UNIVERSIDADE ESTADUAL DO CEARÁ
CENTRO DE CIÊNCIA E TECNOLOGIA
MESTRADO ACADÊMICO EM CIÊNCIA DA COMPUTAÇÃO

PAULO ALBERTO MELO BARBOSA

UMA ABORDAGEM BASEADA EM OTIMIZAÇÃO PARA A PRIORIZAÇÃO DE REQUISITOS DE SOFTWARE CONSIDERANDO A ESTABILIDADE DO REQUISITO

FORTALEZA – CEARÁ
2013

PAULO ALBERTO MELO BARBOSA

UMA ABORDAGEM BASEADA EM OTIMIZAÇÃO PARA A PRIORIZAÇÃO DE REQUISITOS DE SOFTWARE CONSIDERANDO A ESTABILIDADE DO REQUISITO

Dissertação submetida à Comissão Examinadora do Programa de Pós-Graduação Acadêmica em Ciência da Computação da Universidade Estadual do Ceará, como requisito parcial para obtenção do grau de Mestre em Ciência da Computação.

Orientação:
Professor Dr. Jerffeson Teixeira de Souza

FORTALEZA - CEARÁ
2013

Dados Internacionais de Catalogação na Publicação

Universidade Estadual do Ceará

Biblioteca Central Prof. Antônio Martins Filho

Bibliotecário (a) Leila Cavalcante Sátiro – CRB-3 / 544

B238u Barbosa,Paulo Alberto Melo.

Uma abordagem baseada em otimização para a priorização de requisitos de software considerando a estabilidade do requisito/Paulo Alberto Melo Barbosa. — 2013.

CD-ROM 71f. : il. (algumas color.) ; 4 ¾ pol.

"CD-ROM contendo o arquivo no formato PDF do trabalho acadêmico, acondicionado em caixa de DVD Slin (19 x 14 cm x 7 mm)".

Dissertação (mestrado) – Universidade Estadual do Ceará, Centro de Ciências da Ciências e Tecnologia, Mestrado Acadêmico em Ciências da Computação, Fortaleza, 2013.

Área de Concentração: Ciências da Computação.

Orientação: Prof. Dr. Jerffeson Teixeira de Souza.

Estabilidade de requisitos. 2. Planejamento de release de software. 3. Otimização multiobjetivo. 4. Search-based Software Engineering.(SBSE). I. Título.

CDD: 001.6

UNIVERSIDADE ESTADUAL DO CEARA – UECE
PRO-REITORIA DE POS-GRADUACAO E PESQUISA - PRPGPq
CENTRO DE CIENCIAS E TECNOLOGIA – CCT
Mestrado Acadêmico em Ciência da Computação – MACC

Av. Paranjana, 1700. Bairro: Serrinha. Campus do Itapery Fone: (85) 3101 9776

ATA DA QUINQUAGÉSIMA OITAVA DEFESA PÚBLICA DE DISSERTAÇÃO DE MESTRADO

Aos vinte e sete dias do mês de setembro de dois mil e treze, no mini-auditório do prédio de Pesquisa e Pós-Graduação em Computação, do **Mestrado Acadêmico em Ciência da Computação – MACC,** realizou-se a sessão de defesa pública da dissertação de **PAULO ALBERTO MELO BARBOSA,** aluno regularmente matriculado no Mestrado Acadêmico em Ciência da Computação – MACC, tendo como título: **"UMA ABORDAGEM BASEADA EM OTIMIZAÇÃO PARA A PRIORIZAÇÃO DE REQUISITOS DE SOFTWARE CONSIDERANDO A ESTABILIDADE DO REQUISITO"**. A Banca Examinadora reuniu-se no horário de 15:00 às 17:00 horas, sendo constituída pelos professores Jerffeson Teixeira de Sousa, Ph. D, orientador e presidente da Banca, Dr. Paulo Henrique Mendes Maia, ambos da Universidade Estadual do Ceará – UECE e Tibérius de Oliveira e Bonates, Ph. D. da Universidade Federal Rural do Semi-Árido-UFERSA. Inicialmente o mestrando expôs seu trabalho e a seguir foi submetido à arguição pelos membros da Banca, dispondo cada membro de tempo para tal. Finalmente a Banca reuniu-se em separado e concluiu por considerar o mestrando APROVADO, por sua dissertação e sua defesa pública. Eu, Professor Jerffeson Teixeira de Sousa, orientador da dissertação e presidente da Banca, lavro a presente Ata que será assinada por mim e demais membros da Banca. Fortaleza, 27 de setembro de 2013.

Prof. Jerffeson Teixeira de Sousa, Ph. D,
Orientador - UECE

Prof. Tibérius de Oliveira e Bonates, Ph. D
Membro externo - UFERSA

Prof. Dr. Paulo Henrique Mendes Maia
UECE

À minha família e minha esposa pelo incentivo, compreensão, amor e por serem a maior razão da minha felicidade e existência.

AGRADECIMENTOS

Em primeiro lugar, gostaria de agradecer a Deus, pela oportunidade de desenvolver este trabalho e de encontrar pessoas nesta jornada que me fizeram crescer tanto intelectualmente como moralmente.

Ao Professor Dr. Jerffeson Teixeira de Souza que, além de seu papel de orientador, foi um amigo nos momentos difíceis que passei durante esta etapa da minha vida. Fico sem palavras para demonstrar o quanto sou grato pela sua paciência e, principalmente, pela sua dedicação à minha orientação.

À minha grande amiga MSc. Márcia Brasil pela presteza e paciência que teve, pela ajuda nas ideias e no texto da dissertação e por ter sempre acreditado em mim.

Aos professores Gustavo Lima e Mariela Cortés, pelos comentários feitos no meu exame de qualificação, os quais foram de grande valia para este trabalho.

A Universidade Estadual do Ceará e ao Departamento de Ciência da Computação, pela oportunidade de realizar meus estudos durante os anos de mestrado.

Aos professores do Mestrado Acadêmico em Ciência da Computação (MACC) no qual tive o prazer de assistir as aulas durante o mestrado.

Aos colaboradores do MACC, em especial ao Marcos pela força nos momentos mais difíceis dessa caminhada.

Aos colegas de trabalho do IFCE/Aracati Prof. Mário Wedney e Adriana Carvalho pela grande ajuda durante a revisão de tópicos deste trabalho.

Em especial, à minha família por estar sempre ao meu lado, pois sem ela, nada disso seria possível. Especialmente aos meus pais, Eliane e Paulo, quero agradecer pela formação que me proporcionaram, não medindo esforços para me dar a melhor educação possível.

Finalmente, agradeço a minha esposa Raquel Silveira, com a qual compartilhei angústias, felicidades e realizações. Agradeço ainda pela paciência, colaboração e compreensão.

Muito Grato!

“A vida é construída nos Sonhos e
concretizada no Amor."

Chico Xavier

RESUMO

A priorização de requisitos, segundo a sua estabilidade, contribui para um melhor cumprimento de prazos de entrega, pois garante que serão implementados primeiro os requisitos mais estáveis. A aplicação de técnicas de otimização mostra-se como um meio promissor para realizar essa priorização. O uso de metaheurísticas na Engenharia de Software deu origem a uma área de pesquisa chamada *Search-based Software Engineering* (SBSE) que objetiva automatizar a construção de soluções e tem a Engenharia de Requisitos como uma área de aplicação, onde o Planejamento de *Releases* é um exemplo dessa aplicação. Levando em consideração esse contexto, a Dissertação apresenta uma estratégia que vai buscar uma boa solução para a priorização de requisitos de acordo com a sua importância e estabilidade. Assim, se pretende dar suporte ao processo de decisão gerencial por meio das soluções encontradas. Para avaliar a abordagem proposta, foram realizados experimentos. Foram aplicadas algumas metaheurísticas para a busca de soluções de instâncias geradas para o problema. Os resultados foram examinados e corroborados com o uso de métricas de desempenho.

Palavras–Chave: Estabilidade de requisitos. Planejamento de *Release* de Software. Otimização Multiobjetivo. *Search-based Software Engineering* (SBSE).

ABSTRACT

The prioritization of requirements according to their stability contributes to a better fulfillment of delivery, since it ensures that the first requirements to be implemented are the most stable. The application of optimization techniques is shown as a promising means to accomplish this prioritization. The use of metaheuristics in software engineering has resulted in a research area called Search-based Software Engineering (SBSE) that aims to automate the construction of solutions and has the Requirements Engineering as an application area, where the Planning of Releases is an example of such application. Considering this context, the dissertation presents a strategy that will get the good solution for prioritizing requirements according to their importance and stability. Using the solutions found, we intend to give support to the process of management decision. To evaluate the proposed approach, experiments were performed. Some metaheuristics have been applied to the search for solutions to the problem instances generated. The results were reviewed and corroborated with the use of performance metrics.

Keywords: Stability requirements. Software Release Planning. Multiobjective Optimization. *Search-based Software Engineering* (SBSE).

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

AE	*Algoritmos Evolucionários*
AHP	*Analytical Hierarchy Process*
GECCO	*Genetic and Evolutionary Computation Conference*
GRASP	*Greedy Randomized Adaptive Search Procedure*
MOCell	*Multiobjective Cellular Genetic Algorithm*
NSGA-II	*Non-dominated Sorting Genetic Algorithm II*
SBES	Simpósio Brasileiro de Engenharia de Software
SBSE	*Search Based Software Engineering*
SSBSE	*Symposium on Search-Based Software Engineering*

SUMÁRIO

1 INTRODUÇÃO

É fundamental que a tarefa de selecionar e priorizar requisitos seja efetuada da forma mais eficiente possível. As mudanças de requisitos quase sempre são o principal fator para aumento de tempo e custo nos projetos de desenvolvimento de software. Portanto, selecionar e priorizar requisitos levando em consideração o seu grau de estabilidade pode elevar a eficácia de todo o processo de desenvolvimento do software. Neste capítulo apresentamos a estrutura do trabalho e os objetivos a serem alcançados levando em consideração a seleção e a priorização de requisitos estáveis.

1.1 Motivação

À medida que a capacidade de produzir software aumentou, também cresceu a complexidade dos sistemas de software requeridos. Por essa razão e pelo fato de muitas empresas não aplicarem as técnicas de engenharia de software de maneira eficaz, ainda existem problemas, tais como estimativas concretas de custo e tempo, durante o desenvolvimento de software (SOMMERVILE, 2004). Raramente, durante o desenvolvimento de um software, é dedicado tempo para coletar os dados, tais como indicadores de qualidade ou cumprimento de prazos, que estimam sobre o processo de desenvolvimento em si. Devido à pouca quantidade deste tipo de informação, as tentativas em estimar a duração/custo de produção de um software têm conduzido a resultados bastante insatisfatórios como atraso na entrega ou orçamento acima do estimado; além disso, a falta destas informações impede uma avaliação eficiente das técnicas e metodologias empregadas no desenvolvimento.

Uma abordagem estruturada para o desenvolvimento de software apresenta-se como uma boa relação custo benefício. Além disso, o desafio do fornecimento de software diz respeito a reduzir o tempo para o fornecimento do sistema, sem comprometer a qualidade (SOMMERVILE, 2004).

O modelo em *releases*, oriundos do desenvolvimento de software incremental, permite que os clientes recebam partes do software antecipadamente. Portanto, isso permite a atribuição de valores para os requisitos a serem desenvolvidos. Cada versão do sistema é um conjunto de características que formam um sistema completo que vai ter um valor para o cliente (RUHE, 2005).

Destaca-se que os métodos ágeis também utilizam entregas incrementais do software. Essas entregas, conhecidas como *sprints*, possuem um curto intervalo para serem

desenvolvidas e são trabalhadas por uma equipe através de um completo ciclo de desenvolvimento de software que termina quando o produto é apresentado ao cliente. Um conjunto de características que serão incluídas no incremento vem da negociação entre clientes e a equipe de desenvolvimento. Tais características são selecionadas a partir de um conjunto de requisitos de alto nível que cobrem todas as necessidades do cliente (SAGRADO, 2009).

Um problema enfrentado pelas empresas de desenvolvimento e manutenção de grandes e complexos sistemas de software desenvolvidos para grandes e diversificados clientes é o de determinar quais requisitos vão ser implementados na próxima *release* do software (BAGNALL, 2001).

> O planejamento de *releases* de software define os estágios de entrega do *software* cada um fornecendo um subconjunto das funcionalidades do sistema. O planejamento de *releases* envolve todos os aspectos e decisões relacionados à priorização e alocação de requisitos em seqüências de releases do software. Assim, realizar um planejamento adequado é uma atividade não somente importante como intrinsecamente complexa, que normalmente é realizada informalmente através do conhecimento e experiências anteriores de gerentes (BRASIL, 2010, p. 15).

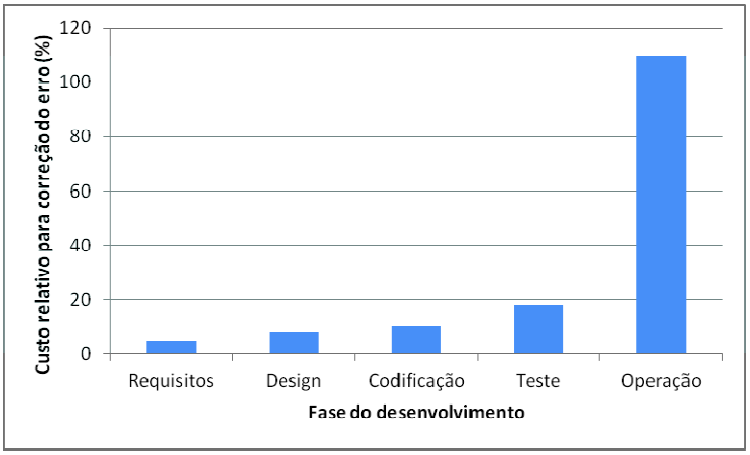
Requisitos são fundamentais no processo de desenvolvimento do software. São eles que proporcionam as bases para estimar custos e esforço, além de permitir a elaboração das estimativas do desenvolvimento e especificações de testes. Embora um conjunto inicial possa ser bem documentado, requisitos mudarão durante o ciclo de desenvolvimento do software. Essas constantes mudanças (adições, exclusões e modificações) nos requisitos durante o ciclo de desenvolvimento impactarão nas estimativas de custos, tempo e qualidade do produto resultante (ZOWGHI, 2002).

Idealmente, os requisitos aprovados pelo cliente devem ficar estabilizados ou com o mínimo de mudanças. Mudanças de requisitos devem declinar de 3% no início do projeto para 1% na fase de codificação e, idealmente, para 0% durante os testes. No entanto, os requisitos sempre mudam e isso pode ter um efeito negativo durante os últimos estágios do desenvolvimento de software. Requisitos que mudam durante a fase de codificação tendem a aumentar a quantidade erros encontrados na fase de testes (CAPERS JONES, 1997).

> Estudos realizados demonstraram que as taxas de erros associados com as novas características adicionadas durante a fase de codificação é de aproximadamente 50% maior do que aqueles dos erros associados com os requisitos originais (CAPERS JONES, 1997, p. 01).

O resultado inevitável de erros de requisito é o tempo dispendioso de retrabalho. O retrabalho pode consumir de 30 a 40 por cento do esforço total exercido sobre um projeto de desenvolvimento de software. A **Erro! Fonte de referência não encontrada.** demonstra que pode custar até 110 vezes mais corrigir erros decorrentes da mudança de requisitos do que se estes fossem definidos ainda na fase de elicitação de requisitos (SMITH, 2006).

FIGURA 1 – Custo para corrigir um defeito dependendo de quando é descoberto



Fonte: Smith, 2006

Tais mudanças nos requisitos originais são causadas pela má elicitação, evolutivas necessidades dos clientes, mudanças tecnológicas, mudanças no ambiente de negócio ou política da empresa (M. CHRISTEL, 1992).

Desse modo, o Planejamento de *Release* de software deve considerar uma nova variável: a volatilidade do requisito. Requisitos voláteis são considerados como um fator que pode causar grandes dificuldades durante o desenvolvimento do software (B. CURTIS, 1988). No entanto, se as implementações de tais requisitos forem organizadas de modo que os requisitos menos voláteis sejam os primeiros a serem implementados e os mais voláteis, ou seja, aqueles que ainda dependem de ajustes entre desenvolvedores e clientes, sejam implementados por último, poderemos ter um ganho de produtividade e satisfação de empresas de desenvolvimento de software e clientes.

Uma comparação entre as diversas técnicas de planejamento de *Releases* de software apresentadas por Karlsson (1998) evidencia que quanto mais complexo o software, maior o tempo para chegarmos a um resultado satisfatório em termos de planejamento de *Releases*.

Durante o processo de elaboração do planejamento de *releases* de software podemos encontrar várias restrições, tais como: orçamento do projeto, precedência entre requisitos (RUHE, 2005), limitações de tempo e ainda a disponibilidade de desenvolvedores.

A Otimização em Engenharia de Software é determinada como um meio de resolver problemas da Engenharia de Software através da utilização de técnicas de otimização matemática. Harman et al (2001) publicaram um trabalho em que apresentou a *SBSE (Search Based Software Engineering)*. O trabalho enfatiza que essa sub-área da engenharia de software procura reformular os problemas da engenharia de software como um problema de busca, sendo adequada a aplicação de técnicas de busca metaheurísticas, como por exemplo, algoritmos genéticos, entre outros.

Essa abordagem foi empregada na formulação e resolução de vários problemas, tais como a análise de requisitos, a otimização de código e a geração e seleção de casos de testes, incluindo suas fases (FREITAS, 2009). Entre outras, a área de testes de software mostrou-se bastante promissora para a aplicação da SBSE. Destacam-se, entre outros, a priorização e seleção de requisitos.

