de times de desenvolvimento disponíveis na empresa, $t = \{1,2,...,s\}$.
Parâmetros e Símbolos	**Descrição**
vrc_{ij}	É valor do requisito i para o cliente j, podendo variar de 0 a 10 de acordo com a sua importância.
vc_j	É o valor do cliente j para a empresa.
$pvrc_{ij}$	É o percentual da importância do requisito i para o cliente j. $(\frac{vrc_{ij}}{vtr_j})$.
vtr_j	É a soma dos valores atribuídos a cada requisito i do cliente j. Estes valores são. determinados pelo grau de importância que cada requisito i tem para o cliente j $(\sum_{i=1}^{n} vrc_{ij})$.
$tamTime_t$	Número de integrantes time t.
$capPPP_t$	Representa a capacidade máxima de desenvolvimento do time t em pontos do *Planning Poker*.
$dursprint$	O tempo em horas disponíveis no *time-box* da *sprint*.
$disp_t$	Representa a disponibilidade em horas produtivas levando em consideração. os integrantes do time t. $(tamTime_t * dursprin.)$.
tam_i	Representa o tamanho em pontos do *Planning Poker* que cada requisito i possui.
$horaPonto_t$	Representa a estimativa em horas da duração de um ponto do *Planning Poker* no time t, ou seja, quantas horas são necessárias para a implementação de um ponto. $(\frac{disp_t}{capPPP_t})$.
$tempoDev_{it}$	É o tempo que o requisito i leva para ser desenvolvido pelo time t. $tam_i * horaponto_t$.
$valorHora_t$	É o valor pago pela empresa por hora de desenvolvimento do time t.
$G(R, E)$	Representa a dependência entre os requisitos, como no modelo tradicional. O grafo é direcionado e acíclico. $(r, r') \in (G)$ se e somente se r é um pré-requisito de r'.

$G1(R, E)$	Representa os requisitos que devem ser implementados juntos (*And*). O grafo não é direcionado e pode haver ciclos. $(r, r') \in E(G)$ se e somente se r e r' assumem o mesmo valor. Ambos são selecionados ou não.
$G2(R, E)$	Representa os requisitos conflitantes (XOr). O grafo não é direcionado e pode haver ciclos. $(r, r') \in E(G)$ se e somente se r e r' assumem valores distintos. Se um deles é selecionado, o outro não é.

Variáveis	**Descrição**
r_i	Variável binária que assume o valor 1 se o requisito i for selecionado para a *sprint*.
x_{it}	Variável binária que assume o valor 1 quando o requisito i é implementado pelo time t.
y_j	Variável que pode assumir valores reais no intervalo $[0, 1]$ e que indica o percentual de satisfação do cliente j. Por exemplo, se y_j=1, todos os requisitos do cliente j são selecionados.

Tabela 3.3: Notação da modelagem do problema.

Tendo em vista que neste caso o objetivo não é em função dos clientes, e sim dos requisitos, não precisa haver uma garantia de que todas as histórias de um cliente j devem ser implementadas para que ele esteja satisfeito. O objetivo é calcular a satisfação do cliente de forma qualitativa e não quantitativa, ou seja nem todos os requsitos dele devem ser implementados, e sim, o máximo possível levando em conta o vcr_{ij}.

No modelo tradicional, a satisfação global do cliente para a empresa é dada por uma varíavel y_j que assume 1 quando todos os requisitos do cliente foram selecionados para o *release* ou 0 caso contrário. Na adequação realizada, esta variável poderá assumir valores reais entre 0 e 1, que indicam a porcentagem de satisfação, como visto Tabela 3.3, e pode ser formalizada como:

$$y_j = \sum_{i \in Requisitos}^{n} pvrc_{ij} * r_i.$$

Nesta modelagem foram propostas 3 funções objetivos que irão auxiliar o gerente de projetos à obter a melhor alocação de requisitos em seus times para a *sprint*, podendo:

- Maximizar a satisfação do cliente

$$max \sum_{j \in Clientes} vc_j * y_j.$$

- Minimizar o tempo de desenvolvimento

$$min \sum_{i \in Requisitos} \sum_{t \in Times} tempoDev_{it} * x_{it}.$$

- Minimizar o custo de desenvolvimento

$$min \sum_{i \in Requisitos} \sum_{t \in Times} tempoDev_{it} * valorHora_t * x_{it}.$$

Sujeitas às seguintes restrições:

- Restrições:

  Disponibilidade do Time - A disponibilidade dos times de desenvolvimento não pode ser extrapolada.

$$\sum_{i \in Requisitos} tempoDev_{it} * x_{it} \leq disp_t, \quad \forall t \in Times.$$

  Requisito Implementado por Apenas um Time - Cada requisito i só pode ser alocado em apenas um time t.

$$\sum_{t \in Times} x_{it} = r_i, \quad \forall i \in Requisitos.$$

  Precedência entre Requisitos - O requisito r é pré-requisito do requisito r'

$$x_r \geq x_{r'}, \quad \forall (r, r') \in E(G).$$

  Requisitos que Devem ser Implementados Juntos - Quando um requisito r é selecionado, o requisito r' também tem que ser escolhido

$$x_r - x_{r'} = 0, \quad \forall (r, r') \in G1(R, E).$$

  Requisitos Conflitantes - Os requisitos r e r' são conflitantes entre si. Somente um entre r e r' pode ser selecionado

$$x_r + x_{r'} = 1, \quad \forall (r, r') \in G2(R, E).$$

Capítulo 4

Estudo de Caso

4.1 Metodologia de Solução

O modelo matemático foi implementado na linguagem de modelagem matemática da ferramenta de otimização AIMMS[7]. Esta ferramenta pode ser obtida gratuitamente para propósitos acadêmicos e já vem integrada com diversos pacotes de otimização (CPLEX, MINOS, XPRESS, entre outros). Além disso, a ferramenta também está integrada com uma biblioteca que possibilita a construção de interfaces amigáveis. A representação textual do modelo encontra-se no ApêndiceC.

O modelo é um problema de programação inteira mista e para sua solução no AIMMS foi utilizado o *solver* CPLEX versão 12.6.2. O CPLEX é um *solver* que hoje pertence à IBM e resolve problemas de programação inteira, problemas grandes de programação linear, problemas de programação quadrática convexos e não-convexos e problemas de programação inteira mista[8].

O algoritmo do CPLEX para programação inteira mista é um *Branch and Bound* com melhorias. O método *Branch and Bound*, utiliza a estratégia de dividir para conquistar, e também encontra sempre a solução ótima, porém, não percorre todo o espaço de soluções: ele percorre uma árvore de enumeração (*branch*) que particiona o conjunto de soluções do problema e, sempre que percorre um ramo que não é promissor ou que é inviável, ele poda-o

[7]https://www.aimms.com/

[8]http://main.aimms.com/aimms/solvers/cplex/

(*bound*) sem percorrê-lo. Para cada iteração do *Branch and Bound* é resolvido um problema de programação linear.

