Universidade FUMEC
Faculdade de Ciências Empresariais
Programa de Pós-Graduação em Sistemas de Informação e Gestão do Conhecimento

Otimização Multiobjetivo Usando Algoritmos Evolutivos na Alocação de Times Ágeis

Júnea Eliza Brandão Caldeira

Belo Horizonte
2019

Júnea Eliza Brandão Caldeira

Otimização Multiobjetivo Usando Algoritmos Evolutivos na Alocação de Times Ágeis

Dissertação de mestrado apresentada ao Programa de Pós-Graduação em Sistemas de Informação e Gestão do Conhecimento como parte dos requisitos para a obtenção do título de Mestre em Sistemas de Informação e Gestão do Conhecimento. Área de concentração: Gestão de Sistemas de Informação e do Conhecimento. Linha de pesquisa: Tecnologia e Sistemas de Informação.

Orientador: Prof. Dr. Fernando Silva Parreiras

Belo Horizonte

2019

Dados Internacionais de Catalogação na Publicação (CIP)

C146o Caldeira, Júnea Eliza Brandão, 1974 -

Otimização multiobjetivo usando algoritmos evolutivos na alocação de times ágeis / Júnea Eliza Brandão Caldeira. – Belo Horizonte, 2019.
109 f : il. ; 29,7 cm

Orientador: Fernando Silva Parreiras
Dissertação (Mestrado em Sistemas de Informação e Gestão do Conhecimento), Universidade FUMEC, Faculdade de Ciências Empresariais, Belo Horizonte, 2019.

1. Recursos humanos - Brasil. 2. Engenharia de software - Brasil. 3. Algoritmos genéticos. 4. Projetos de sistemas - Metodologia. I. Título. II. Parreiras, Fernando Silva. III. Universidade FUMEC, Faculdade de Ciências Empresariais.

CDU: 65.012.2

Ficha catalográfica elaborada pela Biblioteca Universitária-FUMEC

UNIVERSIDADE
FUMEC

Dissertação intitulada "**Otimização Multi-Objetivo Usando Algoritmos Evolutivos na Alocação de Times Ágeis**" de autoria de Júnea Eliza Brandão Caldeira, aprovada pela banca examinadora constituída pelos seguintes professores:

Prof. Dr. Fernando Silva Parreiras – Universidade FUMEC
(Orientador)

Prof. Dr. Ronaldo Darwich Camilo – Universidade FUMEC
(Examinador Interno)

Prof. Dr. Humberto Torres Marques Neto – PUC Minas
(Examinador Externo)

Juliano Silveira Alves, Esp. – TOTVS S.A.
(Consultor *Ad Hoc*)

Prof. Dr. Fernando Silva Parreiras
Coordenador do Programa de Pós-Graduação em Sistemas de Informação e Gestão do Conhecimento da Universidade FUMEC

Belo Horizonte, 20 de março de 2019.

Dedicado aos meus pais, meus queridos filhos, Tiago e Henrique, e ao meu esposo, Hugo.

AGRADECIMENTOS

Em primeiro lugar, agradeço ao meu orientador Professor Dr. Fernando Silva Parreiras, pelo conhecimento compartilhado.

Agradeço à Faculdade COTEMIG e a TOTVS por me proporcionarem esta grande oportunidade de crescimento profissional e pessoal.

Agradeço aos meus colegas de trabalho que me apoiaram e tornaram possível a realização deste trabalho.

Agradeço aos colegas do LAIS, pelo apoio e incentivo e ao mestre Sérgio Roberto Imaeda Yoshioka, pela parceria.

Por fim, agradeço a minha família, por sempre estarem ao meu lado. Sou muito grata aos meus filhos e esposo, que todo o tempo me apoiaram e acreditaram no meu propósito.

Resumo

O desenvolvimento de *software* é uma atividade dinâmica e complexa, que não é determinada apenas pela escolha adequada das tecnologias e metodologias, mas também pelo conhecimento e habilidades das pessoas envolvidas. Garantir que o time atenda aos requisitos dos projetos é essencial para assegurar a qualidade do produto disponibilizado no mercado e consequentemente o sucesso do projeto. Neste contexto, o estudo avaliou três técnicas de otimização de problemas da Engenharia de *Software*, *NSGAII*, *SPEA2* e *MOCell*, quanto a capacidade de apoiar os gestores de projetos na composição dos times ágeis de desenvolvimento de *software*. Inicialmente foi realizada uma revisão sistemática de literatura para identificar as técnicas adequadas, no domínio da SBSE, para o problema proposto e posteriormente as técnicas selecionadas foram testadas em um experimento realizado em uma empresa de desenvolvimento de *software*, em que foram avaliados quatro projetos executados recentemente pela empresa. A abordagem levou em consideração as características das atividades do projeto, os recursos humanos disponíveis, o perfil dos recursos humanos, as restrições do projeto (escopo e tempo para execução) e as restrições estabelecidas pela organização. As técnicas estudadas retornaram soluções com o número de recursos necessários para a realização do projeto, bem como os recursos como mais qualificação para projeto, levando ao resultado de menor custo e produtividade adequada para o prazo estabelecido pelo cliente. Uma ferramenta de apoio à decisão foi implementada e usada no experimento. Os resultados mostraram que as três técnicas avaliadas apresentaram desempenhos consistentes. O *NSGAII* e o *SPEA2* tiveram resultados e comportamento bem similares. Já o *MOCell* mostrou um melhor desempenho em esforço computacional e necessitou de uma população maior para sua saturação.