O contexto abordado neste trabalho será o Planejamento de *Releases* de Software, que indica uma abordagem semi-automatizada para ser usada por gerentes de software como meio para obter um planejamento eficaz levando em consideração a seleção de requisitos mais prioritários, tendo como critério a sua estabilidade.

A SBSE complementa as técnicas convencionais da Engenharia de Software. Os avanços conseguidos nas últimas décadas em teoria e prática nesta disciplina continuam válidos. No caso, o que acontece é que alguns problemas que antes não eram completamente resolvidos ou sequer tratados pelas metodologias e métodos convencionais passam a ser estudados e solucionados (Freitas et al. 2009).

1.2 Objetivos

A partir da motivação descrita, este trabalho tem como objetivo principal apresentar uma abordagem baseada em otimização multiobjetivo para o problema da seleção e priorização de requisitos de software, considerando as características inerentes aos projetos reais. Entre essas características, podemos citar: a) estabilidade do requisito, b) custos para implementação do requisito, c) precedência entre os requisitos, d) importância dos *steakholders* para a empresa e as suas preferências em relação aos requisitos. Como objetivos específicos, pretende-se:

- Definir a modelagem matemática, as dependências e restrições das variáveis do problema;
- Empregar, buscando solução do problema modelado, metaheurísticas multiobjetivos e mostrar a eficácia dessas metaheurísticas;
- Projetar, implementar e demonstrar resultados de experimentos alcançados em determinadas instâncias para demonstrar a eficiência e a viabilidade da abordagem proposta;
- Comparar a eficácia das metaheurísticas na solução do problema através de medidas de desempenho e comparar esse desempenho das metaheurísticas com o resultado da implementação de uma solução de um algoritmo aleatório.

1.3 Estrutura do trabalho

Esse trabalho tem seu desenvolvimento conforme segue.

Capítulo 2 – Fundamentação Teórica: Apresenta as definições da engenharia de requisitos, otimização matemática mono-objetivo e multiobjetivo, além de explanar sobre as metaheurísticas NSGA-II e MOCell. Ao final, é demonstrada a atuação e característica da *Search-based Software Engineering (SBSE)*.

Capítulo 3 – Trabalhos Relacionados: Expõe os trabalhos relacionados à seleção, alocação e priorização de requisitos relacionando estes a SBSE e enfatizando as principais características;

Capítulo 4 – Modelagem do problema de seleção e priorização de requisitos de software: Apresenta e detalha a abordagem proposta, assim como a modelagem matemática do problema. Por fim, mostra os pontos relevantes levados em consideração para a modelagem do problema;

Capítulo 5 – Avaliação: Demonstra como o problema modelado pode ser solucionado através das metaheurísticas multiobjetivos. Por fim, apresenta resultados de experimentos a fim de ratificar a eficiência e aplicabilidade da abordagem proposta;

Capítulo 6 – Conclusão: Apresenta as considerações finais desse trabalho, enfatizando as principais contribuições desta pesquisa, bem como suas limitações e demonstrando as possibilidades de trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo, será apresentada uma fundamentação teórica acerca dos campos de estudo relacionados a esse trabalho. Introdutoriamente, são demonstrados os conceitos de engenharia de requisitos, necessários para o entendimento da abordagem proposta. Logo após, são vistos os conceitos de otimização matemática mono-objetivo e multiobjetivo. Por fim, o capítulo aborda a *Search-Based Software Engineering (SBSE),* área na qual este trabalho está inserido.

2.1 Engenharia de requisitos

Os problemas que os engenheiros de software têm para solucionar são, muitas vezes, imensamente complexos. Compreender a natureza dos problemas pode ser muito difícil, especialmente se o sistema for novo. Consequentemente, é difícil estabelecer com exatidão o que o sistema deve fazer. As descrições das funções e das restrições são os requisitos para o sistema; e o processo de descobrir, analisar, documentar e verificar essas funções e restrições é chamado de engenharia de requisitos (SOMMERVILLE, 2003).

A engenharia de requisitos tem sido reconhecida como uma das mais importantes fases do processo de engenharia de software. Este reconhecimento decorre da descoberta que a maior parte dos problemas, e geralmente os mais dispendiosos e de maior impacto negativo no desenvolvimento de software, são originados nas etapas iniciais do desenvolvimento. Estas etapas constituem o processo de engenharia de requisitos, no qual as principais atividades podem ser definidas como: elicitação, análise, negociação, especificação, gerenciamento e validação de requisitos (KOTONYA et al. 1997).

A engenharia de requisitos consiste em um processo sistemático de desenvolvimento de requisitos através de um processo iterativo de análise do problema, documentação das observações resultantes e verificação acerca da precisão de entendimento (KANG, 1992).

O principal objetivo da Engenharia de Requisitos é criar e manter documentos de requisitos de sistemas. O processo de engenharia de requisitos, como um todo, contém quatro grandes sub-processos que são: a) utilidade do software ao negócio (viabilidade), b) descoberta de requisitos (elicitação), c) conversão de tais requisitos em um formato padrão (especificação) e d) descoberta de que tais requisitos realmente definem o sistema tal como o usuário deseja (validação).

Segundo Sommerville (2004), no geral, a engenharia de requisitos possui as seguintes atividades: a) análise de requisitos do software, b) projeto, c) codificação, d) testes e e) suporte. De acordo com Pressman (2007), nem sempre é fácil identificar os requisitos de software, até mesmo devido à natureza abstrata própria de um projeto de software. Porém, se esse processo não for bem feito, a documentação do projeto ficará inviabilizada comprometendo todas as fases do projeto.

A Engenharia de Requisitos é a fase do desenvolvimento de sistemas de software responsável pela identificação dos objetivos do sistema pretendido, pela operacionalização de tais objetivos em serviços e restrições e pela atribuição da responsabilidade dos requisitos resultantes para agentes como: humanos, hardware, e software (LAMSWEERDE, 2000).

Boehm (1989) define Engenharia de Requisitos como uma atividade que objetiva desenvolver uma especificação completa, consistente, não ambígua e correta dos requisitos que sirva, inclusive, de base para um acordo entre as partes envolvidas no processo de desenvolvimento do software. Nesse sentido, a Engenharia de Requisitos caracteriza-se como um processo que requer um grande envolvimento entre cliente e desenvolvedores, evitando decisões de projeto durante a definição dos requisitos. Essa definição parece separar a Engenharia de Requisitos de outras preocupações do desenvolvimento de software (HOFFMANN, 1993).

A especificação de requisitos é uma etapa essencial do processo de desenvolvimento de software, que compreende uma definição completa do comportamento externo do sistema de software, tanto em termos de requisitos funcionais quanto de requisitos não funcionais (CARPENA, 1998). Vários estudos identificaram que a definição inadequada de requisitos é responsável por uma parte significativa dos erros detectados ao longo do processo de desenvolvimento de sistemas, principalmente no caso de sistemas dedicados, de tempo real e críticos (LUTZ, 1993). Outros estudos indicam que a eliminação de erros de especificação torna-se cada vez mais difícil e dispendiosa à medida que o sistema avança para etapas posteriores do seu ciclo de vida, como projeto e implementação (DAVIS, 1993).

A Engenharia de Requisitos está relacionada com a assimilação de metas a serem atingidas pelo sistema a ser implementado, assim como a operacionalização de tais metas em serviços e restrições (ZAVE, 1997).

A definição de requisitos é necessária para o entendimento da nossa proposta. Várias definições têm sido usadas para requisitos de software. Elas representam perspectivas diferentes de detalhes e precisão. O IEEE (IEEE, 1997) define requisito como sendo:

- Uma condição ou uma capacidade de que o usuário necessita para solucionar um problema ou alcançar um objetivo ou;
- Uma condição ou uma capacidade que deve ser alcançada ou possuída por um sistema ou componente do sistema para satisfazer um contrato, um padrão, uma especificação ou outros documentos impostos formalmente ou;
- Uma representação documentada de uma condição ou capacidade.

Essa definição do IEEE abrange a visão do usuário sobre requisitos (comportamento externo do sistema), a visão dos desenvolvedores e o importante conceito de quais requisitos devem ser documentados.

Outro conceito indica que um requisito é uma necessidade do usuário ou uma característica, função ou atributo necessário do sistema que pode ser percebido de uma posição externa daquele sistema (DAVIS, 1993). Essa definição enfatiza o que o sistema deve conter.

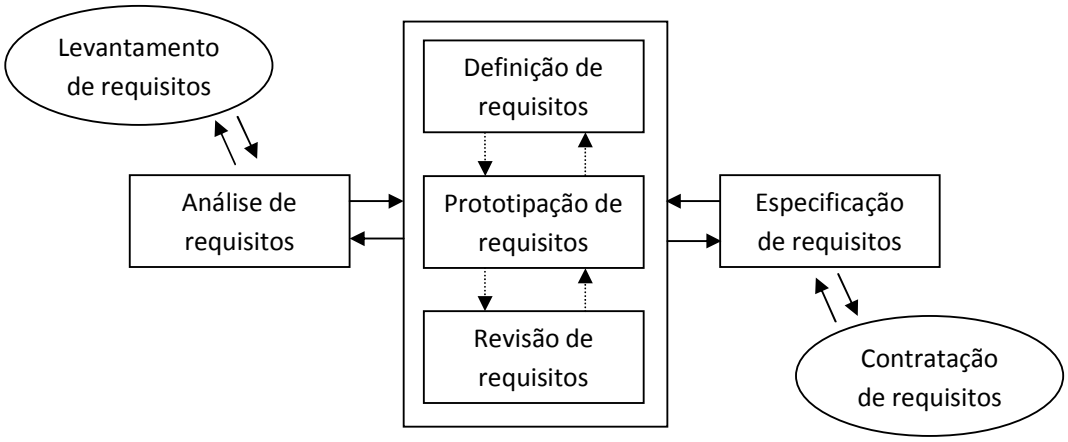
Segundo Kotonya et al. (1997), um requisito pode descrever:

- Uma facilidade no nível do usuário;
- Uma propriedade muito geral do sistema;
- Uma restrição específica no sistema;
- Uma restrição no desenvolvimento do sistema;

Uma definição bastante simples para requisitos é dada por Macaulay (1996). Segundo este autor, requisito é simplesmente algo que o cliente necessita. Requisitos são fenômenos ou propriedades do domínio da aplicação que devem ser executados, normalmente expressos em linguagem natural, diagrama informal ou em notação apropriada ao entendimento do cliente e da equipe de desenvolvimento (JACKSON, 1995).

A FIGURA 2 é um exemplo de um processo comum de engenharia de requisitos.

FIGURA 2 – Um processo de engenharia de requisitos



Fonte: Tsui; Karam (2013)

Durante todo o processo de desenvolvimento e implementação de um software, são muitos os problemas que podem acontecer relacionados à base da construção do software, ou seja, os requisitos desse software podem gerar impactos negativos no final da construção. Requisitos incompletos ou defeituosos podem causar problemas no produto final.

A elicitação de requisitos é geralmente executada por uma metodologia ou uma série de técnicas, com o objetivo comum de dar aos analistas o entendimento necessário sobre um problema (MAIDEN, 1996). Embora alguns analistas acreditem que uma única metodologia ou técnica possa ser aplicada em todas as situações, vários pesquisadores afirmam que não existe uma única, capaz de suportar a resolução de todas as dificuldades das várias situações de um problema (GALETTI, 2003).

2.2 Otimização matemática

A otimização matemática, comumente chamada de programação matemática, possui uma história relativamente curta. Essencialmente, possui a característica de procurar minimizantes, ou maximizantes, de certa função num certo conjunto, geralmente definida por equações e inequações algébricas (SOARES, 2005).

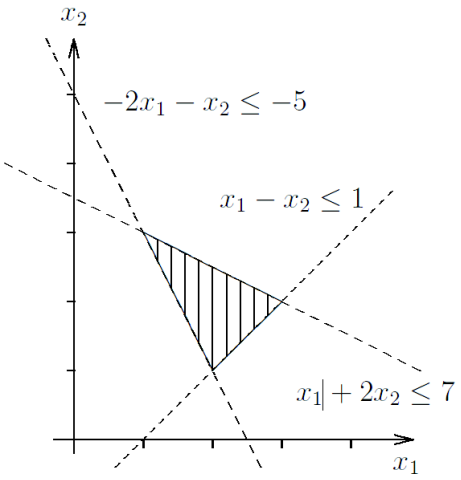
FIGURA 3 – Representação matemática sujeita a restrições

$$\begin{aligned} &maximizar \;\; x_1 + x_2 \\ &sujeito\ a\text{:} \quad x_1 - x_2 \leq 1, \\ &\qquad -2x_1 - x_2 \leq -5, \\ &\qquad x_1 + 2x_2 \leq 7. \end{aligned}$$

Fonte: Soares (2005)

Na FIGURA 3, $x_1 + x_2$ representa a função objetivo a ser otimizada, nesse caso maximizada, respeitando as restrições $x_1 - x_2 \leq 1$, $-2x_1 - x_2 \leq -5$ e $x_1 + 2x_2 \leq 7$. A FIGURA 4 fornece uma representação geométrica para a modelagem apresentada na FIGURA 3.

FIGURA 4 – Esboço da região admissível



Fonte: Soares (2005)

De acordo com suas características, podemos classificar os problemas de otimização. A classificação que interessa ao contexto desta pesquisa é a classificação em que os problemas são divididos de acordo com a quantidade de funções objetivos. Portanto, podemos definir a função objetivo como o ponto formado pelas variáveis de projeto que extremizam uma função e satisfazem as restrições.

2.2.1 Otimização mono-objetivo

Os problemas onde exista apenas uma função objetivo caracterizam a otimização mono-objetivo. As técnicas de otimização mono-objetivo possuem algumas limitações quando aplicadas a problemas de otimização multiobjetivo, tais como a composição de funções e a repetição do processo de busca (JUNIOR, 2005).

Na seção 2.2.3.1 é apresentado um algoritmo genético mono-objetivo.

2.2.2 Otimização multiobjetivo

Fonseca e Fleming (1995) afirmam que a otimização multiobjetivo é diferente da otimização mono-objetivo pelo fato de raramente aceitar uma simples solução. Portanto, a

solução é quase sempre composta por uma família de soluções localizadas na frente de Pareto e devem ser consideradas equivalentes.

À medida que se aumenta o número de objetivos a serem otimizados, o problema de buscar uma solução na frente de Pareto torna-se complexo de forma rápida e progressiva (PEREIRA, 2004).

A otimização multiobjetivo aborda os problemas de otimização que possuem várias funções objetivos a serem simultaneamente maximizados ou minimizados. A simples decisão da compra de um computador pode ser considerada um problema multiobjetivo, por buscarmos o melhor desempenho pelo menor custo. Habitualmente, o computador com melhor desempenho vai custar financeiramente mais para o comprador. Por outro lado, o computador de menor desempenho irá custar menos. Neste típico problema, os objetivos (ou decisões conflitantes) precisam ser ponderados pelo comprador.

Para este tipo de problema, existe um conjunto de soluções que representa um compromisso entre os objetivos. Diversas técnicas de otimização tradicionais têm sido propostas na literatura (COELHO et al., 2002; DEB, 2001). Ainda que essas técnicas não garantam encontrar o conjunto de soluções de boa qualidade ou Pareto-ótimas, gastam muito recurso. Devido a esta limitação, uma série de técnicas heurísticas e estocásticas têm sido desenvolvidas dentre as quais têm-se destacado os Algoritmos Evolucionários (AE) (COELHO et al., 2002; DEB, 2001).

Segundo Castro (2001), um problema multiobjetivo pode ser descrito com um vetor y, com n objetivos, que dependem de um vetor x de m variáveis independentes. A formulação de um problema multiobjetivo é apresentada na FIGURA 4, de acordo com Castro (2001).

min/max:	$y = (f_1(x), f_2(x), \ldots, f_n(x))$
sujeito a:	$x = (x_1, x_2, \ldots, x_m) \in \boldsymbol{X}$
	$y = (y_1, y_2, \ldots, y_n) \in \boldsymbol{Y}$

Onde:

- y: vetor de funções objetivos;
- $f_i(\cdot), 1 \leq i \leq n$: i-ésima função objetivo;
- $\boldsymbol{Y}$: espaço das funções objetivos;

- x: vetor de variáveis de decisão;
- $x_i, 1 \leq i \leq m$: i-ésima variável de decisão;
- $\boldsymbol{X}$: espaço das variáveis de decisão.

As variáveis de decisão que não podem melhorar o valor de qualquer objetivo são consideradas como um conjunto de soluções, conforme apresentado na expressão acima. Esse conjunto de decisão também é conhecido como frente de Pareto.

Segundo ainda Castro (2001), podemos descrever as soluções ótimas de Pareto como um problema de minimização, onde as funções objetivos devem ser minimizadas, de acordo com as definições a seguir:

- Um ponto $A = (a_1, a_2, \dots, a_m) \in u$ domina outro ponto $B = (b_1, b_2, \dots, b_m)$ se: $\forall\, i \in \{1, 2, \dots, n\} : f_i(A) \leq f_i(B) e \exists\, j \in \{1, 2, \dots, n\}: f_j(A) < f_j(B)$ (considerando que as funções objetivos devem ser minimizadas);
- Frente de Pareto é um subconjunto F de u tal que $\forall A \in F$. A não pode ser dominado por nenhum outro ponto.

O principal objetivo da otimização multiobjetivo é encontrar o conjunto de soluções não-dominadas, ou frente de Pareto. Este conjunto de soluções pode ser aproveitado por um tomador de decisões para facilitar uma escolha da solução que seja a mais adequada para o problema proposto.