A poda é realizada de acordo com a estimativa dos limites superior e inferior para o valor mínimo (caso a função objetivo seja minimizar) ou máximo (caso a função objetivo seja maximizar), dado um subconjunto S. Com a comparação desses limites é possível eliminar subespaços de S que possuam soluções fracas.

Em seguida é apresentado o Pseudocódigo do algoritmo *Branch and Bound* para o Problema da Mochila.

Algoritmo 1: Pseudocódigo do algoritmo Branch and Bound para o Problema da Mochila.

Entrada: Número de itens candidatos, Lista de pesos $[p_1, ..., p_n]$, Lista de valores $[v_1, ..., v_n]$, Capacidade da mochila c. Obs.: Os itens devem estar ordenados de acordo com a razão valor/peso.

Saída: Solução ótima *melhorsolucao*

inicio

N = número de itens candidatos;

Pesos = lista de pesos;

Valores = lista de valores;

Conjunto de nós ativos: $NosAtivos \leftarrow \{raiz\}$;

valormaximo = `Valor` (raiz);

Inicialização do conjunto da melhor solução: $melhorsolucao = \emptyset$;

repita

Escolher um nó para ramificar: no = `Escolher(`*NosAtivos*`)`;

se `Limite Superior` *(no, c) > valormaximo* **então**

Gerar filho esquerdo (considera que o próximo item entra na mochila) do nó corrente para incluir o próximo item;

se `Valor Acumulado` *(filho esquerdo) > valormaximo* **então**

valormaximo = `Valor Acumulado` (filho esquerdo);

Atualizar melhorsolucao;

fim

se `Limite Superior` *(filho esquerdo, c) > valormaximo* **então**

$NosAtivos \leftarrow \{filhoesquerdo\}$;

fim

Gerar filho direito (considera que o próximo item não entra na mochila) do nó corrente para incluir o próximo item;

se `Limite Superior` *(filho direito, c) > valormaximo* **então**

$NosAtivos \leftarrow \{filhodireito\}$;

fim

fim

até $NosAtivos = \emptyset$;

fin

Algoritmo 2: Pseudocódigo da função Limite Superior do Branch and Bound.

Entrada: no, Capacidade da mochila

Saída: Limite superior: *limitesuperior*

inicio

limitesuperior = `Valor Acumulado` (no) + (Capacidade da mochila - `Peso` (no))
$* \left(\frac{\texttt{Valor}(proximo - no)}{\texttt{Peso}(proximo - no)} \right)$;

fim

4.2 Experimentos Realizados

Após a observaçao dos processos das empresas apresentadas no Capítulo 3, duas organizações foram selecionadas para os experimentos dos estudos de caso. Estas empresas estão passando por consultoria da SWQuality Consultoria e Sistemas para melhoria de processos de desenvolvimento de software alinhados ao nv.2 de maturidade no *Capability Maturity Model Integration for development* - CMMI-DEV [10] com a implantação e maturação da metodologia ágil Scrum iniciada em novembro de 2014. Seus nomes foram eticamente preservados, recebendo a denominação fictícia de "Empresa X"e "Empresa Y".

Para os experimentos, vamos ter que levar em consideração que para cada empresa, os times possuem habilidade para desenvolver quaisquer dos requisitos presentes no *backlog* dos diversos clientes. As empresas possuem duração de sprint semanal, ou seja 40 horas, na Empresa X, e 44 horas, na Empresa Y, para entrega dos requisitos selecionados para a sprint. Quanto ao custo, será inferido de acordo com a produtividade. Será atribuído um valor que a empresa gasta por hora com cada time, sendo o time mais caro, aquele com melhor desempenho baseado em seu histórico obido com o gerente de projetos.

Para o estudo de caso foi realizada uma variação da duração da sprint semanal para duas e três semanas com o o com o objetivo de verificar a distribuição dos pontos por time nas diferentes funções objetivos, minimizando o tempo e o custo.

A Figura 4.1 apresenta a tela do AIMMS com o modelo matemático. Foram inseridos os modelos com as três funções objetivos, cada uma das restrições, as variáveis e os parâmetros apresentados.