Palavras-chave: Métodos ágeis. Alocação de recursos humanos. *Search-based software engineering* - SBSE. . Engenharia de *Software*. Algoritmo Genético.

Abstract

Software development is a dynamic and complex activity that is determined not only by the appropriate choice of technologies and methodologies but also by the knowledge and skills of the people involved. Ensuring that the team meets the requirements of the projects is essential to ensure the quality of the product available in the market and consequently the success of the project. In this context, the study evaluated three techniques for optimizing software engineering problems, NSGAII, SPEA2 and MOCell, so that they can support project managers in the composition of agile software development teams. Initially, a systematic literature review was carried out to identify the appropriate techniques for the proposed problem, and later the selected techniques were tested in an experiment carried out in a software development company, in which four projects recently executed by the company were evaluated. The approach took into account the characteristics of the project activities, available human resources, human resource profile, project constraints (scope and time for execution) and constraints established by the organization. As a result, the techniques returned solutions with the number of resources needed to carry out the project, as well as resources such as more project qualification, lower cost and productivity adequate for the term established by the client. A decision support tool was implemented and used in the experiment. The results showed that the three techniques evaluated presented consistent performances. NSGAII and SPEA2 had very similar results and behavior. Already the MOCell showed a better performance in computational effort and needed a larger population for its saturation.

Key-words: Agile methods. Allocation of human resources. Search-based software engineering - SBSE. Software Engineering. Genetic Algorithm.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

cGA	*Algoritmos genéticos celulares*
EA	*Algoritmo evolutivo*
EMO	*Problema de otimização multiobjetivo evolutivo*
EP	*Programação evolutiva*
ES	*Engenharia de Software*
EV	*Estratégia de evolução*
GA	*Algoritmo genético*
GP	*Programação genética*
KM	*Knowledge Management*
KSA	*Knowledge, Skills, Attitudes*
MOCELL	*MultiObjective Cellular genetic algorithm*
MOEA	*Algoritmo de otimização multiobjetivo evolutivo*
MOP	*Problema de otimização multiobjetivo*
MPO	*Master Product Owner*
NSGAII	*Non-dominated Sorting Genetic Algorithm*
PO	*Product Owner*
SBO	*Search-Based Optimization*
SBSE	*Search Based Software Engineering*
SM	*Scrum Master*
SPEA2	*Strength Pareto Evolutionary Algorithm*
QA	*Quality Assurance*

Sumário

1 INTRODUÇÃO

Os avanços tecnológicos têm possibilitado que cada vez mais pessoas e empresas participem do mercado globalizado (1), no qual o aumento de competidores significa que as organizações precisam se adaptar a novas exigências do mercado em tempo real (2). Logo, as empresas têm buscado mais eficiência, produtividade, redução de custos (3) e, para alcançar estes objetivos, é essencial gerenciar adequadamente seus recursos. Auferir e entender o desempenho dos times nesses ambientes em constantes mudanças representa um desafio (2). Nesse mercado altamente competitivo, os projetos de *software* têm que garantir o orçamento definido, cobrir o maior número de requisitos de cliente e minimizar os esforços de produção (em tempo e custo) (4).

O desenvolvimento de *software* pode ser caracterizado como um trabalho complexo, cooperativo, sujeito a rápidas mudanças (5) e que as habilidades possuídas pelos desenvolvedores influenciam diretamente na produtividade dos times e na qualidade no produto disponibilizado no mercado (6). As organizações de desenvolvimento de *software* sobrevivem em um mercado competitivo, transformando o potencial de seus desenvolvedores em *softwares* que agregam valor aos negócios dos seus clientes (6).

Uma abordagem que vem sendo adotada no desenvolvimento de *software*, considerando o alto grau de mudanças e incertezas desses projetos, é a metodologia ágil. Existem pesquisas que comprovam que o emprego de métodos ágeis em circunstâncias certas resultam em projetos de baixo risco e, finalmente, melhor produtividade e qualidade (7). Métodos ágeis de desenvolvimento de *software* seguem um estilo de desenvolvimento iterativo e incremental, que de forma colaborativa as equipes se auto-organizam e se ajustam dinamicamente às mudanças nos requisitos do cliente (8).

Pesquisadores na área de Engenharia de *Software* têm concentrado suas atenções em encontrar maneiras de fornecer às empresas de desenvolvimento de *software* várias ferramentas e tecnologias com o objetivo de reduzir falhas no seus projetos (9). Esse problema da Engenharia de *Software*, mais especificamente a alocação de recursos no projeto de desenvolvimento de *software*, é um típico problema de otimização (10) conhecido como *Search-based Software Enginering* - SBSE. SBSE é o nome dado a um campo de pesquisa na qual pesquisas da computação (técnicas de otimização mais usualmente associadas à pesquisa operacional) realizadas para endereçar esses problemas (11).