Segundo Horn (1997), na solução de problemas multiobjetivos, dois problemas podem ser identificados: a) a busca de soluções, que se refere ao processo de otimização no qual o conjunto de soluções viáveis deve ser guiado para o encontro do conjunto de soluções na frente de Pareto; e b) a seleção de um critério apropriado para a escolha de uma solução na frente de Pareto. Este critério será utilizado pelo responsável, ou decisor, para a tomada de decisão, ou seja, ele poderá ponderar entre as diferentes soluções conflitantes.

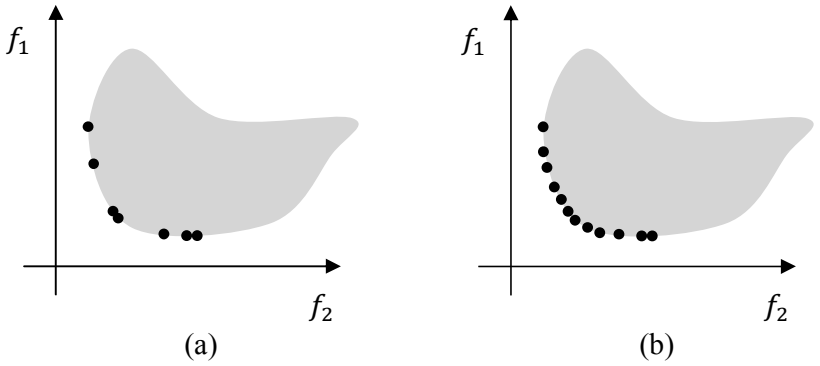
Deb (2001) indica duas metas importantes a serem alcançadas em otimização multiobjetivo: a) Encontrar um conjunto de soluções que esteja numa região próxima a frente de Pareto e b) encontrar um conjunto de soluções com a maior diversidade possível.

Soluções muito distantes da frente de Pareto não são ideais. Encontrar a maior diversidade possível dentro das soluções admissíveis é uma meta específica para a otimização multiobjetivo.

A FIGURA 5a mostra a frente de Pareto com baixa diversidade, ou seja, soluções em apenas algumas regiões.

Já FIGURA 5b apresenta uma distribuição quase uniforme das soluções na frente de Pareto. É indispensável garantir um maior número de soluções possíveis na frente de Pareto uma vez que esses conjuntos de soluções estão entre os objetivos admissíveis desejados.

FIGURA 5 – Distribuições de soluções da frente de Pareto



Fonte: Elaborado pelo autor (2013)

A frente de Pareto real, ou ótima, é formada pela criação de uma frente de pontos não dominados decorrentes da junção de todas as execuções de um mesmo problema. O momento em que usamos a frente de Pareto real durante esse trabalho pode ser identificado na FIGURA 17c.

São apresentadas a seguir metaheurísticas baseadas em algoritmos genéticos evolucionários para resolução de problemas de otimização, as quais foram usadas na resolução da abordagem proposta neste trabalho.

2.2.3 Metaheurísticas

Segundo Becceneri (2007), metaheurística é uma ferramenta algorítmica geral, que pode ser aplicada a diferentes problemas de otimização, com modificações relativamente pequenas, a fim de torná-las adaptáveis a um problema específico. Assim, podemos considerar metaheurísticas como procedimentos heurísticos que possuem estratégias genéricas de fuga de ótimos locais. Assim, as chances de obtenção de melhores soluções são maiores, quando comparadas a heurísticas simples. Tais métodos exploram melhor o espaço de busca, buscando, assim, soluções em outras regiões.

As metaheurísticas podem facilmente incorporar novas restrições e explorar regiões de um conjunto na tentativa de superar a otimalidade local. Embora não possam garantir a otimalidade global, elas são capazes de identificar inúmeros pontos de ótimos locais.

O prefixo meta é utilizado para descrever uma heurística que está sob outra heurística, compondo assim outro nível "heurístico". De maneira geral, uma metaheurística constitui uma estrutura mais genérica baseada em princípios ou conceitos, sobreposta a uma heurística específica do problema (ARROYO, 2002)

Algoritmos evolutivos são metaheurísticas que simulam seleção natural como meio de buscar soluções de problemas (COELLO, 2005). Devido a sua abordagem baseada em população, esse processo permite encontrar várias soluções ótimas em apenas uma única execução (DEB, 2009).

FIGURA 6 – Estrutura básica de um algoritmo evolutivo

1. **Procedimento** *AlgoritmoEvolutivo*
2. $t \leftarrow 0$;
3. **Inialize** $P(t)$;{População Inicial} **Avalie** $P(t)$;
4. **enquanto** *CriteriodeParada* **faça**
5. $t \leftarrow (t + 1)$
6. **Selecione** $P(t)$ *de* $P(t - 1)$;
7. **Altere** $P(t)$;
8. **Avalie** $P(t)$;
9. **fim**
10. **Fim**

Fonte: Mccallum (1977)

Segundo o fluxo apresentado na FIGURA 6, um algoritmo evolutivo necessita receber uma população de soluções e cada solução precisa ser avaliada por uma função que mede sua qualidade, adaptação, "sobrevivência", etc. Esta função é denominada função *fitness*, função de aptidão, ou função de avaliação.

Pelo fato de os algoritmos evolutivos possuírem bom desempenho, exige-se que seja empregada a estratégia do elitismo, ou seja, os melhores indivíduos de uma geração são preservados e passados para a geração seguinte.

2.2.3.1 Algoritmo genético mono-objetivo

Fundamentados na teoria da evolução natural das espécies, os algoritmos genéticos partem do princípio de que os melhores indivíduos devem sobreviver e gerar descendentes que preservem as suas características genéticas (GOLDBERG, 1989).

A partir das soluções iniciais recebidas e enquanto o critério de parada não é atendido, o algoritmo genético mono-objetivo realiza procedimentos de avaliação, classificação, elitização e aplicação dos operadores de seleção, recombinação e mutação.

A função *fitness* fornece à aptidão de um indivíduo em relação aos demais. A elitização corresponde a seleção de *S* soluções da geração atual que possua o melhor valor de *fitness*. Esses indivíduos selecionados farão parte de uma próxima geração de soluções (SANTOS, 2005).

Algoritmos genéticos são aplicados em problemas de busca onde, dado um conjunto de indivíduos, espera-se encontrar aquele que mais atenda a certas condições especificadas (BARBOSA, 1996).

Genericamente, podemos representar um algoritmo genético conforme segue (CASTRO, 2001):

1. **Procedimento** *Algoritmo genético*
2. **Inicialize** a população (geração i = 1)
3. **Repita** (evolução)
4. **Selecione** indivíduos para reprodução
5. **Aplique** operadores de recombinação e/ou mutação
6. **Avalie** indivíduos gerados na população
7. **Selecione** indivíduos para sobreviver (geração i = i + 1)
8. **Até** objetivo final ou máximo de gerações
9. **Fim**

2.2.3.2 NSGA-II

Proposto por Srinivas e Deb (1994), o *Non-dominated Sorting Genetic Algorithm* (NSGA) tem a característica original de ordenar as soluções através da camada em que as mesmas se encontram na frente de Pareto. Dessa forma, soluções não dominadas são mantidas de uma geração para outra e também soluções dominadas podem permanecer. De acordo com Deb et al. (2002), o NSGA possui três pontos críticos, que foram aperfeiçoados na nova versão: a) complexidade computacional na ordenação de soluções não-dominadas; b) abordagem não-elitista e c) necessidade de um parâmetro de compartilhamento (MOURÃO, 2009).

Para reparar o primeiro problema, foi implementado um novo método de ordenação de soluções com complexidade $O(M\ N^2)$, onde o M representa o número de objetivos e N o tamanho da população, aperfeiçoando o anterior, que possui complexidade $O(M\ N^3)$.

Para suprir o segundo ponto crítico do NSGA, foi implementada uma abordagem elitista para o NSGA-II, de forma que as soluções são distribuídas em *frentes*, com a primeira *frente* sendo das soluções não-dominadas, e a segunda *frente* contendo as soluções dominadas por pelo menos uma solução contida na primeira *frente*, e as demais *frentes* são definidas de maneira análoga. Cada solução dominada possui uma probabilidade de sobreviver de acordo com a quantidade de soluções que a domina e também de acordo com a *frente* dessas soluções que a domina.

Por fim, para corrigir o terceiro ponto crítico do NSGA, foi implementado no NSGA-II o uso do valor da *crowding distance* (distância de agrupamento) onde não há necessidade de um parâmetro de compartilhamento, utilizado para manter a diversidade na população, necessário em seu antecessor.

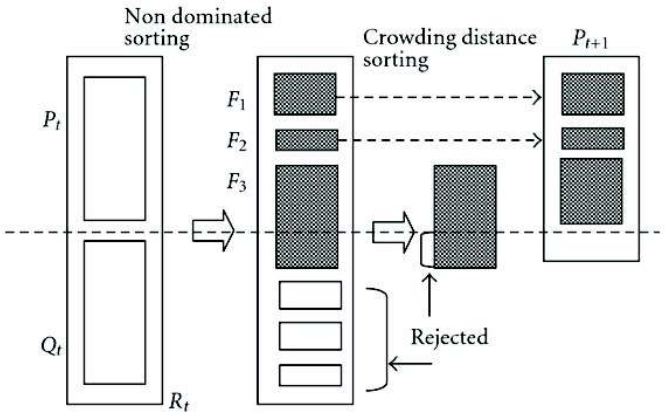
O funcionamento do algoritmo NSGA-II inicia-se com uma população P_0 e a essa população é aplicada a técnica *Fast non Dominated Sort*, que busca soluções próximas a frente de Pareto. Cada uma das soluções recebe um *rank* conforme o seu nível de não-dominância. Esse *rank* define o índice da fronteira (*front*) à qual a solução pertence. Através dos operadores comuns dos algoritmos genéticos, é constituída uma nova população Q_0. De maneira geral, na geração t, temos P_t e Q_t, sendo essa última gerada através de operadores de seleção, recombinação e mutação.

A próxima etapa é gerar uma população $R_t = P_t \cup Q_t$. Em seguida, o algoritmo *Fast non Dominated Sort* recebe a população R_t e são geradas as fronteiras. As soluções contidas em F_1 são as não-dominadas e formam o conjunto de soluções candidatas à frente de Pareto. Após determinar os valores das frentes, os valores da *crowding* são calculados para a última frente, conforme segue. Seja a solução n_{sol}, sendo n_{pop} o número de indivíduos do NSGA-II, pertencente a uma determinada *frente* n_{rank}. Se a solução $n_{sol} + 1$ pertencer à *frente* $n_{rank} + 1$, então o cálculo da *crowding distance* não se faz necessário, pois todas as soluções até n_{pop} serão aceitas para a geração seguinte. Mas se a solução $n_{sol} + 1$ pertencer à mesma *frente* n_{rank}, então é efetuado o cálculo da *crowding distance* em cada solução da

frente n_{rank} e as soluções que passarão para a próxima geração serão as soluções até a *frente* n_{rank}, sendo que nesta frente serão escolhidas as soluções que tiverem os maiores valores de *crowding distance*.

Em relação ao elitismo, inicialmente são preferidas as soluções que pertencem a frentes menores e, em seguida, é efetuado o cálculo da *crowding distance* na *frente* em que isso se faz necessário, e são preferidas as soluções com maiores valores de *crowding distance* (MOURÃO, 2009). A FIGURA 7 demonstra este procedimento.

FIGURA 7 – Procedimento NSGA-II



Fonte: Deb et al. (2002)

2.2.3.3 MOCell

Nessa seção apresentamos o MOCell (NEBRO, 2009), um algoritmo multiobjetivo baseado no modelo de GA (Algoritmo Genético). No caso multiobjetivo, esse algoritmo possui uma frente de Pareto (definição em 2.2.2). Para realizar inserções na frente de Pareto com o objetivo de obter um conjunto de soluções diversificadas é utilizado um estimador de distância *crowding*, conforme visto no item 2.2.3.2. Esse método também é utilizado para eliminar as soluções da frente de Pareto quando esta frente está muito densa.

De acordo com a FIGURA 8, o MOCell inicia criando uma frente de Pareto vazia. Os indivíduos se situam num plano de duas dimensões cartesianas e vai aplicando-se sucessivamente o ciclo reprodutor até ser alcançada a condição de parada. Assim, para cada indivíduo, o algoritmo consiste em selecionar pais na vizinhança, recombiná-los para obter um descendente, mutá-lo, avaliar o indivíduo resultante e inseri-lo tanto em uma população

auxiliar (caso essa solução não seja dominada) como na frente de Pareto. Finalmente, depois de cada geração, substitui-se a população antiga pela população auxiliar e aciona-se um processo de *feedback* para substituir um número fixo de indivíduos da população escolhidos aleatoriamente.

FIGURA 8 – Pseudocódigo que ilustra o MOCell

```
1. Proc Step_Up(mocell)
2. Pareto_front = Create_Front()
3. while !TerminationCondition() do
4.    for individual ← 1 to mocell.popSize do
5.          n_list ← Get_Neighborhood(mocell,position(individual));
6.          parents ← Selection(n_list);
7.          offspring ← Recombination(mocell.Pc,parents);
8.          offspring ← Mutation(mocell.Pm,offspring);
9.          Evaluate_Fitness(offspring);
10.         Insert(position(individual),offspring,mocell,aux_pop);
11.         Insert_Pareto_Front(individual);
12.   end for
13.   mocell.pop ← aux_pop;
14.   mocell.pop ← Feedback(mocell,ParetoFront);
15. end while
16. end proc Steps_Up;
```

Fonte: Nebro (2009)

2.2.4 Algoritmo aleatório

Caracterizam-se por utilizar critérios de aleatoriedade como meio para buscar soluções. Possui um gerador de números randômicos que, num espaço definido, escolhe aleatoriamente um caractere numérico. Esse tipo de algoritmo pode ser aplicado em criptografia, programação distribuída, geométrica computacional e em estratégias de busca de uma maneira geral (FIGUEIREDO, 2007).

Possui a vantagem de ser mais rápido e mais simples de se implementar. Como exemplo de uso dessa técnica, temos os algoritmos de Monte Carlo e Las Vegas. Nesse trabalho, foi implementado um algoritmo aleatório para comparação de resultados obtidos em relação às metaheurísticas NSGA-II e MOCell.

2.3 *Search-based Software Engineering*

Os problemas da engenharia de software frequentemente envolvem restrições conflitantes, informações ambíguas e imprecisas dentro de um grande conjunto de escolhas ou decisões. Resolver esses problemas é uma tarefa complexa considerando que não existe uma

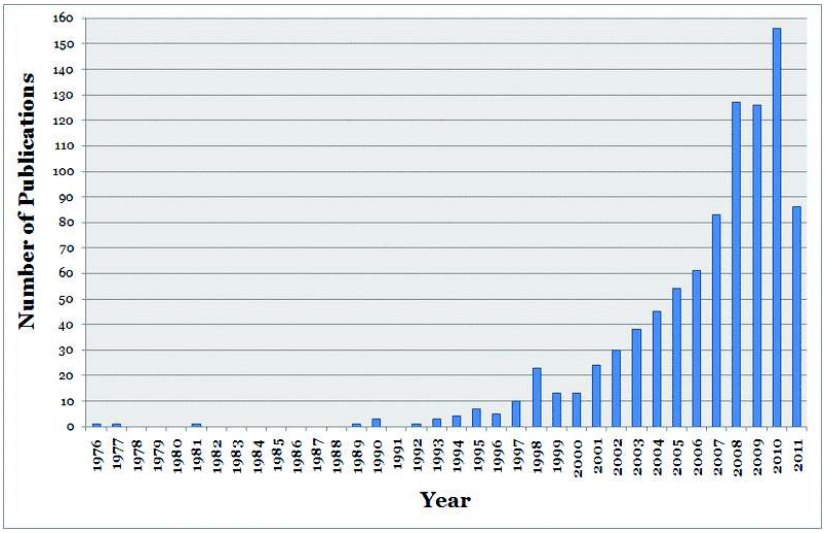
solução ótima (VERGILIO, 2011). Se levarmos em consideração um conjunto de requisitos para compor vários releases, as sequências de requisitos de entrada para esse problema podem ser excessivamente numerosas. Além disso, as composições das soluções devem obedecer a restrições como satisfação de clientes, tempo, custo, entre outros.

A solução desses problemas tem sido abordada por um novo campo de pesquisa chamado de *Search-based Software Engineering* (SBSE), que é uma área de pesquisa que aplica algoritmos de otimização baseados em busca para automatizar a construção de soluções para problemas da Engenharia de Software (HARMAN, 2001).

Em SBSE, os problemas da engenharia de software são formulados como problemas de otimização a serem solucionados através de técnicas baseadas em buscas, tais como algoritmos genéticos. Os problemas a serem abordados são relacionados com complexas tarefas humanas. SBSE provê técnicas de buscas automatizadas que encapsulam os pressupostos e a intuição humana, reduzindo assim o esforço humano (VERGILIO, 2011).

Segundo ainda Vergilio (2011), SBSE foi criada em 1990, onde a técnica baseada em busca era aplicada para testes de software e gerenciamento de projetos. O termo SBSE foi usado pela primeira vez por Harman e Jones, em 2001, num artigo que foi considerado uma apresentação de SBSE e desde então vários trabalhos foram publicados pela comunidade de pesquisa, conforme FIGURA 9, onde os artigos referentes a 2011 ainda não haviam sido totalmente atualizados na base de dados SEBASE (2011).

FIGURA 9 – Número de Publicações em SBSE por Ano entre 1976 e 2011.



Fonte: Repositório SEBASE (2011)

2.3.1 Características e atuação

A abordagem proposta pela computação evolucionária tem ganhado espaço na SBSE por conta das características dos problemas que, por serem NP-difíceis, não são adequados para solução pelas técnicas exatas (HARMAN, 2007). Segundo Harman (2011), a baixa preocupação com o campo da computação evolucionária foi um dos fatores que motivaram a criação de SBSE.