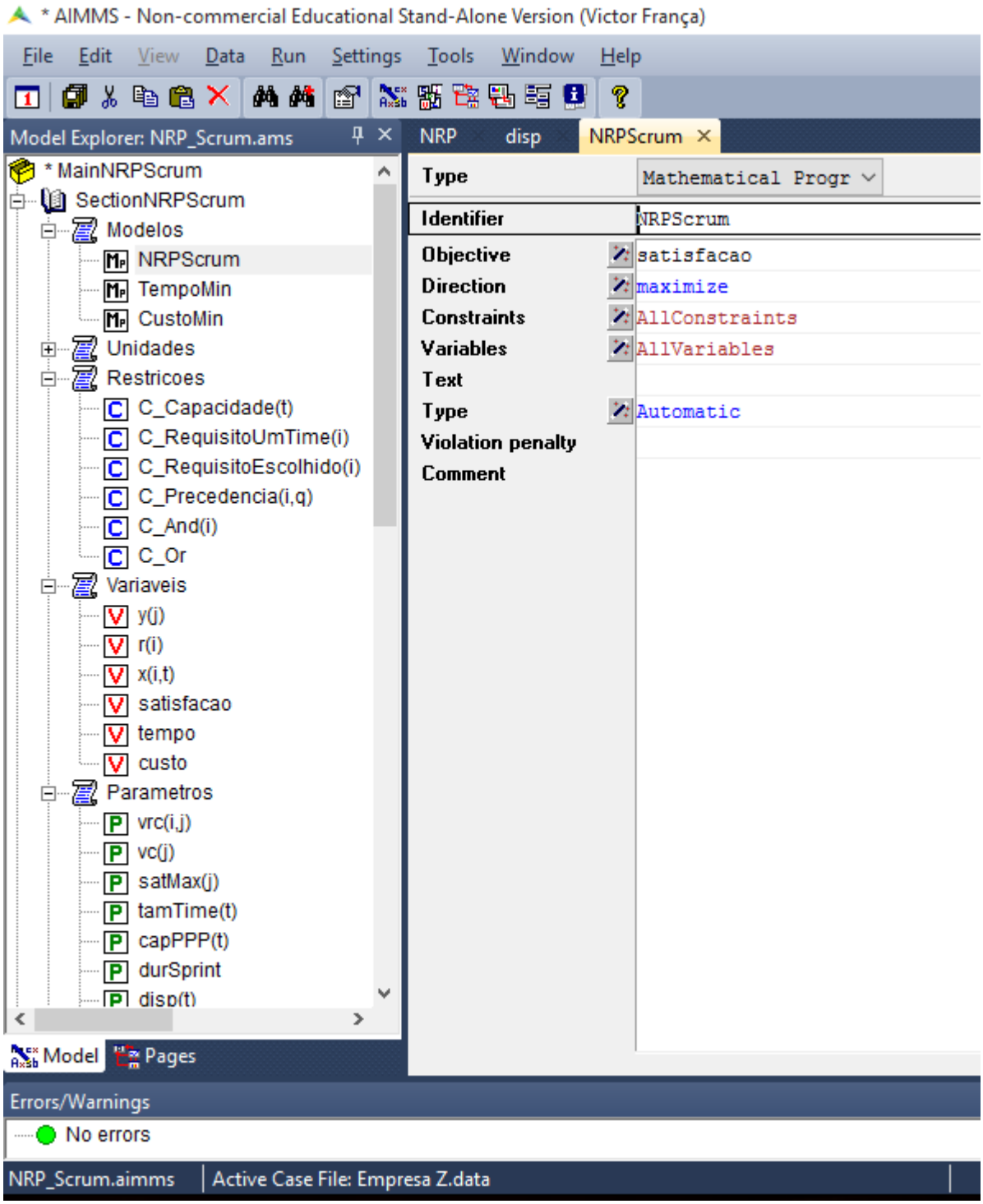

Figura 4.1: Modelagem do Sistema no AIMMS.

A Figura 4.2 mostra os dados referentes ao parametro $tempoDev_{it}$, que é o tempo que cada requisito leva para ser desenvolvido por cada time.

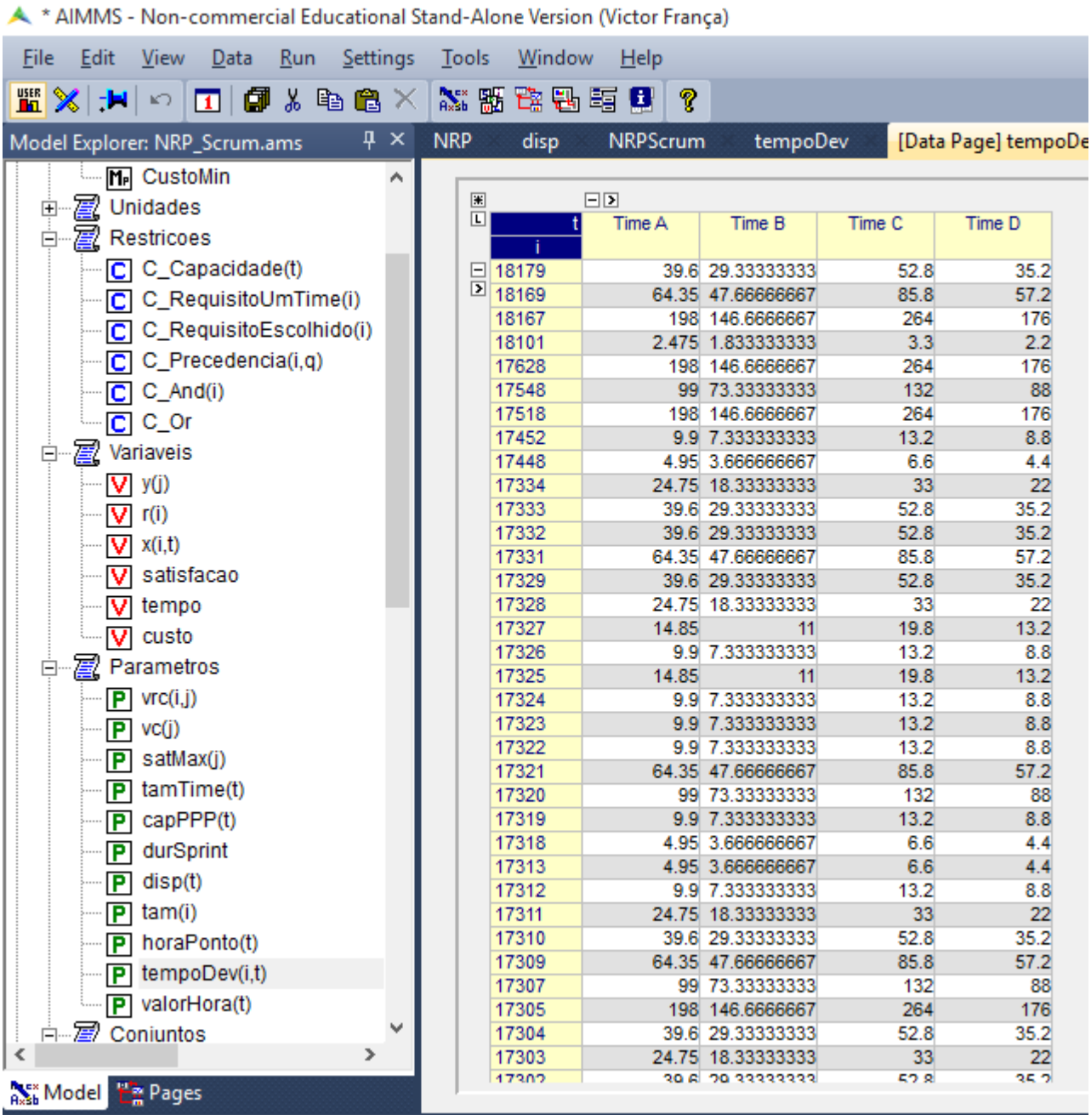

Figura 4.2: Informações Referentes ao Tempo de Desenvolvimento.

A Figura 4.3 apresenta uma interface inicial que mostra para o usuário a alocação dos requisitos selecionados para a próxima *sprint* em cada time de desenvolvimento da empresa, assim como seu tempo de desenvolvimento. Também é possível escolher qual das três funções objetivos se deseja executar. É possível observar o percentual de satisfação dos clientes para a empresa, o custo do desenvolvimento da sprint e o tempo em horas da iteração