Mas, não há nenhuma razão para que SBSE venha a procupar-se apenas com a computação evolutiva. De acorco com Simons (2013), na última década vários outros trabalhos em torno da engenharia de software foram criados, tais como:

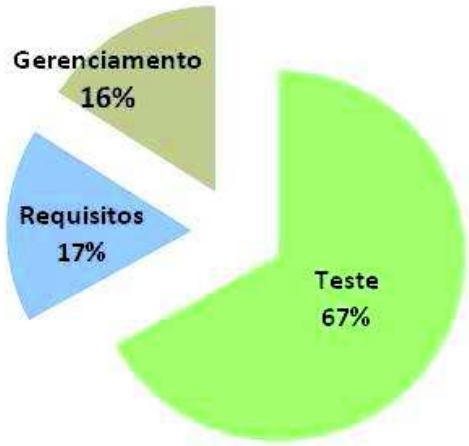
- Análise de requisitos e programação (Ren et al., 2011);
- Técnicas e ferramentas de design (Bowman et al., 2010 e Simons et al., 2010);
- Teste de Software (McMinn, 2004);
- Correção de *bugs* automatizada (Weimer et al., 2010);
- Manutenção de software (O'Keefe e Cinneide, 2008)

Para Harman (2009), as peças chaves para aplicação de SBSE são: a) a escolha da representação do problema e b) a definição de uma função *fitness*, que avalia as soluções encontradas. Esses requisitos fazem com que a aplicação de SBSE seja bastante atraente. Dessa forma, torna-se possível implementar algoritmos baseados em busca que utilizam diferentes abordagens para encontrar boas soluções.

Um grande desafio da SBSE atualmente é fazer com que as técnicas que usem os algoritmos genéticos sejam inclusos pelos engenheiros no processo de fabricação de softwares como um suporte aos processos do desenvolvimento. A otimização deve ser utilizada ao máximo para automatizar as contruções de softwares (Harman, 2011).

Vergilio (2011) mostra a publicação de pesquisas em SBSE no Brasil. Segundo a autora, que considerou trabalhos publicados entre 1992 e 2010 no SEBASE (2011), a partir de 2008, os trabalhos publicados deram um salto em quantidade e qualidade alavancados pelo crescimento dos eventos na área, tais como: SSBSE (*Symposium on Search-Based Software Engineering*), GECCO (*Genetic and Evolutionary Computation Conference*), SBES (Simpósio Brasileiro de Engenharia de Software). Na FIGURA 10, são consolidados os trabalhos publicados em SBSE de acordo com a área.

FIGURA 10 – Publicação de trabalhos brasileiros em SBSE



Fonte: Vergilio (2011)

Adicionalmente, no mapa exibido na FIGURA 11, é exibida a distribuição dos trabalhos em SBSE publicados de acordo com cada estado brasileiro de origem.

FIGURA 11 – Mapa com a distribuição das publicações de trabalhos em SBSE no Brasil

Fonte: Vergilio (2011)

Em HARMAN (2009), os autores demonstram em detalhes toda atividade desenvolvida em SBSE até então, incluindo áreas de aplicação, técnicas utilizadas e os resultados alcançados. Seguindo a publicação, podemos encontrar problemas e desafios a serem enfrentados no campo da SBSE.

2.3.2 A engenharia de requisitos e a SBSE

A SBSE aborda há tempo problemas relacionados a Engenharia de Requisitos. Um dos mais tradicionais na literatura diz respeito à seleção do conjunto de requistos que devem estar presentes nas próximas iterações do desenvolvimento.

Uma contribuição importante da SBSE é o fato das técnicas permitirem aplicação de restrições e mudanças ao longo da implementação. Ela pode, por exemplo, fornecer ótimas soluções e mesmo ocorrendo uma mudança, essas soluções continuarem sendo ótimas. Esse quadro está mais próximo da realidade dos projetos de software que a robustez sujeita a mudança tem um grande valor para a empresa (ZHANG, 2008).

2.4 Conclusão

Esse capítulo procurou mostrar a fundamentação das áreas que serão tratadas ao longo desse trabalho.

No próximo capítulo, faremos um levantamento bibliográfico acerca dos principais problemas abordados pelo trabalho: seleção, alocação e priorização de requisitos.

3 TRABALHOS RELACIONADOS

Neste capítulo contextualizamos os trabalhos que envolvem seleção, alocação e priorização de requisitos. Consideramos esse o ponto de partida para discutirmos mais à frente a abordagem proposta por esse trabalho.

Ao final, trataremos da seleção e da priorização de requisitos de acordo sua estabilidade.

3.1 Seleção, alocação e priorização de requisitos

O trabalho publicado por Bagnall (2001) trata da determinação dos requisitos que devem ser executados para a próxima entrega (*release*) do software. O autor prediz que os clientes possuem diferentes níveis de importância para a empresa e aponta os requisitos que possuem pré-requisitos e que devem ser realizados em *release* anterior ou em paralelo a que está sendo implementada. Os algoritmos aplicados nessa estratégia mostram a obtenção de soluções rápidas para problemas pequenos.

Para Karlsson e Ryan (1996), um dos maiores riscos enfrentados por organizações que desenvolvem software comercial está associado ao não atendimento das necessidades e expectativas dos usuários. Para esses autores, esse risco pode ocasionar danos na reputação, perda de pedidos e redução dos lucros da empresa.

Pesquisas que visam à identificação das causas para o problema citado apontam a fase de elicitação e alocação de requisitos como básica para a melhoria do processo. Ela é uma das atividades que ocorre no início do desenvolvimento de software. Para Sadraei et al. (2007), os requisitos de software são determinantes críticos da sua qualidade.

Greer et al. (2004) afirmam que definir em qual *release* entregar o requisito é uma decisão que depende de várias variáveis que se relacionam de forma complexa. Tratam-se das diferentes perspectivas dos *stakeholders* e do planejamento dos *releases*, incluindo restrição de esforço.

A abordagem feita por Ruhe e Saliu (2005) utiliza um modelo que usa alguns critérios: variáveis de decisão, dependência entre funcionalidades e a valorização dos requisitos tanto para o cliente quando para a empresa de desenvolvimento de software.

Isso reafirma que o processo de desenvolvimento de software é muito mais complexo do que simplesmente desenvolver o próprio software. O número de variáveis que compõem as *releases* muitas vezes são maiores que as já percebidas pelo gerente de software.

Por isso é fundamental uma boa elaboração de planejamento, pois planejamentos mal elaborados podem resultar em clientes insatisfeitos, planejamento de releases que não atendem às restrições e releases que não agregam valor ao negócio (RUHE e SALIU, 2005).

Bagnall et al (2001) definem a seleção dos requisitos para o enquadramento no próximo *release* de acordo com os pré-requisitos de cada requisito, grau de satisfação dos clientes, prioridade do cliente e o tempo e o esforço da implementação de cada requisito.

A proposta de alocação elaborada por Greer (2004) leva em consideração o valor do negócio agregado, os recursos disponíveis e a precedência entre os requisitos. Porém essa abordagem não leva em consideração os riscos inerentes a cada requisito.

Baker et al (2006) definem a alocação dos requisitos de um release de acordo com a evolução do software. Usam-se algoritmos gulosos e têmpera simulada e comparam com o julgamento de um especialista para um conjunto de características do mundo real, representado pelos componentes em um software.

Na alocação de requisitos, é importante ressaltar que devemos levar em consideração os recursos que implementarão esses requisitos. No que se refere à alocação dos recursos, Burdett e Li (1995) tratam a formação de grupos de trabalho de acordo com algumas características próprias dos recursos, como: preferências que devem ser quantificadas e representadas, fatores de habilidade e o custo salarial.

É difícil atender a todos os requisitos identificados para um sistema, principalmente devido a restrições de tempo e orçamento. Os requisitos geralmente são desenvolvidos em etapas e a priorização ajuda a definir quais devem ser implementadas primeiro (ALLEN et al., 2008).

Segundo Karlsson, Wohlin e Regnell (1998), os requisitos devem ser alocados em diferentes versões do software e, para Berander (2004), a seleção “correta” dos requisitos que farão parte de cada versão é a etapa principal em direção ao sucesso de um projeto ou produto. Por isso, é preciso distinguir aqueles que terão maior impacto para a satisfação dos usuários. Alguns métodos destacados por Allen et al. (2008) são descritos na TABELA 1.

A priorização é relevante porque os usuários têm expectativas em relação ao software que está sendo gerado. A entrega dos requisitos segundo a percepção dos usuários, além de contribuir para a satisfação deles, pode ser essencial para a continuidade do projeto (CORDEIRO, 2011).

TABELA 1 – Métodos para priorização de requisitos de software

Métodos	**Breve descrição**
Atribuição numérica	Utilizando uma escala de valores inteiros que varia de 1 a 5, os *stakeholders* devem identificar a qual nível da escala cada requisito corresponde.
Método dos 100 pontos	100 pontos devem ser distribuídos entre os requisitos, atribuindo mais pontos aos mais importantes.
Triagem de requisitos	Processo que visa definir a prioridade relativa dos requisitos, estimar recursos para cada requisito e selecionar um subconjunto de requisitos para otimizar a probabilidade de sucesso do projeto.
AHP (*Analytic Hierarchy Process*)	Método de apoio à decisão utilizado em situações nas quais múltiplos objetivos estão presentes. Utiliza a comparação paritária para calcular o valor (importância) relativo de cada item (requisito).

Fonte: Allen et al. (2008)

Além dos fatores já vistos, podemos encontrar outros aspectos, como a volatilidade que impacta na priorização de requisitos. Faz-se necessário grande esforço para selecionar e priorizar requisitos voláteis. Esse tipo de requisito geralmente é considerado um problema indesejável. Estudos anteriores já haviam identificado que as suas características podem produzir impactos adversos sobre os processos do desenvolvimento de software (NURMULIANI, 2004). Por exemplo, um estudo realizado por Curtis (1988) indica que os requisitos voláteis correspondem a uma significativa parcela dos problemas enfrentados por empresas de desenvolvimento de softwares.

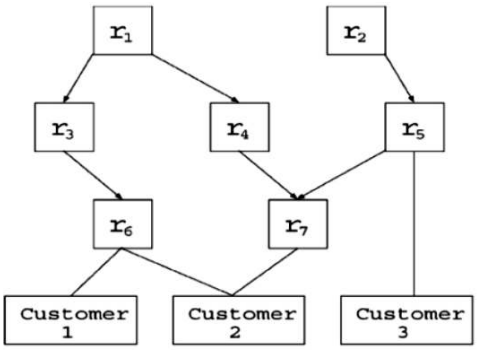
Nurmuliani (2004) realizou um estudo real numa empresa de desenvolvimento de software a fim de identificar as causas de volatilidade nos requisitos e o impacto disso nos projetos da empresa. Em ordem decrescente, o autor considerou que as maiores mudanças em requisitos devem-se: a) inclusão de novos requisitos ao sistema, b) exclusão de requisitos e c) modificação das características dos requisitos.

O autor pontuou também que essas mudanças devem-se a fatores externos, tais como: a) mudanças das demandas do mercado, b) maior entendimento dos desenvolvedores quanto à implementação dos requisitos e c) outras considerações do cliente, como redução do escopo e mudanças na política da empresa.

Bagnall (2001) propôs um trabalho chamado de *The next release problem*, onde o autor, pioneiro nesse campo de pesquisa, apresenta o problema como uma busca de quais

características devem ser escolhidas em relação às variáveis, dependências entre requisitos e prioridade de requisitos. A seguir a FIGURA 12 demonstra que os requisitos r(n), onde n representa a identificação do requisito, estão associados aos *clientes(n)*, onde *n* representa a identificação do cliente.

FIGURA 12 – Representação da associação de requisitos aos clientes.

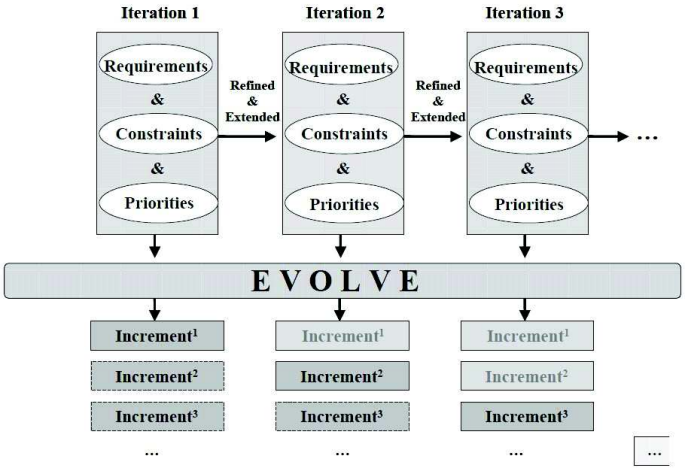


Fonte: Bagnall (2001)

No caso da FIGURA 12, o autor utilizou as metaheurísticas Têmpera Simulada (LAARHOVEN, 1987), GRASP (*Greedy Randomized Adaptive Search Procedure*) (FEO E RESENDE, 1989), entre outros. Nesse caso, a têmpera simulada demonstrou ser mais eficaz na solução do problema.

Greer (2004) publicou um trabalho que usa no Planejamento de Releases de Software os algoritmos genéticos. O método EVOLVE, como foi definido, possui duas funções cujo os objetivos são de maximizar o benefício total e minimizar as penalidades. Por isso, o método auxilia na tomada de decisões em ambientes sujeitos a mudança. Cada *stakeholder* possui uma importância para a organização e atribui valores de importância e urgência a cada requisito. A dependência e precedência entre as releases são levadas em consideração nessa abordagem. Destaca-se também que o número de *releases* não é fixo. Para avaliar a proposta da aplicação do EVOLVE, o autor aplicou-o em um estudo de caso com um projeto abstrato.

FIGURA 13 – Metodologia EVOLVE



Fonte: Greer (2004)

Brasil (2010) elaborou uma abordagem onde a autora utiliza uma nova formulação multiobjetivo de otimização matemática para o problema do planejamento de *releases* considerando os seguintes fatores: a) satisfação de clientes, b) priorização, c) valor de negócio, d) riscos envolvidos, e) recursos disponíveis e f) precedência entre requisitos. A abordagem proposta objetiva: a) maximizar a satisfação do cliente, priorizando os requisitos mais importantes para este, b) minimizar os riscos do projeto implementando primeiro os requisitos de maior risco e tem como restrição os limites de tempo, custo e precedência entre cada requisito. Para implementar uma solução, a autora usou as metaheurísticas NSGA-II (Deb, 2002), MOCell e uma estratégia aleatória. Como resultado dessa abordagem, deduziu-se que a estratégia aleatória foi superada pelas metaheurísticas. Ambas as metaheurísticas geraram soluções melhores considerando os objetivos, o que corrobora o uso dessas abordagens.

A abordagem proposta nesse trabalho trata de uma nova modelagem para o planejamento de *release* de software que leva em consideração a estabilidade do requisito como fator para a seleção e priorização de requisitos que serão alocados no próximo release.

3.2 Conclusão

Os fundamentos teóricos acerca da engenharia de requisitos demonstram os problemas cotidianos que envolvem os processos das empresas de desenvolvimento de software. As abordagens baseadas em otimização mostram ser eficientes na busca de soluções de problemas, agregando a SBSE grande valia para solução de complexos problemas.

4 MODELAGEM DO PROBLEMA DE SELEÇÃO E PRIORIZAÇÃO DE REQUISITOS DE SOFTWARE

Este capítulo apresenta a contextualização do problema abordado neste trabalho. Serão observados pontos importantes sobre requisitos tais como priorização e volatilidade. Além disso, será apresentada uma visão geral sobre os componentes envolvidos e a estratégia proposta para resolver o problema de priorização de requisitos estáveis.

4.1 Contextualização

Uma vez que um conjunto inicial de requisitos tem sido elencado no levantamento de requisitos, existe um problema de análise de nível de negócio: as escolhas têm de ser feitas para identificar as melhores soluções aos gerentes de negócio (ZHANG, 2008).

Requisitos podem ter certo grau de volatilidade por estarem submetidos a constantes mudanças e indefinições. Essa característica, se levada em consideração, pode influenciar na ordem em que esses requisitos vão ser implementados.

Esse trabalho propõe uma abordagem para o Planejamento de *Release* de Software que leve em consideração a volatilidade do requisito como critério para priorização em *releases* de software.

Desse modo, o trabalho tem como proposta indicar uma estratégia envolvendo o Planejamento de Relesase de Software, objetivando ser útil como uma ferramenta semi-automatizada de apoio ao processo decisório, uma vez que a partir dos dados coletados sobre o problema abordado, os gerentes tenham à sua escolha soluções que possam ser analisadas e avaliadas.

4.2 Pontos relevantes

Baseando-se no que foi colocado no item 2.3 que aborda sobre SBSE e no item 3.1 que trata da seleção e priorização de requisitos, apresentam-se aqui alguns dos atributos que envolvem os requisitos na engenharia de software e que serão utilizados durante a execução da abordagem proposta.

4.2.1 Requisitos

Os requisitos formam um conjunto de enunciados que descrevem as necessidades e os desejos do usuário. Esses requisitos devem ser entendidos de forma clara e integral pelos

engenheiros de software responsáveis pelo desenvolvimento do sistema de software. Contudo muitas vezes encontramos requisitos que invadem o limite do "como" e adentram a área de design da solução. Embora devamos tentar limitar os requisitos à esfera do "o quê", nem sempre tais fronteiras são nítidas (TSUI e KARAM, 2013).