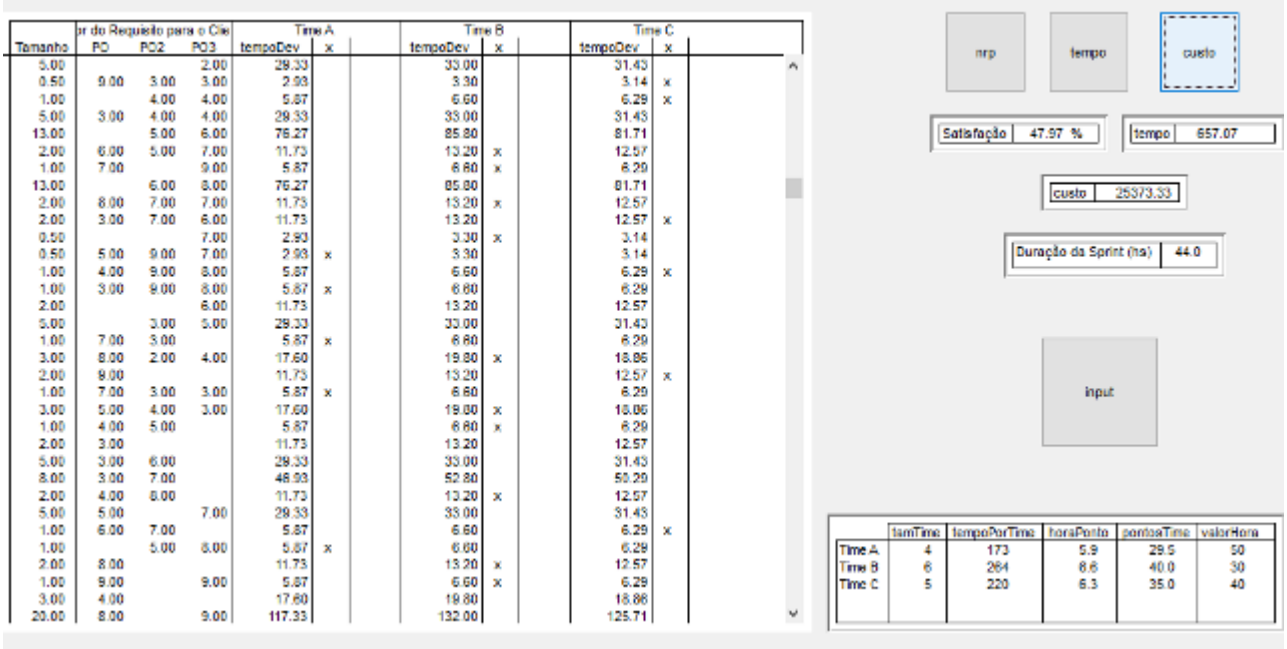

Figura 4.3: Interface Inicial.

Por fim na Figura 4.4 são apresentadas as informações técnicas referentes à execução do modelo matemático no AIMMS, assim como qual *Solver* foi utilizado, quantas interações, constantes e variáveis utilizadas e o tempo total da execução, entre outras informações.

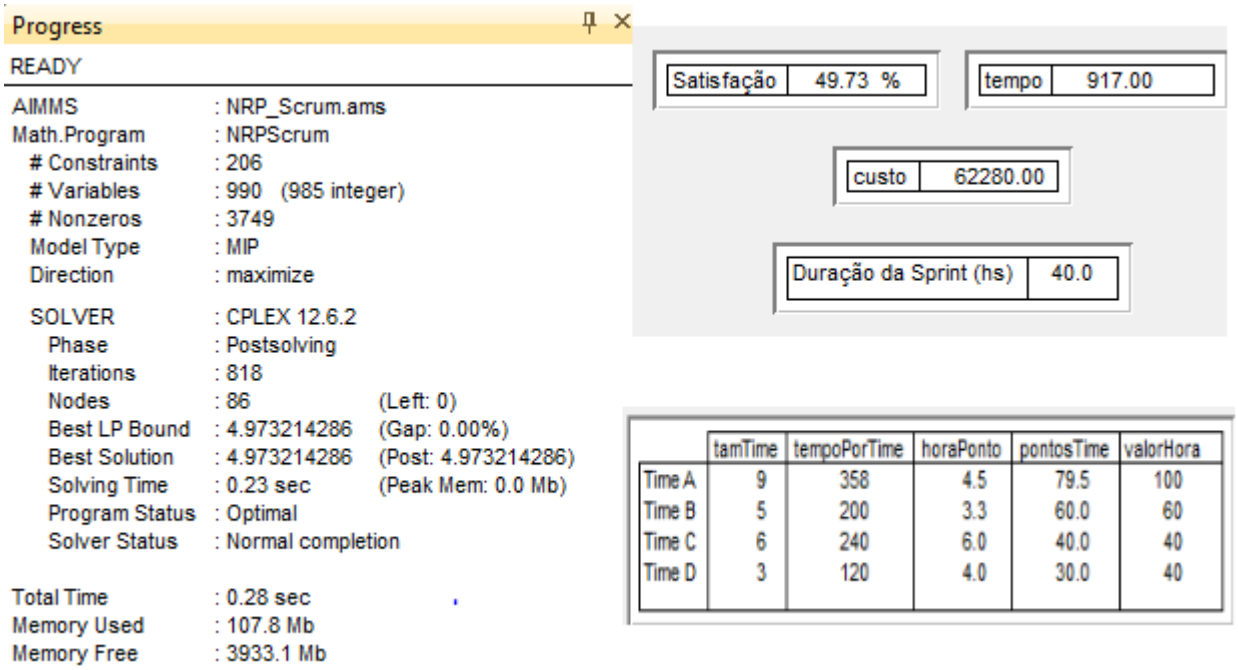

Figura 4.4: Execução do Modelo Minimizando o custo.

4.2.1 Empresa X

Na Empresa X tínhamos, no momento da coleta das informações, um *backlog* com 198 itens a serem desenvolvidos, estimados e priorizados. As informações referentes aos tamanhos dos requisitos encontram-se no Anexo A. Na Tabela 4.1 são apresentados os times da empresa, quantos membros cada time possui, qual a relação hora por ponto e o valor pago pela hora em cada time:

Time	Tamanho do Time	Hora por Ponto	Valor da Hora
Time A	9	4,5	R$100,00
Time B	5	3,5	R$60,00
Time C	6	6,0	R$40,00
TIme D	3	6,0	R$40,00

Tabela 4.1: Informações dos times da Empresa X.

Foi realizada a vairação da duração da *sprint* entre uma, duas e três semanas com as funções objetivos de minimizar o tempo e em seguida minimizar o custo, assim como mostrado na Figura 4.5.

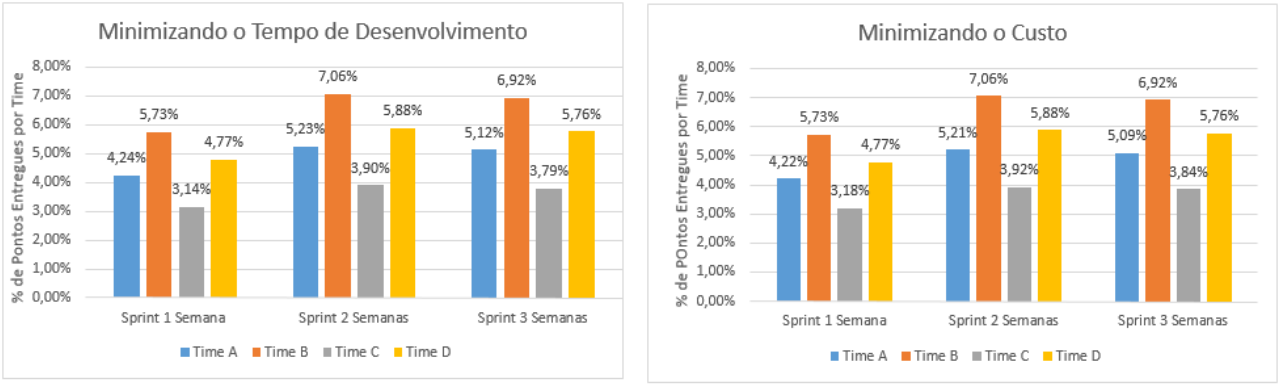

Figura 4.5: Minimizando o Tempo e o Custo para a Empresa X.

Como temos mais requisitos no *backlog* do que a capacidade dsiponível de desenvolvimento das equipes, o modelo faz o máximo para alocar a maior quantidade de requisitos possíveis em cada time, ou seja, a capacidade alocada tende a ser o mais próxima possível da capacidade disponível.

O que muda é que quando aplicamos a função de minimizar o custo percebe-se uma diminuição de pontos alocados por membro da equipe no Time A, que possui um valor de hora mais elevado e um aumento de pontos alocados ao time C, que possui menor valor de hora. Essa diminuição poderia ser observada tanto no Time C quanto no D, que possuem mesma produtividade e custo para a empresa.

4.2.2 Empresa Y

Para a empresa Y a amostra de requisitos possuia 160 itens, e havia três times de desenvolvimento na organização. Três *Product Owners* faziam a interface com os clientes e priorizavam os requisitos, sendo considerados para a análise o valor do requisito para eles.

Informações referentes aos tamanhos dos requisitos encontram-se no Anexo B e dados referentes aos times são apresentadas na Tabela 4.2:

Time	Tamanho do Time	Hora por Ponto	Valor da Hora
Time A	4	5,9	R$50,00
Time B	6	6,6	R$30,00
Time C	5	6,3	R$40,00

Tabela 4.2: Informações dos times da Empresa Y.

Para esta empresa também foram utilizadas as funções objetivos de minimizar custo e tempo como apresentado an Figura 4.6

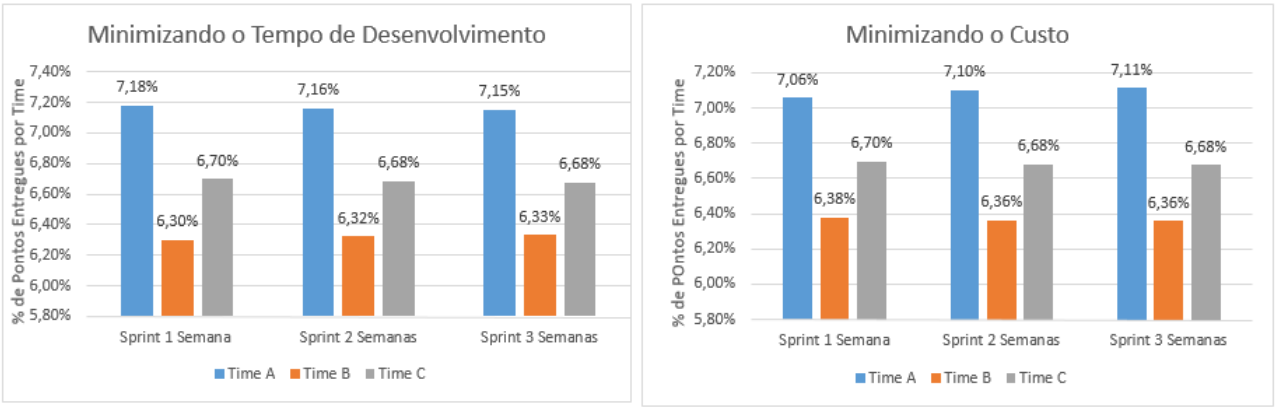

Figura 4.6: Minimizando o Tempo e o Custo para a Empresa Y.

Desta vez a variação observada na quantidade de pontos alocadas por membro das equipes pode ser observada nos times A e B, que representam, respectivamente, maior e menor custo para a organização. Novamente, pela característica do *backlog*, a variação que pode ser observada é muito pequena, pois sempre irá sobrar requisitos para as próximas *sprints*.

4.2.3 Distribuição Temporal do *Backlog* em *Sprints*

Para uma segunda análise, foi realizada uma distribuição temporal de todo o *backlog* em sprint semanais com a satisfação dos clientes para a empresa, obtida em cada iteração, como mostrado na Figura 4.7

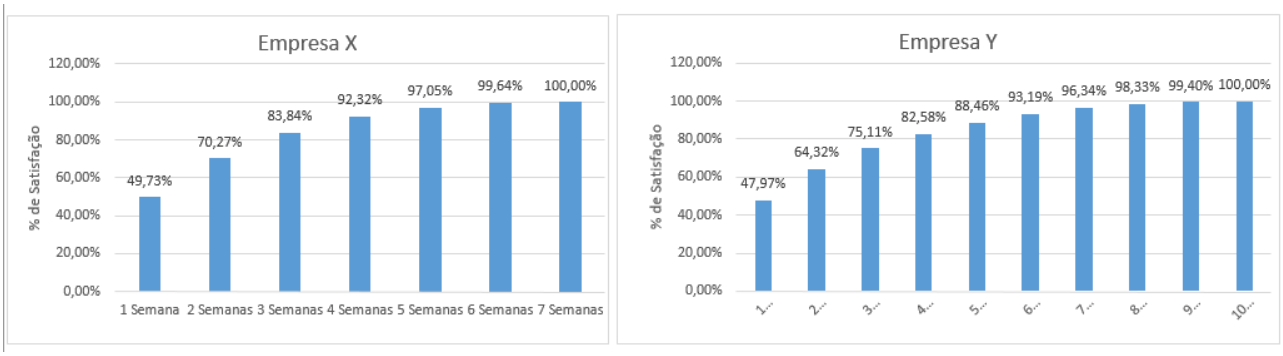

Figura 4.7: Minimizando o Tempo e o Custo para a Empresa Y.

Para a Empresa X seria necessário sete semanas e um custo de R$ 361.425,00 para acabar todo o *backlog*, isso contando que não entrasse nenhum item novo, já para a Empresa Y, dez semanas seriam necessárias com um custo de R$ 246.269,05 com os times existentes na organização.

Simulações de contratação de novos integrantes para os times foram realizadas. Primeiro contratando-se três membros para o time mais produtivo, e em seguida para o menos produtivo, como mostrado nas Tabelas 4.3 e 4.4.

Contratação	Duração	Custo
Formação Original	7 *sprints*	R$361.425,00
3 membros para o Time C	5 *sprints*	R$355.875,00
3 membros para o Time B	4 *sprints*	R$385.725,00

Tabela 4.3: Simulação de Contratação na Empresa X.