Uma das principais avarias em projetos de software são especificações de requisitos incompletas. Um dos fatores importantes para o sucesso de um projeto pode ser atribuído ao claro enunciado dos requisitos. A importância dos requisitos de usuário agora é bem valorizada. A gestão de requisito e o envolvimento do usuário estão adquirindo *status* de tarefas-chave em desenvolvimento de software, independente do modelo de processo de desenvolvimento (TSUI e KARAM, 2013).

Um projeto de software envolve as seguintes atividades de engenharia de requisitos (TSUI e KARAM, 2013):

- Levantamento de requisitos;
- Análise de requisitos;
- Revisão e validação de requisitos;
- Contrato e aceite dos requisitos do projeto a serem implementados.

Nem todas estas atividades envolvendo requisitos são necessárias a todos os projetos de software num mesmo nível.

Na abordagem consideramos $R = \{r_i | \ i = 1, 2, \dots, N\}$ o conjunto de requisitos a serem desenvolvidos.

4.2.1.1 Custos de requisitos

O custo do requisito diz respeito aos recursos necessários para sua implementação. Existem muitas abordagens que se baseiam em reduzir ao máximo possível o custo de implementação de todos, ou, ao menos, de parte dos requisitos do projeto.

Assim, a função $cost_i$ representa a quantidade de recursos que são necessários para a implementação completa do requisito r_i. Esse custo de implementação de r_i é representado numa escala que vai de 10, onde temos o menor valor necessário para implementação do requisito, até 20, que representa o maior valor estimado de recursos necessários.

Essa escala serve para parametrizar e balizar as avaliações de importância, estabilidade, custo, tempo e permitir o desenvolvimento da modelagem matemática do problema.

4.2.1.2 Volatilidade de requisitos

Mudanças em requisitos ocorrem enquanto os requisitos estão sendo elicitados, analisados, validados e após o sistema ter entrado em operação. Alterações em requisitos são inevitáveis e podem ser resultados de fatores como a) conflitos entre requisitos, b) evolução do conhecimento do cliente em relação ao software ou c) mudanças de prioridades do cliente.

Embora a mudança seja inevitável, é usual o caso em que alguns requisitos são mais estáveis que outros. Requisitos estáveis são concebidos com a essência de um sistema e seu domínio da aplicação, e mudam mais lentamente que requisitos voláteis.

Os requisitos voláteis são específicos para a instalação de um sistema em um ambiente particular e para um cliente particular. Há pelo menos quatro tipos de requisitos voláteis (KOTONYA, 1997):

- Requisitos mutáveis: mudam em função de mudanças no ambiente no qual o sistema opera. Por exemplo, os requisitos para um sistema que calcula taxas de dedução que evolui conforme as leis de taxação mudam;
- Requisitos emergentes: podem ser completamente definidos quando o sistema é especificado, mas que emergem quando o sistema está projetado e implementado. Por exemplo, pode não ser possível especificar de antemão os detalhes de como a informação será exibida. Conforme os stakeholders vêem exemplos de apresentações possíveis, eles podem pensar em novas maneiras de exibição da informação que seria útil para eles;
- Requisitos conseqüentes: são baseados em suposições de como o sistema será usado. Quando o sistema é posto em uso, algumas destas suposições podem estar erradas. Usuários podem encontrar novas maneiras de usar as funcionalidades, o que resultará em demandas dos usuários para mudanças no sistema;
- Requisitos de compatibilidade: dependem de outro processo. Conforme muda esse processo, os requisitos também mudam.

Pode ser uma boa prática de gerenciamento de requisitos tentar antecipar mudanças de requisitos, o que envolve classificar os requisitos para identificar os mais voláteis e implementar primeiro os mais estáveis.

Na abordagem proposta, cada requisito r_i possui um grau de estabilidade associado, mensurado numa escala que vai de 1 (menos estável) a 10 (mais estável), representado por $stability_{\mathrm{i}}$, que pode ser definido quão estável um requisito é, em termos de sua definição, probabilidade de mudança, entre outros.

4.2.1.3 *Stakeholders*

O ciclo de vida do software é composto por diversas responsabilidades atribuídas a pessoas, grupos e entidades a quem chamamos de *stakeholders*. Entre essas responsabilidades, podemos citar o financiamento, o projeto, o desenvolvimento, o teste, o uso e a manutenção do software (ROZANSKI, 2005).

É importante lembrar que dentro de um mesmo grupo de *stakeholders* podem existir interesses conflitantes entre si. Afinal, um grupo pode se organizar em subgrupos de interesses comuns, mas um subgrupo pode demonstrar interesses conflitantes com outro subgrupo. Portanto, subgrupos diferentes de usuários ou de desenvolvedores resultam em requisitos diferentes, que significam atributos de qualidade diferentes e que são frutos de arquiteturas diferentes (ROZANSKI, 2005).

Resolver satisfatoriamente o problema de planejamento de *releases* envolve atender as necessidades de um grupo variado de *stakeholders* que podem ser clientes, usuários, gestores de negócio, líderes de projeto, equipe de desenvolvimento, entre outros.

Saber quais são as reais necessidades dos *stakeholders* é fundamental para o desenvolvimento de uma solução eficiente para o Planejamento de *Releases* de Software. Em geral, os *stakeholders* têm diferentes perspectivas sobre o problema e as diferentes necessidades que devem ser abordadas pelos planos de *releases* em termos de qualidade, tempo ou valor de negócio (YANG, 2008). Ressalta-se ainda que os *stakeholders* podem possuir diferentes valores (importância) para a empresa de desenvolvimento de software, conforme pontuado na representação a seguir.

Para nossa abordagem, consideraremos $C = \{c_m | \, m = 1, 2, \dots, M\}$ o conjunto de clientes inerentes ao processo de desenvolvimento aos quais os requisitos devem ser atendidos. Consideramos ainda um valor de atribuição c_m referente à importância desse *stakeholder* para a empresa de desenvolvimento de software. Desse modo, podemos definir que w_m representa a importância do *stakeholder* para a empresa, onde numa escala progressiva, 1 (valor mínimo) seja a atribuição de menor importância do *stakeholder* para a empresa e 10 (valor máximo) a maior importância.

4.2.1.4 Valor do requisito (versus importância do cliente)

Requisitos têm valores variáveis tanto para a empresa de desenvolvimento de software quanto para o cliente. Ambos podem identificar se determinados requisitos são ou não viáveis ao longo do processo de implementação. Isso pode ocorrer por oscilações de curto prazo no mercado, alocação de pessoas capacitadas num tipo de ferramenta entre outros.

Clientes podem organizar-se em comitês para tomada de decisões ao longo do desenvolvimento do projeto. Esses clientes podem ter diferentes expectativas em relação a um requisito. Isso faz com que a importância dada a esse requisito também seja fixa para cada cliente. Isso pode influenciar na priorização desses requisitos por parte do gerente de projeto.

Uma saída para esse problema é classificar os clientes, colocando-os num *ranking* de acordo com sua importância para a empresa e então selecionar os requisitos de acordo com a nota atribuída pela empresa a um determinado cliente que por sua vez também atribuiu uma nota a determinado requisito.

No presente trabalho, consideramos $value(m, i)$, valor que quantifica a importância que um *stakeholder* c_m associa a um requisito r_i, atribuindo um valor que varia de 1 (baixa importância) a 10 (alta importância).

4.2.1.5 Precedência entre requisitos

A precedência entre requisitos em engenharia de software diz respeito ao fato de que um requisito não é passível de implementação sem ter um antecessor já implementado.

Nos projetos de desenvolvimento, a precedência técnica entre os requisitos é algo importante para a organização do projeto, a alocação de recursos, a efetuação correta das entregas além da própria disponibilidade técnica para possibilitar a implementação do requisito. Por exemplo, supondo que um requisito r_j possui uma dependência técnica com um requisito r_i, r_i é pré-requisito para r_j. Neste caso, o requisito r_i deve ser implementado antes do requisito r_j.

4.2.1.6 Priorização de requisitos

Priorizar requisitos consiste em ordená-los de modo que sejam implementados e liberados para os clientes aqueles com prioridades mais altas. Muitas vezes, um produto de software de múltiplas versões é planejado ao longo de vários trimestres e até mesmo anos por

requisitos priorizados. A prioridade dos requisitos pode ser estabelecida com base em muitos critérios, incluindo os seguintes (TSUI e KARAM, 2013):

- Exigências atuais do cliente;
- Concorrência e condição do mercado atual;
- Necessidades de futuros e novos clientes;
- Vantagem de vendas imediatas;
- Problemas críticos no produto existente.

Além das ferramentas de auxílio à tomada de decisões, os analistas de requisitos realizam reuniões com clientes e especialistas para colaborarem nas discussões de priorização. Grande parte da priorização de requisitos de software é executada com pessoal e clientes experientes, adotando-se uma abordagem informal na qual se pontua arbitrariamente os requisitos mais prioritários. Essa abordagem informal é frequentemente empregada, mas raramente possui bons resultados. (TSUI e KARAM, 2013).

A priorização de um requisito contribui para se decidir se um requisito em particular será incluído no *release* atual, no próximo *release* ou em alguma versão futura.

Um método de priorização de requisito chamado de Processo Analítico Hierárquico (*Analytical Hierarchy Process,* AHP) foi proposto por Karlsson e Ryan (1997). Nesse método, cada requisito é comparado a cada um dos demais requisitos em regime de dois a dois. Um ‘valor de intensidade’ é atribuído a essa relação. Os requisitos com os valores de intensidade global mais altos será essencialmente o mais prioritário. O requisito com o próximo valor mais alto de intensidade relativa global será o segundo requisito mais prioritário, e assim por diante (TSUI e KARAM, 2013).

Priorizar requisitos mais estáveis garante que mudanças advindas sejam passadas para os requisitos do final da fila, ou os mais voláteis, e que ainda não foram implementados. Isso reduz o custo global do projeto, além de assegurar o cumprimento de prazos de entrega ao cliente, já que os primeiros requisitos entregues são mais estáveis em relação aos demais.

4.3 Estratégia da proposta

A seguir é apresentada uma visão geral da estratégia proposta. A intenção é fornecer à estratégia informações referentes a requisitos, *stakeholders*, os dados do projeto e a importância dos requisitos para os *stakeholders*, além dos objetivos e restrições. De posse

dessas informações e usando as metaheurísticas, obtém-se a priorização de requisitos de acordo com a sua estabilidade.

4.3.1 Priorização dos requisitos mais importantes e mais estáveis

Esse trabalho tem como objetivos definidos:

- Maximizar a satisfação dos clientes e o valor de negócio agregado, selecionando para implementação os requisitos mais importantes, do ponto de vista dos *stakeholders* mais importantes;
- Maximizar o grau estabilidade entre os requisitos do projeto, implementando primeiramente os requisitos com maior estabilidade agregada.

Sujeitos às restrições que seguem:

- Respeitar a matriz de precedência técnica existente entre os requisitos do projeto;
- Respeitar os recursos disponíveis para o projeto.

Na estratégia apresentada, é respeitada a precedência entre requisitos e a restrição de recursos. Além disso, esses requisitos são selecionados e priorizados de acordo com a importância e estabilidade de cada um.

Portanto, o objetivo da estratégia é priorizar para implementação os requisitos mais importantes para os *stakeholders* e manter uma ordem de implementação que leve em consideração os requisitos que possuírem o maior grau de estabilidade.

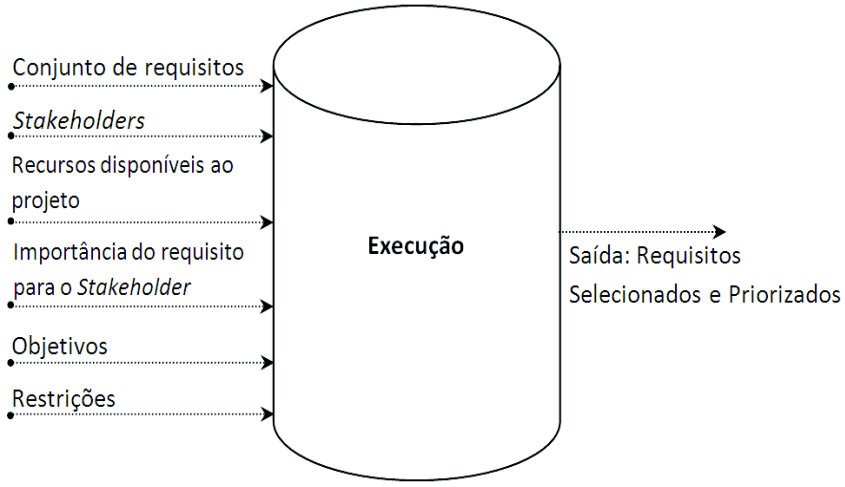
Considerando que os recursos para o projeto são limitados, não se pode garantir que todos os requisitos serão selecionados. Porém, garante-se que no provável esgotamento de recursos, os requisitos mais estáveis já foram implementados.

Como entrada para a abordagem, temos:

- Requisitos – quantidade de requisitos (N), custo de implementação $(\mathrm{cost_i})$, estabilidade $(stability_i)$, matriz de precedência técnica dentre os requisitos;
- *Stakeholders* – quantidade de *stakeholders* (M), importância do *stakeholder* para a organização (w_m) desenvolvedora do software;
- Projeto – recursos disponíveis para o projeto $(\mathrm{resourceProject})$;
- Relacionamentos – importância do requisito para o *stakeholder* $(value(m, i))$.

A figura a seguir ilustra essa modelagem.

FIGURA 14 – Estratégia – Seleção/Priorização de Requisitos



Fonte: Elaborado pelo autor (2013)

4.4 Modelagem matemática do problema

A formulação matemática para a elaboração da estratégia descrita no item 4.3 foi elaborada conforme segue.

(I) $Max\ f_{VALUE}(y) = \sum_{i=1}^{N} score_i . y_i$

(II) $Min\ f_{VOLATILITY}(x^{Pos}) = \sum_{i=1}^{N} (stability_i . x^{Pos}{}_i) . y_i$

Sujeito a:

(III) $\sum_{i=1}^{N} cost_i . y_i \leq resourceProject$

(IV) $x^{Pos}{}_i < x^{Pos}{}_j$, se $Dr_i, r_j = T1$ $(Precedência\ Técnica: r_i\ precede\ tecnicamente\ r_j)$

A variável $x^{Pos}{}_i$ aponta a posição do requisito r_i, podendo assumir um valor em $\{0, 1, 2, \ldots \mathrm{N}\}$, na ordem para a implementação estabelecida pela priorização, para $\mathrm{i} = 1, 2, \ldots N$.

A variável $\mathrm{y_i}$ indica se o requisito $\mathrm{r_i}$ será implementado ($\mathrm{y_i} = 1$) ou não ($\mathrm{y_i} = 0$), para $\mathrm{i} = 1, 2, \ldots \mathrm{N}$.

Função I – Demonstra o grau de satisfação dos *stakeholders* à implementação de um grupo de requisitos, onde o $score_i = \sum_{m=1}^{M} w_m . value(m, i)$ expressa o valor de negócio ao requisito r_i. Desse modo e, considerando ponderadamente a importância do cliente, a função agrega mais valor à medida que mais requisitos forem selecionados.

Função II – Representa o grau de estabilidade dos requisitos do projeto, por meio da implementação adiantada dos requisitos considerados mais estáveis. Essa função calcula o produto entre a estabilidade do requisito e a posição na qual ele foi alocado. Desse modo, um valor menor da função indica que os requisitos com maior estabilidade foram priorizados.

Por fim, são apresentadas em III e IV as restrições da estratégia, onde III é a restrição de custo da implementação dos requisitos ao orçamento disponível e IV representa a restrição de precedência entre os requisitos. Se um requisito r_i precede um requisito r_j, então r_i deve ser implementado antes de r_j $\left(x^{Pos}{}_i < x^{Pos}{}_j\right)$.

4.5 Conclusão

Por meio da contextualização apresentada e da formulação matemática elaborada, a abordagem foi detalhada a um maior grau de entendimento neste capítulo. Nos capítulos seguintes faremos a avaliação dos experimentos realizados e chegaremos à conclusão dos estudos.

5 AVALIAÇÃO

Para ratificar a eficiência da abordagem proposta, descrevemos neste capítulo as instâncias que serão os cenários para a avaliação, as métricas de desempenho utilizadas, os resultados provenientes das técnicas (metaheurísticas) aplicadas e a análise dos resultados.

5.1 Planejamento dos experimentos

A presente seção objetiva parametrizar e catalogar toda a fase de planejamento dos experimentos.

5.1.1 Definições das instâncias

Com o objetivo de analisar a abordagem proposta, foram criadas, através do desenvolvimento de um gerador (FIGURA 17a), as instâncias que simulam os ambientes do problema. Essas instâncias foram aleatoriamente geradas obedecendo às configurações de parâmetros definidos numa escala, variável para cada atributo. No apêndice A, o conteúdo da instância A1_I.10.5.5.0.80.0 é detalhado.

Na TABELA 2, observam-se as características de cada instância utilizada no projeto.

TABELA 2 – Descrição das instâncias

Nome da Instância	Número de Requisitos	Número de Clientes	Orçamento Disponível no Projeto	Precedência Técnica
A1_I.10.5.5.0.80.0	10	5	80%	5%
A1_I.50.5.5.0.80.0	50	5	80%	5%
A1_I.50.10.5.0.80.0	50	10	80%	5%
A1_I.50.15.5.0.80.0	50	15	80%	5%
A1_I.100.5.5.0.80.0	100	5	80%	5%
A1_I.100.10.5.0.80.0	100	10	80%	5%
A1_I.100.15.5.0.80.0	100	15	80%	5%
A1_I.200.5.5.0.80.0	200	5	80%	5%
A1_I.200.10.5.0.80.0	200	10	80%	5%
A1_I.200.15.5.0.80.0	200	15	80%	5%

Fonte: Elaborado pelo autor (2013)

Foram definidos no item 4 o conjunto de requisitos a serem desenvolvidos, o grau de estabilidade de um requisito, o nível de importância do cliente para a organização e a importância que um cliente associa a um requisito.

Definimos em 80% o orçamento disponível no projeto para executar todos os requisitos. Para a precedência técnica entre requisitos, foi estabelecido o valor de 5%, ou seja, para uma instância de 200 requisitos, 10 apresentarão dependência técnica. Esse percentual foi levado em consideração para geração das matrizes de precedência.

5.1.2 Estratégia aplicada para solução do problema

As metaheurísticas NSGA-II (SRINIVAS e DEB, 1994) e MOCell (NEBRO, 2009), respectivamente apresentadas no item 2.2.3.2 e no item 2.2.3.3, foram aplicadas a fim de buscar soluções para o problema. Além dessas, neste trabalho, utilizou-se também, como um referencial, os algoritmos aleatórios. Isso se deve ao fato da possibilidade de comparação e legitimação entre os resultados obtidos por esses algoritmos e as metaheurísticas.

Segundo Harman e Jones (2001), na SBSE uma metaheurística deve superar um algoritmo aleatório para que se possa ser considerada eficaz.

5.1.3 Configurações dos parâmetros dos algoritmos

Os parâmetros dos algoritmos foram concebidos ao longo dos ensaios das abordagens na busca pela melhor solução do problema.

Para isso, foram definidos os parâmetros descritos na TABELA 3, conforme segue.

TABELA 3 – Definição dos parâmetros das abordagens

Algoritmo	Parâmetros utilizados
NSGA-II	• Tamanho inicial da população: 250 indivíduos; • Número máximo de avaliações: 100.000 (resultando em 400 gerações); • Probabilidade de cruzamento: 0,9 (operador *TwoPointsCrossover*); • Probabilidade de mutação: 1,0 (operador *SwapMutation*); • Seleção utilizando o método do torneiro binário.
MOCell	• Tamanho da população inicial: 256 indivíduos; • Tamanho do arquivo externo: 256; • Número máximo de avaliações: 102.400 (resultando em 400 gerações); • Mecanismo de *feedback*: 20; • Taxa de cruzamento: 0,9 (operador *TwoPointsCrossover*); • Taxa de mutação: 1,0 (operador *SwapMutation*); • Seleção utilizando o método do torneiro binário.
Busca Aleatória	• Número máximo de avaliações: 100.000.

Fonte: Elaborado pelo autor (2013)

5.1.4 Framework *jMetal*

As aplicações das metaheurísticas sob a abordagem proposta foram realizadas com o auxílio do framework *jMetal*. Essa ferramenta implementada em linguagem Java e proposta por Durillo et al. (2006) fornece suporte para construção de metaheurísticas multiobjetivos. Compartilhando os mesmos componentes, o *jMetal* pode ser usado como meio de comparação entre diferentes abordagens para solução de um mesmo problema. Entre as abordagens disponíveis pela ferramenta, encontramos o NSGA-II e o MOCell.

Ainda com o auxílio do *jMetal*, fazemos uso de métricas para verificar a distribuição das soluções. Para isso, elas recebem como parâmetro a frente de Pareto e as funções objetivos do problema. Na FIGURA 17b é possível identificar a localização do *jMetal* na metodologia utilizada nesse trabalho

5.1.5 Métricas de avaliação

Para avaliação da abordagem proposta foram utilizadas as métricas que servem para avaliar a qualidade das soluções encontradas por técnicas de otimização multiobjetivo. Como o resultado de uma otimização multiobjetivo consiste em um conjunto com as melhores soluções, fica inviável realizar um comparação qualitativa considerando apenas o conjunto com as soluções obtidas. Objetivando analisar a qualidade das soluções oriundas do processo de otimização multiobjetivo, foram utilizadas várias técnicas para avaliar eficácia da abordagem proposta, tanto na diversidade quanto na convergência dessas soluções em relação à frente de Pareto.

5.1.5.1 *Hypervolume* (*HV*)

O *Hypervolume* é uma métrica frequentemente utilizada para comparar resultados de algoritmos multiobjetivos. Ela avalia a distribuição (convergência) do conjunto de soluções Q em relação ao espaço de busca e foi proposta por Zitzler (1998). O hipervolume de um conjunto de soluções A é a união de vários hipercubos e mede a área coberta ou dominada pelo conjunto A de pontos obtido por um algoritmo. Para calcular o *hypervolume* do conjunto de soluções, é necessário definir um ponto de referência. Em problemas de maximização, é comum utilizar o ponto (0,0), enquanto que em problemas de minimização um ponto de referência (x, y) é usado para limitar essa área. A área destacada na FIGURA 15a define o *hypervolume* de um conjunto de soluções A para um problema com duas funções objetivo f_1 e f_2, em que o ponto (x, y) foi definido como limite superior. Denotamos por $H(A)$ o

hypervolume de um conjunto de soluções *A* em relação a um ponto de referência. A união de todos os cubos v_i que originam o *hypervolume* é calculada conforme a seguinte formulação:

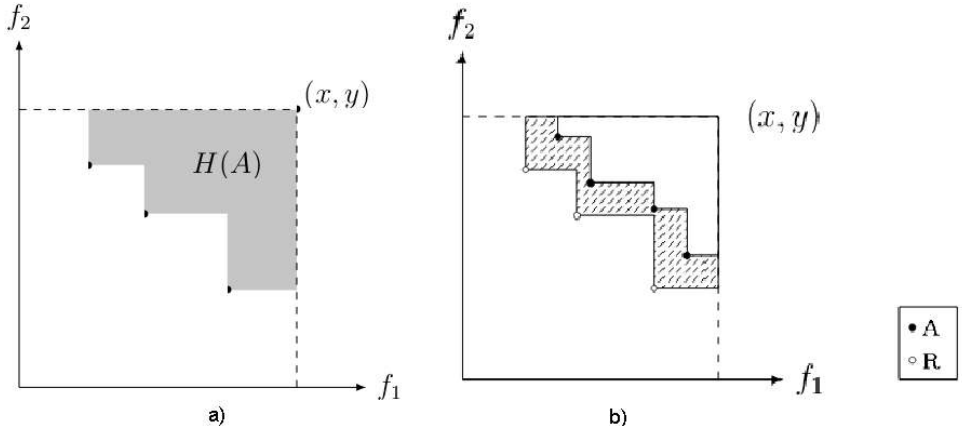
$$HV = volume\left(\bigcup_{i=1}^{|Q|} v_i\right)$$

A métrica diferença do hipervolume *H⁻(A)* fornece a área do espaço de objetivos de um conjunto de soluções *A* que é fracamente dominada pela área do espaço de objetivos do conjunto *R*. O valor de *H(A)* é calculado pela equação a seguir e ilustrada na FIGURA 15b:

$$H^-(A) = H(R) - H(A)$$

Um alto valor de hipervolume é considerado desejável, pois se denota que quanto mais próxima a solução estiver da frente de Pareto, maior será o volume apresentado.

FIGURA 15 – a) *Hypervolume,* b) Diferença do *Hypervolume*



Fonte: Rego (2013)

5.1.5.2 *Spacing (S)*

O *Spacing* (SANTANA, 2009) tem como objetivo medir a separação (diversidade) entre as soluções não-dominadas por toda frente de Pareto. A separação entre uma solução na frente de Pareto e suas soluções vizinhas é calculada segundo a equação a seguir:

$$d_i = min_j(|f_1^i(\vec{x}) - f_1^j(\vec{x})| + |f_2^i(\vec{x}) - f_2^j(\vec{x})|)$$

Onde d_i é a distância entre as soluções vizinhas da solução *i* da frente de Pareto em um espaço de dois objetivos. Nesta equação *i, j* = 1, ..., *n*, e *n* é o número de soluções não dominadas da frente de Pareto.

O *Spacing* representa a variância destas distâncias d_i, e é calculado segundo a equação que segue.

$$S = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(d_i - \overline{d})^2}$$

Onde $\overline{d}$ é a média das distâncias d_i de todas as *n* solução da frente de Pareto. Como toda medida de variância, é importante que $\overline{d}$ seja substancialmente maior que zero para termos uma distribuição realmente boa. Consequentemente, as soluções da frente de Pareto estarão separadas mais uniformemente. Um *Spacing* igual a zero significa que todas as soluções estão equidistantes na frente de Pareto.

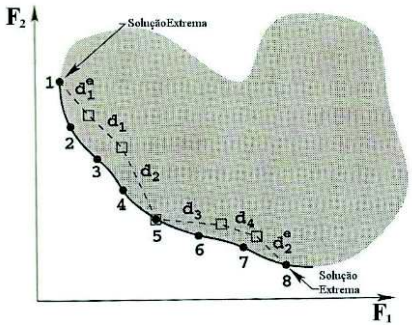
5.1.5.3 *Spread*

Métrica proposta por Deb (2009) que objetiva obter a diversidade entre as soluções não-dominadas de uma população. Sua formulação matemática pode ser vista como segue.

$$\Delta = \frac{\sum_{m=1}^{M} d_m^e + \sum_{i=1}^{|Q|} |d_i - \overline{d}|}{\sum_{m=1}^{M} d_m^e + |Q|\overline{d}}$$

Onde d_i é uma medida de distância entre soluções vizinhas (geralmente usa-se a distância euclidiana) e $\overline{d}$ é a média destas distâncias. O parâmetro d_m^e é a distância entre as soluções extremas de P^* e Q. Um exemplo deste parâmetro e das distâncias entre soluções consecutivas pode ser visto na FIGURA 16.

FIGURA 16 – Spread



Fonte: Deb (2009)

Esta métrica possui um valor ideal igual a zero quando todas as soluções estão distribuídas uniformemente na frente de Pareto. Logo, um algoritmo que encontra um baixo Δ encontrará uma melhor diversidade entre as soluções não-dominadas.

5.1.5.4 *Generational Distance* (*GD*)

A métrica *Generational Distance* (*GD*) (VELDHUIZEN, 1998) calcula a distância relacionado à convergência entre as soluções da frente de Pareto da instância atual (PF_{known}) com relação a frente de Pareto real (PF_{true}) (COELLO et al., 2002). Para cada solução do PF_{known} é calculada a menor distância com relação aos elementos de PF_{true}, definida por:

$$d_i = min_j(||f_{true}^j - f_{known}^i||)_2$$

Nesta métrica, d_i é multiplicada por e_i^{GER}, definindo uma ponderação para a distância. A normalização é obtida pela aplicação $(d_i - \underline{f_j})/(\overline{f_j} - \underline{f_j})$. A métrica é definida por:

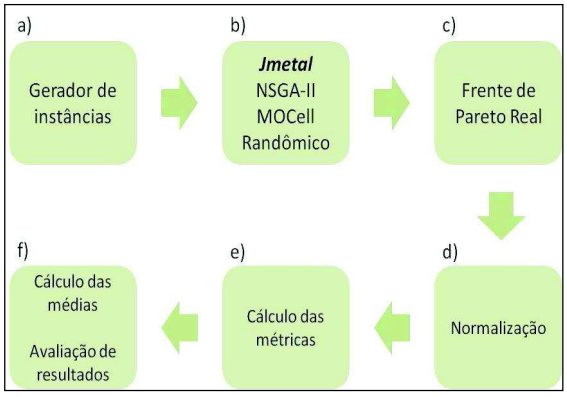
$$NDWD_{GER} = \sum_{i=1}^{N} e_i^{GER} \left(\frac{d_i - \underline{f_j}}{\overline{f_j} - \underline{f_j}}\right)$$

É preferível que um algoritmo alcance um baixo valor para esta métrica.

5.2 Apresentação e análise dos resultados

Para comparação dos resultados obtidos, além da mensuração dos tempos de execução, são utilizadas as métricas *Hypervolume (HV)*, *Spacing (S)*, *Spread*, *Generational Distance (GD)*, discutidas anteriormente, para medir a eficiência dos algoritmos (FIGURA 17e).

FIGURA 17 – Metodologia adotada para solução do problema



Fonte: Elaborado pelo autor (2013)

Para efetuar a medidas desses parâmetros, cada algoritmo é executado dez vezes, armazenando-se o tempo mediano entre todas as execuções. Por conseguinte, as métricas citadas são calculadas para cada uma das execuções. O desempenho dos algoritmos sob as

execuções é verificado através da média e do desvio-padrão de cada métrica em cada instância após dez execuções (FIGURA 17f).

A seguir, são apresentados e analisados os resultados das avaliações da estratégia proposta para as instâncias e algoritmos anteriormente mencionados. As tabelas a seguir exibem a média e o desvio-padrão dos resultados para cada instância após uma sequencia de dez execuções. Ao longo das apresentações dos resultados são feitas análises quantitativas e qualitativas.

Ao término dos experimentos e analisando os resultados para os tempos de execução, expostos na TABELA 4, observou-se que o algoritmo de busca aleatória apresentou o menor tempo e o menor desvio-padrão entre as instâncias executadas, obtendo assim o melhor desempenho neste quesito. Isso se deve ao fato de que os algoritmos aleatórios não utilizam critérios na geração de soluções o que, consequentemente, os faz ter melhor desempenho. No entanto, as soluções geradas nesse processo não possuem boa qualidade. Portanto, o torna inviável do ponto de vista qualitativo, pois as soluções são geradas baseando-se na aleatoriedade, sendo esta uma característica modesta para resolução de problemas. Isso ocorre por conta da convergência da técnica baseada apenas em critérios probabilísticos.

Considerando apenas os resultados do NSGA-II e do MOCell, observa-se que até 100 requisitos o MOCell apresentou os menores tempos de execução. Porém, ao alcançar a marca de 200 requisitos, o NSGA-II mostrou-se mais eficaz nas instâncias A1_I.200.5.5.0.80.0 e A1_I.200.15.5.0.80.0.

TABELA 4 – Resultados para tempos e execução (milissegundos)

Nome da Instância	MOCell	NSGA-II	Busca Aleatória
A1_I.10.5.5.0.80.0	4492,90 ± 1614,45	23101,10 ± 11669,68	7915,80 ± 3157,91
A1_I.50.5.5.0.80.0	8119,70 ± 2273,16	34787,00 ± 6945,98	3243,60 ± 604,76
A1_I.50.10.5.0.80.0	9144,40 ± 2308,71	38600,30 ± 11725,12	3552,40 ± 608,96
A1_I.50.15.5.0.80.0	9210,90 ± 2042,09	41268,40 ± 10923,21	3957,40 ± 754,31
A1_I.100.5.5.0.80.0	16090,50 ± 3266,34	45191,70 ± 2809,50	4547,00 ± 532,87
A1_I.100.10.5.0.80.0	15377,90 ± 3942,89	31941,70 ± 12081,66	4579,20 ± 1416,43
A1_I.100.15.5.0.80.0	14615,30 ± 3892,61	33337,00 ± 12476,66	5118,30 ± 1467,59
A1_I.200.5.5.0.80.0	34575,20 ± 7546,70	33923,30 ± 15307,08	7820,80 ± 1595,94
A1_I.200.10.5.0.80.0	36615,50 ± 5040,60	41512,30 ± 12111,01	8646,00 ± 2260,25
A1_I.200.15.5.0.80.0	34525,60 ± 6518,82	31328,90 ± 10910,06	8920,20 ± 1078,32

Fonte: Elaborado pelo autor (2013)

Devido à alta disparidade e à diferença de escalas entre os resultados obtidos em relação à frente de Pareto real, foi aplicada uma normalização dos valores das soluções com o objetivo de equiparar os resultados. Identifica-se através da FIGURA 17d, que depois de normalizados, os resultados foram avaliados pelas métricas (item 5.1.5).

Consoante com a seção 5.1.5.2, o *Spacing* objetiva medir a separação entre as soluções não-dominadas por toda frente de Pareto. Um baixo valor para essa métrica significa que as soluções estão igualmente separadas pela frente de Pareto, o que é desejável.

Conforme denotado na TABELA 5, observa-se que a busca aleatória obteve os maiores valores quando comparada com as metaheurísticas. Quando são comparados os resultados das metaheurísticas, observa-se que o NSGA-II foi melhor em oito das dez instâncias. O MOCell obteve bons resultados na menor (A1_I.10.5.5.0.80.0) e na maior instância (A1_I.200.15.5.0.80.0). Em linhas gerais, os três algoritmos obtiveram baixos valores, demonstrando assim uma boa separação de valores ao longo da frente de Pareto.

TABELA 5 – Resultados obtidos para a métrica *Spacing*

Nome da Instância	MOCell	NSGA-II	Busca Aleatória
A1_I.10.5.5.0.80.0	0,048201 ± 0,000314	0,049831 ± 0,001585	0,124781 ± 0,069482
A1_I.50.5.5.0.80.0	0,013268 ± 0,002391	0,010980 ± 0,002587	0,038650 ± 0,006109
A1_I.50.10.5.0.80.0	0,014843 ± 0,002399	0,011025 ± 0,001447	0,037753 ± 0,009123
A1_I.50.15.5.0.80.0	0,016610 ± 0,003162	0,011639 ± 0,002854	0,032237 ± 0,010928
A1_I.100.5.5.0.80.0	0,011908 ± 0,001841	0,010036 ± 0,001501	0,017277 ± 0,002547
A1_I.100.10.5.0.80.0	0,012234 ± 0,002180	0,010505 ± 0,001016	0,017375 ± 0,002061
A1_I.100.15.5.0.80.0	0,012194 ± 0,001057	0,010326 ± 0,001514	0,015525 ± 0,002879
A1_I.200.5.5.0.80.0	0,012980 ± 0,003993	0,011526 ± 0,003564	0,014682 ± 0,005082
A1_I.200.10.5.0.80.0	0,010608 ± 0,003915	0,009890 ± 0,003878	0,015449 ± 0,006295
A1_I.200.15.5.0.80.0	0,012928 ± 0,003397	0,012941 ± 0,006132	0,015715 ± 0,006917

Fonte: Elaborado pelo autor (2013)

Na TABELA 6, observam-se os resultados para a métrica *Generational Distance*, que segundo a seção 5.1.5.4, calcula a distância entre as soluções da frente de Pareto da instância atual com relação à frente de Pareto real. Portanto, quanto mais próximas as soluções estiverem da frente de Pareto real, menor o valor para essa métrica.