Contratação	Duração	Custo
Formação Original	10 *sprints*	R$246.269,05
3 membros para o Time B	7 *sprints*	R$246.009,50
3 membros para o Time A	6 *sprints*	R$274.186,00

Tabela 4.4: Simulação de Contratação na Empresa Y.

Com isso é possível perceber que as empresas podem diminuir o tempo de desenvolvimento de todo o *backlog* sem necessariamente aumentar o seu custo, contratando membros para os times certos. E o maior percentual de satisfação é obtido sempre nas primeiras *sprints*, devido a priorização correta dos requisitos e sua estimativa realizada previamente.

Esta análise é apenas um tipo de estudo que pode ser realizado com o modelo inicialmente proposto. Além de alocar os requisitos dentre as equipes de desenvolvimento, podendo utilizar as três funções objetivos, pode-se realizar análises como por exemplo, dado um prazo mais apertado pelo cliente, quanto de hora extra é necessária para atender o novo prazo de entrega da *sprint*; analisar individualmente a satisfação de cada cliente para decisões estratégicas de fidelização deles, entre outras análises. E também atende ao principal objetivo, que é planejar a próxima Sprint.

Capítulo 5

Conclusão

Este trabalho teve como objetivo a adaptação do *Next Release Problem* para empresas de manutenção e evolução de *software* que utilizam a metodologia ágil Scrum em seu processo de desenvolvimento.

Após observações nos processos de algumas organizações foram realizadas adaptações ao modelo tradicional do NRP, foram concebidos modelos com três diferentes funções objetivo e novos parâmetros e restrições foram criadas para atender a realidade destas empresas.

Por fim o modelo foi implementado na ferramenta AIMMS e assim, foi possível utilizá-lo para realizar a alocação de requisitos a serem implementados por times de desenvolvimento na próxima sprint.

O novo modelo traz alguns conceitos ágeis utilizados nas organizações analisadas, como por exemplo o *Business Value* dos requisitos e seu tamanho dado pela técnica *Planning Poker*. A escolha dos requisitos é dada de forma qualitativa e não quantativa como no modelo tradicional e a satisfação dos clientes varia de acordo como valor de negócio de cada requisito solicitado que é selecionado para a próxima sprint, e não pela quantidade.

A partir das três funções objetivos do novo modelo (maximizar a satisfação, minimizar o tempo e minimizar o custo), é possível tomar decisões gerenciais estratégicas para um melhor planejamento da próxima sprint.

5.1 Trabalhos Futuros

Organizações que utilizam Scrum como metodologia de desenvolvimento de software, possuem algumas outras especificidades no processo e necessidades que não foram atendidas neste trabalho, que servirão como trabalhos futuros, como por exemplo:

- Realizar o planejamento de mais sprints com o backlog existente, não apenas da próxima;
- Ajustar reserva de capacidade para escopo não planejado;
- Montar a projeção temporal do planejamento das sprints;
- Mapear habilidades dos times para usar como parâmetro na alocação dos requisitos selecionados;
- Permitir a utilização do modelo com times que possuam diferentes durações de *sprints*;
- Melhoria na interface, proporcionando uma melhor experiência para o usuário;
- Utilizar a ferramenta em uma empresa de software para o planejamento das sprints.

Referências Bibliográficas

[1] FF Andrade. *O método de melhorias PDCA.* PhD thesis, Universidade de São Paulo, 2003.

[2] Paixao MHE Araujo, AA. Machine learning for user modeling in an interactive genetic algorithm for the next release problem. *Proceedings of the 6th International Symposium on Search-Based Software Engineering*, 8636:288–233, 2014.

[3] VJ; Whittley IM Bagnall, AJ; Rayward-Smith. The next release problem. *Information and software technology*, 43(14):883–890, 2001.

[4] ; Huang Z Cai, X; Wei. Evolutionary approaches for multi-objective next release problem. *Computing and Informatics*, 31(4):847–875, 2012.

[5] CE; Rivest RL; Stein C; others Cormen, TH; Leiserson. *Introduction to algorithms*, volume 2. MIT press Cambridge, 2001.

[6] A; Agarwal S; Meyarivan TAMT Deb, K; Pratap. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *Evolutionary Computation, IEEE Transactions on*, 6:182–197, 2002.

[7] IM; Orellana FJ del Sagrado, J; del Aguila. Ant colony optimization for the next release problem: A comparative study. In *Search Based Software Engineering (SSBSE), 2010 Second International Symposium on*, pages 67–76. IEEE, 2010.

[8] AG; Rothermel G Elbaum, S; Malishevsky. Test case prioritization: A family of empirical studies. *Software Engineering, IEEE Transactions*, 2002.

[9] JT Ferreira, TN; Souza. An aco approach for the next release problem with dependency among requirements. *Proceedings of the 3rd Brazilian Workshop on Search-Based Software Engineering.*, 2012.

[10] MA Hanscom AFB. *CMMI for Development,Version 1.3.* CARNEGIE MELLON, Software Engineering Institute, 2010.

[11] BF Harman, M; Jones. Search-based software engineering. *Information and Software Technology*, 43(14):833–839, 2001.

[12] M Harman. The current state and future of search based software engineering. *IEEE COMPUTER SOCIETY - Future of Software Engineering.*, 2007.

[13] J; Ren Z Jiang, H; Xuan. Approximate backbone based multilevel algorithm for next release problem. In *Proceedings of the 12th annual conference on Genetic and evolutionary computation.*, pages 1333–1240. ACM, 2010.

[14] VR Larman, C; Basili. Iterative and incremental developments. a brief history. In Computer, editor, *Agile Methods*. IEEE Computer Society, 2003.

[15] FG; Carmo RAF; Maia CL; Souza JT Linhares, GRM.; Freitas. Aplicaçao do algoritmo grasp reativo para o problema do próximo release. *XLII Simpósio Brasileiro de Pesquisa Operacional (SBPO)*, 2010.

[16] MA Moura. Algoritmos de otimização para solução do problema do próximo release. *Universidade Federal Rural de Pernambuco*, 2014.

[17] AN Ruhe, G; The. Hybrid intelligence in software release planning. *International Journal of Hybrid Intelligent Systems*, 1:99–110, 2004.

[18] J Schwaber, K; Sutherland. The scrum guide. *Scrum. org, October*, 2011.

[19] I Sommerville. *Software Engineering.* Addison Wesley, University of St. Andrews, United Kingdom, 2011.

[20] M; Mansouri SA Zhang, Y; Harman. The multi-objective next release problem. In *Proceedings of the 9th annual conference on Genetic and evolutionary computation*, pages 1129–1137. ACM, 2007.