Os resultados mostram que a busca aleatória obteve os maiores, e, portanto os piores resultados. Entre as metaheurísticas, o NSGA-II superou o MOCell em todas as

instâncias, exceto a menor de todas elas (A1_I.10.5.5.0.80.0), mostrando que o MOCell tem maior habilidade para buscar soluções em instâncias menores.

TABELA 6 – Resultados obtidos para a métrica *Generational Distance*

Nome da Instância	MOCell	NSGA-II	Busca Aleatória
A1_I.10.5.5.0.80.0	0,000004 ± 0,000009	0,000045 ± 0,000040	0,003584 ± 0,001447
A1_I.50.5.5.0.80.0	0,001236 ± 0,000103	0,000551 ± 0,000176	0,002959 ± 0,000157
A1_I.50.10.5.0.80.0	0,001320 ± 0,000079	0,000505 ± 0,000049	0,002992 ± 0,000143
A1_I.50.15.5.0.80.0	0,001267 ± 0,000126	0,000440 ± 0,000031	0,002502 ± 0,000052
A1_I.100.5.5.0.80.0	0,001487 ± 0,000110	0,000651 ± 0,000115	0,002952 ± 0,000082
A1_I.100.10.5.0.80.0	0,001510 ± 0,000091	0,000598 ± 0,000073	0,002890 ± 0,000066
A1_I.100.15.5.0.80.0	0,001458 ± 0,000070	0,000628 ± 0,000096	0,002734 ± 0,000094
A1_I.200.5.5.0.80.0	0,001805 ± 0,000624	0,001790 ± 0,001091	0,004312 ± 0,001064
A1_I.200.10.5.0.80.0	0,001894 ± 0,001102	0,001860 ± 0,001811	0,005658 ± 0,001844
A1_I.200.15.5.0.80.0	0,002470 ± 0,001410	0,002055 ± 0,001326	0,004519 ± 0,001055

Fonte: Elaborado pelo autor (2013)

Como descrito na seção 5.1.5.1, o *Hypervolume* avalia a distribuição (convergência) do conjunto de soluções *e*m relação ao espaço de busca. Um alto valor para essa métrica é desejável, pois representa um maior valor de volume tendo em vista a proximidade à frente de Pareto.

Os resultados obtidos pelas metaheurísticas, apresentados na TABELA 7, mostram que o NSGA-II superou o MOCell em todas as instâncias, pois apresentou os maiores valores. O algoritmo de busca aleatória apresentou os menores valores em todas as instâncias, tendo, portanto uma baixa eficácia.

Na instância A1_I.10.5.5.0.80.0, os algoritmos MOCell e NSGA-II obtiveram valor ‘zero’. Isso se deve ao fato de que não foi possível a construção de *hypercubos* a partir do ponto de referência selecionado.

Na TABELA 8 estão os resultados obtidos pela métrica *Spread*, que de acordo com a seção 5.1.5.3, objetiva mensurar a diversidade entre as soluções não-dominadas de uma população. Para uma boa solução, o algoritmo deve alcançar um baixo valor para esta métrica.

Foi verificado que as metaheurísticas obtiveram bons resultados para essa métrica. No entanto, NSGA-II foi superado pela busca aleatória na menor das instâncias (A1_I.10.5.5.0.80.0), sugerindo que o NSGA-II não possui um bom desempenho em pequenas instâncias.

Tratando apenas dos resultados das metaheurísticas, ambas obtiveram resultados semelhantes, porém o algoritmo MOCell apresentou melhor desempenho em todas as instâncias quando comparado ao algoritmo NSGA-II.

TABELA 7 – Resultados obtidos para a métrica *Hypervolume*

Nome da Instância	MOCell	NSGA-II	Busca Aleatória
A1_I.10.5.5.0.80.0	0,000000	0,000000	0,664878 ± 0,057895
A1_I.50.5.5.0.80.0	0,749023 ± 0,004788	0,782002 ± 0,002508	0,663585 ± 0,005744
A1_I.50.10.5.0.80.0	0,726069 ± 0,004395	0,758103 ± 0,002507	0,643974 ± 0,007451
A1_I.50.15.5.0.80.0	0,727749 ± 0,004243	0,764252 ± 0,001783	0,668532 ± 0,003831
A1_I.100.5.5.0.80.0	0,713165 ± 0,004789	0,744202 ± 0,003148	0,653562 ± 0,002294
A1_I.100.10.5.0.80.0	0,691796 ± 0,004335	0,728004 ± 0,002406	0,638147 ± 0,002385
A1_I.100.15.5.0.80.0	0,682588 ± 0,003791	0,715161 ± 0,002521	0,637932 ± 0,003083
A1_I.200.5.5.0.80.0	0,682889 ± 0,01557	0,695368 ± 0,018186	0,592089 ± 0,01964
A1_I.200.10.5.0.80.0	0,674735 ± 0,011103	0,683893 ± 0,022071	0,591251 ± 0,017489
A1_I.200.15.5.0.80.0	0,659053 ± 0,017563	0,667194 ± 0,021146	0,590531 ± 0,013948

Fonte: Elaborado pelo autor (2013)

TABELA 8 – Resultados obtidos para a métrica *Spread*

Nome da Instância	MOCell	NSGA-II	Busca Aleatória
A1_I.10.5.5.0.80.0	0,453465 ± 0,003081	1,838690 ± 0,007606	1,667267 ± 0,166334
A1_I.50.5.5.0.80.0	0,640596 ± 0,033830	1,084968 ± 0,048992	1,851789 ± 0,021134
A1_I.50.10.5.0.80.0	0,684641 ± 0,051718	1,070286 ± 0,044393	1,823198 ± 0,038403
A1_I.50.15.5.0.80.0	0,780767 ± 0,054566	0,985992 ± 0,050200	1,825512 ± 0,037818
A1_I.100.5.5.0.80.0	0,712338 ± 0,043063	0,946712 ± 0,066739	1,767421 ± 0,014038
A1_I.100.10.5.0.80.0	0,735133 ± 0,047144	0,997353 ± 0,047206	1,727317 ± 0,025523
A1_I.100.15.5.0.80.0	0,766453 ± 0,046589	0,922116 ± 0,036755	1,717308 ± 0,026040
A1_I.200.5.5.0.80.0	0,875219 ± 0,052048	0,984989 ± 0,069281	1,462527 ± 0,054464
A1_I.200.10.5.0.80.0	0,843077 ± 0,064536	0,898183 ± 0,075249	1,457327 ± 0,074268
A1_I.200.15.5.0.80.0	0,912450 ± 0,043070	0,979783 ± 0,048453	1,465658 ± 0,056196

Fonte: Elaborado pelo autor (2013)

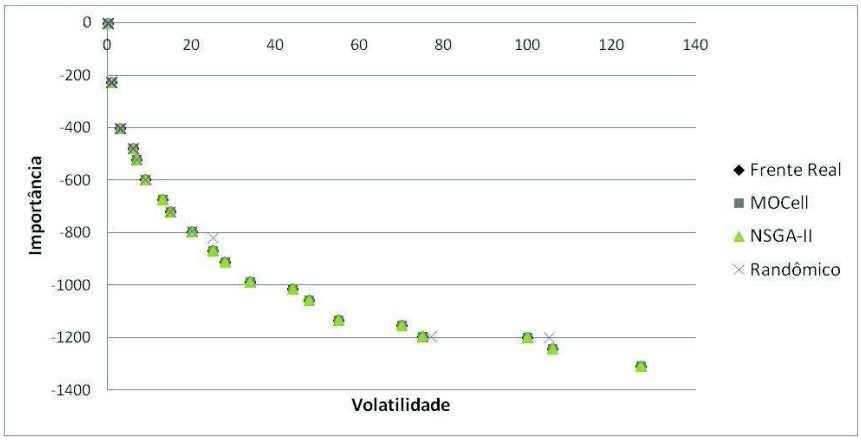
À exceção do *Hypervolume*, durante a execução das abordagens não foram observadas situações em que os algoritmos gerassem apenas uma única solução, que ocasionaria a métrica a retornar um valor médio igual a “zero” como resultado.

5.3 Representação gráfica dos resultados

A seguir são apresentadas uma parcela dos resultados obtidos. Pode-se observar a eficiência das metaheurísticas em relação ao algoritmo de busca aleatória. É possível também verificar e entender a diversidade das soluções em relação à frente de Pareto real.

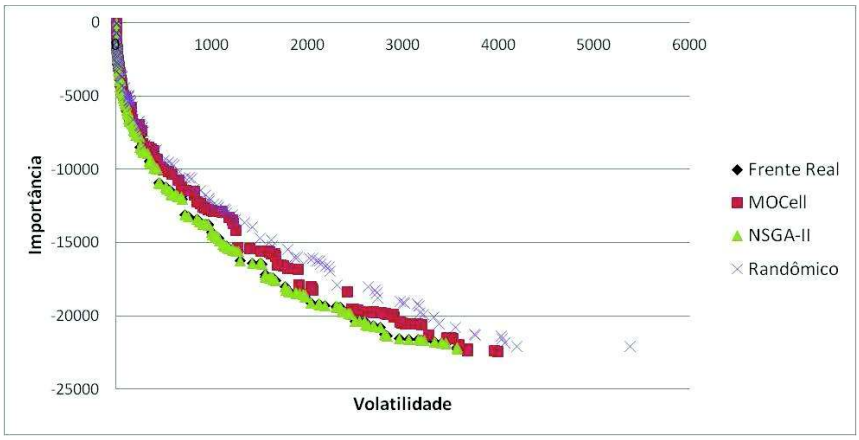
São apresentadas na FIGURA 18, FIGURA 19, FIGURA 20 e FIGURA 21 a amostra de uma das dez execuções dos dados de cada um dos grupos de instâncias classificadas de acordo com o número de soluções: 10, 50, 100 e 200.

FIGURA 18 – Resultados para a instância A1_I.10.5.5.0.80.0



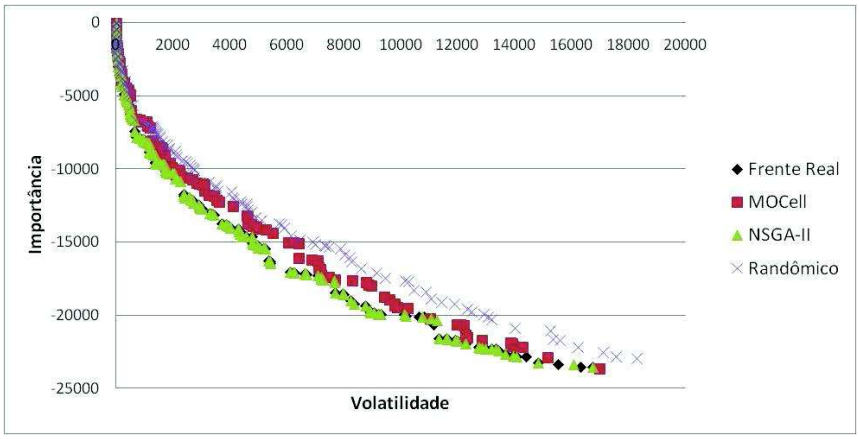
Fonte: Elaborado pelo autor (2013)

FIGURA 19 – Resultados para a instância A1_I.50.15.5.0.80.0



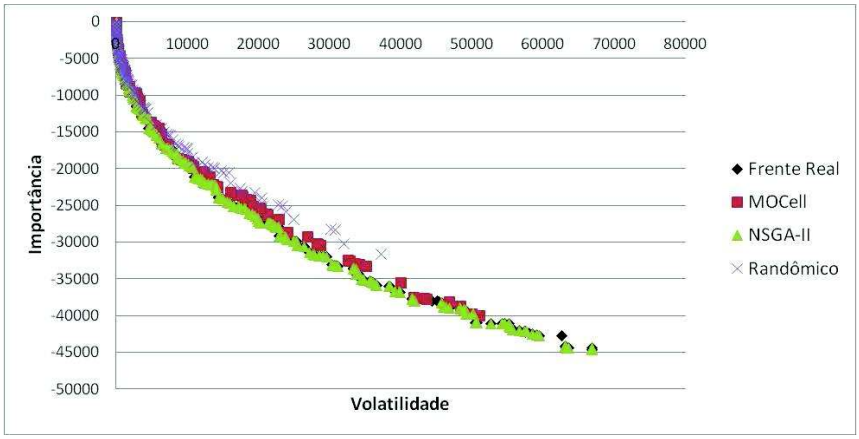
Fonte: Elaborado pelo autor (2013)

FIGURA 20 – Resultados para a instância A1_I.100.10.5.0.80.0



Fonte: Elaborado pelo autor (2013)

FIGURA 21 – Resultados para a instância A1_I.200.10.5.0.80.0



Fonte: Elaborado pelo autor (2013)

5.4 Conclusão

Neste capítulo foram apresentados os resultados para o problema do planejamento de *release* de software levando em conta a priorização segundo o grau de estabilidade (ou volatilidade) dos requisitos. Foram utilizadas as metaheurísticas NSGA-II, MOCell e ainda um algoritmo de busca aleatória para demonstrar a viabilidade da abordagem proposta. Foram apresentadas as métricas que serviram como referencial balizador para medir o desempenho das metaheurísticas. Na TABELA 9 são consolidados os resultados das avaliações de acordo com o desempenho do algoritmo, onde "+" representa o melhor resultado e "-" representa o pior resultado. Adicionalmente, "+/-" representa um resultado intermediário.

TABELA 9 – Consolidação do desempenho obtido pelos algoritmos nas avaliações

Algoritmo	Diversidade das soluções		Convergência		Tempo de execução
	Spacing	Spread	Generational Distance	*Hypervolume*	
NSGA-II	+	+/-	+	+	-
MOCell	+/-	+	+/-	+/-	+/-
Busca Aleatória	-	-	-	-	+

Fonte: Elaborado pelo autor (2013)

Com relação ao tempo de execução, em todos os cenários o algoritmo de busca aleatória, por sua característica simplista, foi superior às metaheurísticas, ou seja, apresentou o menor tempo de execução.

Como visto, para medir a diversidade das soluções pela frente de Pareto, foram utilizadas as métricas *Spacing* e *Spread*. Nessa modalidade, os algoritmos MOCell e NSGA-II obtiveram desempenhos semelhantes. Nas métricas de convergência, foram utilizadas *Generational Distance* e *Hypervolume*. O algoritmo NSGA-II obteve os melhores resultados para as duas métricas utilizadas.

Levando em consideração os resultados obtidos, as metaheurísticas foram os melhores métodos aplicados para se chegar a soluções de maior qualidade, ou seja, aquelas que mais se aproximaram da frente de Pareto real.

Vale ressaltar que o uso de mais de uma metaheurística pode trazer melhores resultados de que usadas de forma isoladas, haja vista que as melhores soluções para os problemas foram obtidas pelo MOCell ou pelo NSGA-II.

6 CONCLUSÃO

Neste capítulo, são apresentadas considerações finais do trabalho. Além disso, são mostradas as oportunidades de trabalhos futuros tendo como base a abordagem proposta.

6.1 Considerações finais

À medida que a capacidade de produzir software aumentou, também cresceu a complexidade dos sistemas de software requeridos. O modelo em releases, oriundo do desenvolvimento de software incremental, permite que os clientes recebam partes do software antecipadamente. Portanto, isso permite a atribuição de valores para os requisitos a serem desenvolvidos.

Idealmente, os requisitos aprovados pelo cliente devem ficar estabilizados ou com o mínimo de mudanças. Requisitos que mudam durante a fase de codificação tendem a maximizar a quantidade erros encontrados na fase de testes.

Desse modo, o Planejamento de Release de software deve considerar uma nova variável: a estabilidade do requisito. Requisitos voláteis são considerados um fator que pode causar problemas durante todo o ciclo de desenvolvimento do software.

Desse modo, a motivação dessa pesquisa foi a aplicação de técnicas de otimização multiobjetivo para priorizar os requisitos levando em consideração o seu grau de estabilidade, considerando esse um grande causador de problemas durante os processos de desenvolvimento, pois executar primeiro requisitos estáveis pode significar um ganho de muitas variáveis como, por exemplo, custo e tempo.

O modelo matemático multiobjetivo foi especificado e planejado para ser utilizado por metaheurísticas, tais como o NSGA-II e o MOCell, utilizados neste trabalho. Para tanto, foram criadas instâncias que simularam diversos cenários e aplicou-se sobre essas as metaheurísicas. Para avaliar os resultados, utilizaram-se as métricas de avaliação da diversidade (*Spread* e *Spacing*) e de convergência das soluções (*Hypervolume* e *Generational Distance*). Ao final, foram apresentados e analisados os resultados obtidos, além de expostos os gráficos de algumas amostras para a corroboração visual dos resultados obtidos.

Diante de exposto, considera-se que os objetivos elencados na seção 1.2 foram plenamente alcançados ao longo do trabalho. Porém, pontos limitadores desta pesquisa podem ser identificados, como a) a aplicação das abordagens para instâncias maiores e que refletissem o ambiente real de trabalho de desenvolvimento de software, b) a utilização de somente duas metaheurísticas como meio de resolução de problemas, c) a utilização de apenas

quatro métricas como agentes de medição de desempenho e a d) criação de uma plataforma que permitisse aos gerentes de projeto informar as variáveis de problemas e através da utilização da proposta deste trabalho, o aplicativo gerar as possíveis (e melhores) soluções.

Como contribuição principal, apresentamos a modelagem de um novo conceito a ser levado em consideração durante o planejamento de *releases* de software: a seleção e priorização de requisitos levando em consideração a estabilidade do requisito como critério.

6.2 Trabalhos futuros

A evolução científica ocorre por meio da agregação de novas implementações aos métodos já existentes para se criarem inovadoras soluções. Através deste trabalho é possível implementar novas ideias a fim de contribuir com soluções inovadores para a área de engenharia de requisitos e SBSE.

Entre as oportunidades para trabalhos futuros, podemos observar a) a utilização de outras metaheurísticas para a obtenção de novos resultados e posterior comparação aos resultados encontrados nesse trabalho, b) aplicação dessa abordagem para soluções com mais variáveis e que essas simulem melhor o ambiente de produção de software, c) utilização de outras métricas para corroborar os resultados encontrados neste trabalho, d) utilização da abordagem com uma maior interdependência e aumento no grau de estabilidade (escala maior) entre requisitos e por fim, f) a criação de uma plataforma funcional e usual para que se permita a utilização da estratégia deste trabalho como meio para solucionar problemas de projetos reais.

REFERÊNCIAS BIBLIOGRÁFICAS

ALLEN, J.H.; BARNUN, S.J.; ELLISON, R.J.; MCGRAW, G.; MEAD, N.R. "Software security engineering: a guide for project managers." Upper Saddle River, NJ : Addison-Wesley. 2008. 368 p.

ARROYO, J. E. C. A. Heurísticas e metaheurísitcas para otimização combinatória multiobjetivo. 2002. 256f. Tese (Doutorado em Engenharia Elétrica) – Faculdade de Engenharia Elétrica e de Computação, Universidade Estadual de Campinas, 2002.

BAGNALL. A.J.; RAYWARD-SMITH V.J.; e WHITTLEY I.M. "The next release problem." *Information and Software Technology*, vol. 43, 15 December 2001. 883-890p.

BAKER, P.; HARMAN, M.; STEINHÖFEL, K.; SKALIOTIS, A. "Search Based Approaches to Component Selection and Prioritization for the Next Release Problem." *Proceedings of the 22nd IEEE International Conference on Software Maintenance (ICSM '06)*. Philadelphia: IEEE Computer Society, 2006. 176-185p.

BARBOSA, H. J. C. "Algoritmos genéticos para a ottimização em engenharia: Uma introdução." *IV seminário sobre elementos finitos e métodos numéricos em engenharia*, Juiz de Fora, 1996.

BECCENERI, J.C. "Metaheurísticas e otimização." r.t. lac, inpe. *Computers and Operations Research*, 2007.

BERANDER, P. "Prioritization of Stakeholder Needs in Software Engineering Understanding and Evaluation." *Thesis (Licentiate of Technology in Software Engineering) - Department of Systems and Software Engineering*, Blekinge Institute of Technology, Sweden, 2004, 172p.

BOEHM, B. "Software Risk Management." *IEEE Computer Society Press*, Washington, DC, 1989.

BRASIL, M. M. A. Planejamento de *releases* de software através da aplicação de técnicas de busca multiobjetivas. 2011. Dissertação de mestrado, Universidade Estadual do Ceará, Fortaleza, CE, 2011.

BRASIL, M. M. A.; FREITAS, F. G.; SILVA, T. N.; SOUZA, J. T.; CORTÉS, M. I. "Uma Nova Abordagem de Otimização Multiobjetivo para o Planejamento de Releases Desenvolvimento Iterativo e Incremental de Software." *Anais do I Workshop Brasileiro de Otimização em Engenharia de Software (WOES'2010)*, 2010.

BURDETT, G.; RAYMOND, Li. K.-Y. "A quantitative approach to the formation of workgroups." *Proceedings of the 1995 ACM SIGCPR conference on Supporting teams, groups, and learning inside and outside the IS function reinventing IS (SIGCPR '95)*. Nashville: ACM, 1995. 202-212p.

CARPENA, F.; KIRNER, T. “Especificação de requisitos de software com o método SCR.” *Universidade Estadual de Maringá*, Maringá SP. XII SBES, Outubro 1998.

CASTRO, R.E. Otimização de estruturas com multi-objetivos via algoritmos genéticos. 2001 Tese de Doutorado, Universidade Federal do Rio de Janeiro, Rio de Janeiro, 2001.

CHRISTEL, M.; KANG, K. “Issues in Requirements Elicitation”, Carnegie Mellon University, Pittsburgh TR.CMU/SEI-92-TR-12. September, 1992.

COELLO, C.A.C. “Recent Trends in Evolutionary Multiobjective Optimization.” *Evolutionary Multiobjective Optimization Theoretical Advances and Applications*, por A. ABRAHAM, L. JAIN, R. GOLDBERG (Eds), Springer, 2005. 7-32p.

COELLO, C.A.C.; VELDHUIZEN, D. A.; LAMONT, G. B. “Evolutionary algorithms for solving multi-objective problem.” *Kluwer Academis Publishers*, 2002. 576p.

CORDEIRO, A. G.; FREITAS, A. L. P.; “Priorização de requisitos e avaliação da qualidade de software segundo a percepção dos usuários.” *Ci. Inf.*, Brasília, DF, v. 40 n. 2, maio/ago., 2011. 160-179p.

CURTIS, B.; KRASNER, H.; ISCOE, N. "A Field Study of the Software Design Process for Large Systems." *Comunications of the ACM*, vol. 31, 1988. 1268-1287p.

DAVIS, A.M. “Software Requirements.” *Prentice Hall, Englewood Cliffs*, New Jersey, 1993.

DEB, K. “Multi-objective optimization using evolutionary algorithms.” New York: John Wiley & Sons, 2001.

DEB, K. “Multi-Objective Optimization Using Evolutionary Algorithms.” New York: John Wiley & Sons, 2009.

DEB, K.; AGRAWAL, S.; PRATAB, A.; MEYARIVAN, T. “A fast and elitist multi-objective genetic algorithm: Nsga-ii.” v. 6, n. 2, 2002. 182-197p.

DURILLO, J. J.; NEBRO, A. J.; LUNA, F.; DORRONSORO, B.; ALBA, E. “jMetal: a Java Framework for Developing Multi-Objective Optimization Metaheuristics.” *TECH-REPORT ITI-2006-10*, Departamento de Lenguajes y Ciencias de la Computación, Campus de Teatinos, University of Málaga, Malaga, 2006.

FEO, T.A.; RESENDE, M.G.C. “A probabilistic heuristic for a computationally difficult set covering problem.” *Operations Research Letters*, 8, 1989. 67-71p.

FIGUEIREDO, C. M. H.; LEMOS, M. J. M.; SÁ, V. G. P.; FONSECA, G. D. “Introdução aos Algoritmos Randomizados.” *26° Colóquio Brasileiro de Matemática*, IMPA, Rio de Janeiro, 2007.

FONSECA, C.M.; FLEMING, P.J. “Genetic algorithms for multiobjective optimization: Formulation, discussion and generalization.” *Anais do Fifth International Conference on Genetic Algorithms*, San Mateo, USA, 1993. 416-423p.

FREITAS, F. G.; MAIA, C.L.B.; COUTINHO, D.P.; CAMPOS, G.A.L; SOUZA, J.T. "Aplicação de metaheurísticas em problemas da engenharia de software: revisão de literatura." *II Congresso tecnológico Infobrasil*, 2009.

FREITAS, F.G.; MAIA, C.L.B.; COUTINHO, D.P.; FRANÇA, D. T.; CAMPOS, G.A.L.; SOUZA, J.T. "Search-based Software Engineering: Revisão de Literatura." *Revista da Fa7*, Volume 7, N° 1, 2009.

GALETTI, I.; PESSOA, M. "Técnicas de Elicitação da Engenharia de Requisitos, 4. CBGDP (Congresso Brasileiro de Gestão de Desenvolvimento de Produto), organizado pela UFRGS – Gramado-RS, 2003.

GOLDBERG, D. E. "Genetic Algorithms in Search, Optimization and Machine Learning." Addison-Wesley, 1989.

GREER, D.; RUHE, G.. "Software release planning: an evolutionary and iterative approach", *Information and Software Technology*, 2004. 243-253p.

HARMAN, M. "The Current State and Future of Search Based Software Engineering." *Future of Software Engineering - FOSE '07 (IEEE Computer Society)*, 2007. 342-357p.

HARMAN, M.; JONES, B. F. "Search-based software engineering" *Information and Software Technology*, 2001. 833-839p.

HARMAN, M.; MANSOURI, S. A.; ZHANG, Y. "Search Based Software Engineering: A Comprehensive Analysis and Review of Trends Techniques and Applications." *Technical Report TR-09-03*, Department of Computer Science, King's College London, London, 2009.

HARMAN, M. "Software engineering meets evolutionary computation." Computer 44, 2011. 31-39p.

HOFFMANN, H. F. Requirements Enqineering - A Survey of Methods and Tools, Institut für Informatik der Universität Zürich. 1993.

HORN, J. "Handbook of Evolutionary Computation, volume 1." Oxford University Press, Oxford, England, 1997.

IEEE, IEEE. "Software Engineering Standards Collection." Computer Society Press, 1997.

JACKSON, M. "Software Requirements and Specifications: A Lexicon of Practice, Principles and Prejudices." 1ed. Addison-Wesley, Massachusetts, USA, 1995.

JONES, C. "Software Quality: Analysis and guidelines for Success." International Thomson Computer Press, 1997.

JUNIOR, A. S. M.; BARBOSA, S. L. "Sistemas de otimização, estudo conceitual sobre sua adoção na indústria baseado em retorno sobre investimentos e baixa imobilização de ativos", *ISA SHOW 2005 - 9º Congresso Internacional e Exposição Sul-Americana de Automação, Sistemas e Instrumentação*, São Paulo, 2005.

KANG, K.C.; CHRISTEL, M.G. “Issues in requirements elicitation.” *Tech. Rep. No. SEI-92-TR- 012*. Carnegie Mellon University, Pittsburgh, EUA, 1992.

KARLSSON, J.; RYAN, K. “A Cost-Value approach to prioriziting requirements”, *IEEE Software,* 1997. 67-74p.

KARLSSON, J.; RYAN, K. “Supporting the selection of Software Requirements.” *In: INTERNATIONAL WORKSHOP ON SOFTWARE SPECIFICATION AND DESIGN (IWSSD ‘96)*, 8th. Proceedings, 1996. 146-149p.

KARLSSON, J.; WOHLIN, C.; REGNELL, B. “An Evaluation of Methods for Prioritizing Software Requirements.” *Information and Software Technology*, 1998. 939-947p.

KOTONYA, G.; SOMERVILLE, I. “Requirementes Engineering: Processes and Techniques.” John Wiley & Sons, Chichester, 1997.

LAARHOVEN, P. J. M.; AARTS, E. H. L. “Simulated Annealing: Theory and applications.” Kluwer Academic Publishers, 1987.

LAMSWEERDE, A. V. “Requirements Engineering in the Year 00: A Research Perspective”, *22nd Proceedings of International Conference on Software Engineering, Limerick*, Ireland. June, 2000.

LUTZ, R.R. "Analyzing Software Requirements Errors in Safety-Critical Embedded Systems." *Proceedings of the ACM SIGSOFT Symposium on the Foundations of Software Engineering*, New York, NY, December 1993. 126-133p.

MACAULAY, L. “Requirements Engineering”, Springer, London, 1996.

MAIDEN, N.; RUGG, G. “ACRE: Selecting Methods for Requirements Acquisition.” *Software Engineering Journal*, 11, 5, 1996. 183-192p.

MCCALLUM, J.C.J. “A generalized upper bounding approach to communications network planning problems.” *Networks, v. 7, n. 1*, 1977. 1-23p.

NURMULIANI, N.; ZOWGHI D.; FOWELL S. “Analysis of requirements volatility during software development life cycle.” *Australian Software Engineering Conference (ASWEC’04)*, 2004.

PEREIRA, G. W. Aplicação da técnica de recozimento simulado em problemas de planejamento florestal multiobjetivo. 2004. Dissertação de mestrado, Universidade Federal de Minas Gerais, Belo Horizonte, MG, 2004.

PRESSMAN, R. “Software Engineerin.” 8ª ed. Pearson: Essex, England, 2007.

REGO, M. Algoritmos Multiobjetivos para o Problema de Sequenciamento de Tarefas em Uma Máquina com Tempo de Preparação Dependente da Sequência e da Família. 2013 Dissertação de Mestrado, Universidade Federal de Ouro Preto, Ouro Preto, MG, 2013.

Repositório SEBASE. 2011. http://www.sebase.org/sbse/publications/ (acesso em Junho de 2013).

ROZANSKI, N.; WOODS, E. “Software Systems Architecture: Working With Stakeholders Using Viewpoints and Perspectives.” Addison-Wesley Professional, 2005.

RUHE, G.; SALIU, M. O. “The Art and Science of Software Release Planning.” IEEE Software, Nov./Dec. de 2005. 47-53p.

SADRAEI, E.; AURUM, A.; BEYDOUN, G.; PAECH, B. “A field study of the requirements engineering practice in Australian software industry.” *Requirements Engineering.* v. 12, 2007. 145-162p.

SAGRADO, J. D.; AGUILA, I. M. D. “Ant Colony Optimization for Requirement selection in Incremental Software development.” *Proceedings of the 1st International Symposium on Search Based Software Engineering (SSBSE '09)*. Windsor: IEEE Computer Society, 2009. 67-76p.

SANTANA, R. A.; PONTES, M. R.; BASTOS-FILHO, C. J. A. A multiple objective particle swarm optimization approach using crowding distance and roulette wheel. *ISDA ’09: Proceedings of the 2009 Ninth International Conference on Intelligent Systems Design and Applications*. Pisa, Itália: IEEE, 2009.

SANTOS, K. C. L. “Meta-heurísticas mono e multi-objetivo para o planejamento de redes celulares de terceira geração”. *XXXVII Simpósio brasileiro de pesquisa operacional*. Gramado-RS, 2005.

SMITH, P. “Effective Requirements Definition and Management.” **Read**, v. 2 n. 2, April 2006, Disponível em: <http://www.borland.com/resources/en/pdf/solutions/rdm_whitepaper.pdf>. Acesso em: 01 fev. 2012.

SOARES, J. L. “Optimização matemática.” Maio 2005, Disponível em: <http://www.mat.uc.pt/~jsoares/research/opti_2005_05_06.pdf>. Acesso em 10 jun. 2013.

SOMMERVILLE, I. “Software Engineering.” 7ª ed. São Paulo: Pearson Addison-Wesley, 2004.

SOMMERVILLE, I. “Software Engineering”. 6ª ed. São Paulo: Pearson Addison-Wesley, 2003.

TSUI, F.; KARAN, O. “Engenharia de requisitos.” 2ª edição. Rio de Janeiro: LTC, 2013.

VELDHUIZEN, D. A. V.; LAMONT, G. B. “Evolutionary computation and convergence to a pareto front.” *In: Stanford University*, California, Morgan Kaufmann, 1998. 221-228p.

VERGILIO, S.; COLANZI, T. E.; POZO, A. T. R.; ASSUNCAO, W. K. G. “Search Based Software Engineering: A Review from the Brazilian Symposium on Software Engineering.”

CbSoft 2011, XXV Simpósio Brasileiro de Engenharia de Software (SBES 2011), 2011. 49-54p.

VON ZUBEN, F. J. V. "Computação Evolutiva: Uma Abordagem Pragmática." UNICAMP, 2000.

YANG, B.; HUAJUN, H.; JUN, Z. "Optimal Software Release Time Determination with Risk Constraint", on page(s): 393, E-ISBN: 978-1-4244-1461-1, Print ISBN: 978-1-4244-1460-4, Issue Date: 28-31 Jan. 2008.

ZAVE, P. "Classification of Research Efforts in Requirements Engineering." *ACM Computer Surveys*, vol 29, n° 4, 1997.

ZHANG, Y.; FINKELSTEIN, A.; HARMAN, M. "Search Based Requirements Optimisation: Existing Work & Challenges." *Proceedings of the 14th international conference on Requirements Engineering: Foundation for Software Quality (REFSQ '08)*. Montpellier: Springer-Verlag, 2008. 88-94p.

ZITZLER, E.; THIELE, L. "Multiobjective optimization using evolutionary algorithms - a comparative case study." *Parallel problem solving from nature-PPSN V*, Springer, 1998. 292-301p.

ZOWGHI, N. "A study of the Impact of requirements volatility on Software Project Performance." *Proceedings of the Ninth Asia-Pacific Software Engineering Conference, APSEC 2002*, Gold Cost, Queensland, Australia, 2002. 3-11p.

APÊNDICE A – Detalhamento do conteúdo da instância A1_I.10.5.5.0.80.0

```
#Número de requisitos
10

#Número de clientes
5

#Custo total do projeto
125

#Importância dos clientes para a empresa desenvolvedora de software
2 10 4 2 8

#Custo de cada requisito
20 14 16 19 10 13 15 16 19 15

#Estabilidade de cada requisito
1 5 4 2 7 10 7 6 1 9

#Precedência técnica entre os requisitos
0
0
0
0
0
0
0
0
0
1 4 //Só pode ser executado após a implementação do requisito 4

#Importância dos requisitos para os clientes
3 1 1 3 10 6 1 1 8 3
1 10 10 7 3 5 7 1 4 6
1 6 10 2 4 6 9 1 9 2
8 3 2 6 9 1 7 3 2 3
5 8 10 3 7 7 9 3 10 4
```