Apêndice A

Tamanho dos Requisitos do *Backlog* da Empresa X

Requisitos	r_1	r_2	r_3	r_4	r_5	r_6	r_7	r_8	r_9	r_{10}	r_{11}	r_{12}	r_{13}	r_{14}
Tamanhos	8	13	2	0,5	40	20	40	2	1	5	8	8	13	8

Requisitos	r_{15}	r_{16}	r_{17}	r_{18}	r_{19}	r_{20}	r_{21}	r_{22}	r_{23}	r_{24}	r_{25}	r_{26}	r_{27}	r_{28}
Tamanhos	5	3	2	3	2	2	2	13	20	2	1	1	2	5

Requisitos	r_{29}	r_{30}	r_{31}	r_{32}	r_{33}	r_{34}	r_{35}	r_{36}	r_{37}	r_{38}	r_{39}	r_{40}	r_{41}	r_{42}
Tamanhos	8	13	20	40	8	5	8	8	2	5	8	3	1	1

Requisitos	r_{43}	r_{44}	r_{45}	r_{46}	r_{47}	r_{48}	r_{49}	r_{50}	r_{51}	r_{52}	r_{53}	r_{54}	r_{55}	r_{56}
Tamanhos	2	2	2	2	2	5	8	8	13	8	5	3	3	3

Requisitos	r_{57}	r_{58}	r_{59}	r_{60}	r_{61}	r_{62}	r_{63}	r_{64}	r_{65}	r_{66}	r_{67}	r_{68}	r_{69}	r_{70}
Tamanhos	3	13	2	2	2	8	5	3	5	5	5	8	3	20

Requisitos		r_{71}	r_{72}	r_{73}	r_{84} r_{75}	r_{76}	r_{77}	r_{78}	r_{79}	r_{80}	r_{81}	r_{82}	r_{83}	r_{84}
Tamanhos	8	3	8	8	8	5	8	5	5	8	8	8	3	13

Requisitos	r_{85}	r_{86}	r_{87}	r_{88}	r_{89}	r_{90}	r_{91}	r_{92}	r_{93}	r_{94}	r_{95}	r_{96}	r_{97}	r_{98}

Tamanhos	1	13	5	8	13	2	2	2	3	3	5	8	13	40

Requisitos	r_{99}	r_{100}	r_{101}	r_{102}	r_{103}	r_{104}	r_{105}	r_{106}	r_{107}	r_{108}	r_{109}	r_{110}	r_{111}	r_{112}
Tamanhos	8	8	8	2	2	2	5	5	3	5	3	8	5	2

Requisitos	r_{113}	r_{114}	r_{115}	r_{116}	r_{117}	r_{118}	r_{119}	r_{120}	r_{121}	r_{122}	r_{123}	r_{124}	r_{125}	r_{126}
Tamanhos	2	2	3	5	8	8	5	13	1	1	1	0,5	0,5	8

Requisitos	r_{127}	r_{128}	r_{129}	r_{130}	r_{131}	r_{132}	r_{133}	r_{134}	r_{135}	r_{136}	r_{137}	r_{138}	r_{139}	r_{140}
Tamanhos	5	3	2	2	2	5	8	5	5	5	5	2	2	2

Requisitos	r_{141}	r_{142}	r_{143}	r_{144}	r_{145}	r_{146}	r_{147}	r_{148}	r_{149}	r_{150}	r_{151}	r_{152}	r_{153}	r_{154}
Tamanhos	2	20	20	2	2	2	8	8	13	5	5	13	8	8

Requisitos	r_{155}	r_{156}	r_{157}	r_{158}	r_{159}	r_{160}	r_{161}	r_{162}	r_{163}	r_{164}	r_{165}	r_{166}	r_{167}	r_{168}
Tamanhos	5	8	8	13	8	8	3	3	2	13	3	3	2	2

Requisitos	r_{169}	r_{170}	r_{171}	r_{172}	r_{173}	r_{184}	r_{175}	r_{176}	r_{177}	r_{178}	r_{179}	r_{180}	r_{181}	r_{182}
Tamanhos	2	2	8	8	8	5	5	5	5	5	3	2	3	3

Requisitos	r_{183}	r_{184}	r_{185}	r_{186}	r_{187}	r_{188}	r_{189}	r_{190}	r_{191}	r_{192}	r_{193}	r_{194}	r_{195}	r_{196}
Tamanhos	3	3	2	2	2	1	3	3	20	5	13	8	13	8

Requisitos	r_{197}	r_{198}
Tamanhos	8	8

Tabela A.1: Tamanhos dos requisitos da Empresa X.

Apêndice B

Tamanho dos Requisitos do *Backlog* da Empresa Y

Requisitos	r_1	r_2	r_3	r_4	r_5	r_6	r_7	r_8	r_9	r_{10}	r_{11}	r_{12}	r_{13}	r_{14}
Tamanhos	13	40	5	3	5	2	8	1	2	2	1	3	40	1

Requisitos	r_{15}	r_{16}	r_{17}	r_{18}	r_{19}	r_{20}	r_{21}	r_{22}	r_{23}	r_{24}	r_{25}	r_{26}	r_{27}	r_{28}
Tamanhos	13	3	1	2	40	20	3	20	13	3	1	2	3	2

Requisitos	r_{29}	r_{30}	r_{31}	r_{32}	r_{33}	r_{34}	r_{35}	r_{36}	r_{37}	r_{38}	r_{39}	r_{40}	r_{41}	r_{42}
Tamanhos	2	2	1	2	3	5	0,5	1	5	13	2	1	13	2

Requisitos	r_{43}	r_{44}	r_{45}	r_{46}	r_{47}	r_{48}	r_{49}	r_{50}	r_{51}	r_{52}	r_{53}	r_{54}	r_{55}	r_{56}
Tamanhos	2	0,5	0,5	1	1	2	5	1	3	2	1	3	1	2

Requisitos	r_{57}	r_{58}	r_{59}	r_{60}	r_{61}	r_{62}	r_{63}	r_{64}	r_{65}	r_{66}	r_{67}	r_{68}	r_{69}	r_{70}
Tamanhos	5	8	2	5	1	1	2	1	3	20	2	40	2	8

Requisitos		r_{71}	r_{72}	r_{73}	r_{84} r_{75}	r_{76}	r_{77}	r_{78}	r_{79}	r_{80}	r_{81}	r_{82}	r_{83}	r_{84}
Tamanhos	5	2	8	5	40	20	1	5	8	3	2	3	13	0,5

Requisitos	r_{85}	r_{86}	r_{87}	r_{88}	r_{89}	r_{90}	r_{91}	r_{92}	r_{93}	r_{94}	r_{95}	r_{96}	r_{97}	r_{98}

Tamanhos	1	13	5	8	13	2	2	2	3	3	5	8	13	40

Requisitos	r_{99}	r_{100}	r_{101}	r_{102}	r_{103}	r_{104}	r_{105}	r_{106}	r_{107}	r_{108}	r_{109}	r_{110}	r_{111}	r_{112}
Tamanhos	2	2	5	5	13	3	0,5	1	13	20	2	8	13	5

Requisitos	r_{113}	r_{114}	r_{115}	r_{116}	r_{117}	r_{118}	r_{119}	r_{120}	r_{121}	r_{122}	r_{123}	r_{124}	r_{125}	r_{126}
Tamanhos	2	20	13	13	8	5	8	5	40	5	1	5	5	5

Requisitos	r_{127}	r_{128}	r_{129}	r_{130}	r_{131}	r_{132}	r_{133}	r_{134}	r_{135}	r_{136}	r_{137}	r_{138}	r_{139}	r_{140}
Tamanhos	5	5	8	0,5	13	1	3	8	5	3	5	8	2	3

Requisitos	r_{141}	r_{142}	r_{143}	r_{144}	r_{145}	r_{146}	r_{147}	r_{148}	r_{149}	r_{150}	r_{151}	r_{152}	r_{153}	r_{154}
Tamanhos	5	2	5	5	13	13	5	8	5	2	2	2	5	2

Requisitos	r_{155}	r_{156}	r_{157}	r_{158}	r_{159}	r_{160}								
Tamanhos	8	13	5	2	13	8								

Tabela B.1: Tamanhos dos requisitos da Empresa Y.

Apêndice C

Representação Textual do Modelo Proposto

```
Section SectionNRPScrum {
   DeclarationSection Modelos {
      MathematicalProgram NRPScrum {
         Objective: satisfacao;
         Direction: maximize;
         Constraints: AllConstraints;
         Variables: AllVariables;
         Type: Automatic;
      }
      MathematicalProgram TempoMin {
         Objective: tempo;
         Direction: minimize;
         Constraints: AllConstraints;
         Variables: AllVariables;
         Type: Automatic;
      }
      MathematicalProgram CustoMin {
         Objective: custo;
         Direction: minimize;
         Constraints: AllConstraints;
```

```
        Variables: AllVariables;
        Type: Automatic;
    }
}
DeclarationSection Restricoes {
    Constraint C_Capacidade {
        IndexDomain: t;
        Definition: sum(i,tempoDev(i,t)*x(i,t))<=disp(t);
    }
    Constraint C_RequisitoUmTime {
        IndexDomain: (i);
        Definition: {
            if r2(i)<1 then
            sum(t,x(i,t))=r(i)
            else
            sum(t,x(i,t))=r2(i)
            endif
        }
    }
    Constraint C_RequisitoEscolhido {
        IndexDomain: i;
        Definition: {
            if r2(i)=1 then
            r(i)=r2(i)
            endif
        }
    }
    Constraint C_Precedencia {
        IndexDomain: (i,q) | (i,q) in S_Precedencia;
        Definition: r(q)>=r(i);
    }
    Constraint C_And {
        IndexDomain: (i,q) | (i,q) in S_And;
        Definition: {
```

```
            r(q)-r(i)=0
        }
    }
    Constraint C_Or {
        IndexDomain: (i,q) | (i,q) in S_Or;
        Definition: r(q)+r(i)=1;
    }
}
DeclarationSection Variaveis {
    Variable y {
        IndexDomain: j;
        Range: free;
        Definition: vc(j)*sum(i,vrc(i,j)*r(i)*(pvrc(i,j)));
    }
    Variable r {
        IndexDomain: i;
        Range: binary;
    }
    Variable x {
        IndexDomain: (i,t);
        Range: binary;
    }
    Variable satisfacao {
        Range: free;
        Definition: sum(j,y(j));
    }
    Variable tempo {
        Range: free;
        Definition: sum((i,t),tam(i)*horaPonto(t)*x(i,t));
    }
    Variable custo {
        Range: free;
        Definition: sum((i,t),tam(i)*horaPonto(t)*valorHora(t)*x(i,t));
    }
```

```
}
DeclarationSection Parametros {
    Parameter vrc {
        IndexDomain: (i,j);
        Range: {
            {0..10}
        }
    }
    Parameter vc {
        IndexDomain: j;
    }
    Parameter pvrc {
        IndexDomain: (i,j);
        Definition: vrc(i,j)/vtr(j);
    }
    Parameter vtr {
        IndexDomain: j;
        Definition: sum(i,vrc(i,j));
    }
    Parameter tamTime {
        IndexDomain: t;
        Range: integer;
    }
    Parameter capPPP {
        IndexDomain: t;
    }
    Parameter durSprint {
        InitialData: 40;
    }
    Parameter disp {
        IndexDomain: t;
        Definition: tamTime(t)*dursprint;
    }
    Parameter tam {
```

```
            IndexDomain: i;
        }
        Parameter horaPonto {
            IndexDomain: t;
            Definition: disp(t)/capPPP(t);
        }
        Parameter tempoDev {
            IndexDomain: (i,t);
            Definition: tam(i)*horaPonto(t);
        }
        Parameter valorHora {
            IndexDomain: t;
        }
    }
    DeclarationSection Conjuntos {
        Set S_Requisitos {
            Index: i, q;
        }
        Set S_Clientes {
            Index: j;
        }
        Set S_Times {
            Index: t;
        }
        Set S_Precedencia {
            SubsetOf: (S_Requisitos,S_Requisitos);
        }
        Set S_Or {
            SubsetOf: (S_Requisitos,S_Requisitos);
        }
        Set S_And {
            SubsetOf: (S_Requisitos,S_Requisitos);
        }
    }
```

```
    DeclarationSection Auxiliares {
        Variable satAux {
            Range: free;
            Unit: %;
            Definition: (satisfacao*10)/sum(j,vc(j));
        }
        Parameter r2 {
            IndexDomain: (i);
            Range: binary;
        }
        Parameter tempoPorTime {
            IndexDomain: t;
            Definition: sum(i,tempoDev(i,t)*x(i,t));
        }
        Parameter totalReq {
            IndexDomain: t;
            Definition: sum(i,x(i,t));
        }
    }
}


Procedure NRP {
    Body: {
        r2(i):=0;
        r(i):=0;
        Solve NRPScrum;

        if (NRPScrum.ProgramStatus <> 'Optimal') then

        empty NRPscrum,  r(i);
        endif;
        r2(i):=r(i);
    }
```

```
}


Procedure MenorTempo {
    Body: {

        Solve TempoMin;

        if (TempoMIn.ProgramStatus <> 'Optimal') then
        empty TempoMin, x(i,t);

        endif;
    }
}


Procedure MenorCusto {
    Body: {
        Solve CustoMin;

        if (CustoMin.ProgramStatus <> 'Optimal') then
        empty TempoMin, x(i,t);

        endif;
    }
}
```