A SBSE é uma reformulação dos típicos problemas de Engenharia de *Software*, que utiliza métodos de meta-heurística para encontrar soluções (12) para qualquer problema que represente um conjunto de fatores que se competem e em que não haverá uma única solução. Em vez disso, é provável que haja várias soluções. Nesse espaço implícito de

escolhas, técnicas de busca meta-heurística são ideais para peneirar soluções potenciais para aqueles que contêm um balanceamento ideal de fatores (12). Os profissionais da SBSE aplicam uma variedade de técnicas, entre elas: algoritmos genéticos e evolutivos, montanhistas - *hill-climbers*, *tabu search*, otimização de enxame de partículas e otimização de colônias de formigas - *ant colony optimization* (12).

Algoritmos evolutivos são técnicas relativamente novas, mas poderosas, usadas para encontrar soluções para muitos problemas de busca e otimização. Muitos desses problemas têm múltiplos objetivos, o que leva à necessidade de obter um conjunto de soluções, conhecidas como soluções eficazes. Verificou-se que o uso de algoritmos evolutivos é uma maneira de encontrar várias soluções eficazes em uma única execução de simulação (13).

Nesse contexto, este trabalho investigou técnicas computacionais aplicáveis na otimização de problemas de alocação de recursos humanos e competências em times ágeis, considerando multiobjetivos, para orientar a formação e manutenção de equipes de alto desempenho.

1.1 Problema de pesquisa

As práticas ágeis de desenvolvimento de *software* atingem agilidade ao promover equipes auto-organizadas, colaboração com o cliente, melhor qualidade, menos documentação e redução do tempo de lançamento do *software* no mercado (14). Como os métodos ágeis dependem substancialmente de colaboração e comunicação, a equipe é fundamental para o seu sucesso (15). As equipes são organizadas para lidar com a complexidade e pressões de prazos durante o desenvolvimento do projeto e garantir a sua conclusão. Eles próprios tomam decisões e se adaptam de acordo com a mudança de situações. Equipes auto-organizadas fazem um trabalho muito melhor de utilizar talentos da equipe porque mais mentes estão envolvidas em qualquer atividade (14).

Uma das limitações dos métodos atuais de planejamento é a falta de um processo sistemático para equilibrar a entrega apropriada de funcionalidades com a seleção de recursos humanos que possuam as competências necessárias para a realização das atividades relacionadas (16). A produtividade, por exemplo, pode variar significativamente entre os desenvolvedores, pois estes possuem produtividade diferente em diferentes tipos de tarefas (16). Menos desenvolvedores competentes mas experientes, podem entregar os resultados desejados em menos tempo e com melhor qualidade. Desenvolvedores experientes podem reduzir consideravelmente o tempo de desenvolvimento do *software* em comparação aos desenvolvedores inexperientes (14). A adequada alocação de recursos humanos, isto é, selecionar os desenvolvedores mais qualificados para as tarefas que compõem o projeto, assume ainda mais importância (16) e pode garantir a entrega nos prazos e custos previstos e especialmente com o nível de qualidade exigido.

Nas empresas que se utilizam de métodos empíricos para compor as equipes que atuam nos projetos de desenvolvimento de *software*, a composição da equipe é definida com base na observação dos gestores, que podem muitas vezes não levar em conta todo o conhecimento, habilidades, atitudes, desempenho e o custo relacionado a cada membro da equipe. Quando o objetivo é analisar times, muitos fatores devem ser considerados: fatores relacionados ao projeto, processos, produto, organização e fatores humanos (17). Assim, monitorar as equipes e garantir a correta alocação dos recursos humanos nos projetos é fundamental para manter a sua adequada evolução e seu sucesso. A partir do mapeamento de competências do time de desenvolvimento de *software* (perfil do time), os gestores e líderes do projeto necessitam avaliar as alocações dos times que atuam nos projetos em curso, bem como as alocações de times para novos projetos, permitindo que a empresa melhore sua eficiência e assertividade dos projetos.

Neste trabalho foi realizada uma revisão sistemática da literatura com o objetivo de identificar as principais técnicas computacionais aplicáveis em problemas relacionados à alocação de times ágeis de desenvolvimento de *software*. Com base nas técnicas identificadas foi utilizada a *framework* conhecida como JMetal, que implementa as respectivas técnicas. Na sequência, foi realizado um experimento em uma empresa de desenvolvimento de *software* de grande porte. Nesse experimento foram testados quatro projetos realizados recentemente pelos times. Os resultados apresentados para cada um dos algoritmos avaliados foram comparados e estão registrados no capítulo 4 deste trabalho.

Diante do exposto, tem-se a seguinte questão de pesquisa: qual o desempenho comparado das técnicas de otimização na alocação de times ágeis em projetos de desenvolvimento de *software*?

1.2 Objetivos

Nesta seção, apresentaremos os objetivos, geral e específicos, que orientaram a construção desta pesquisa.

1.2.1 Objetivo geral

O objetivo principal desta pesquisa é comparar o desempenho de algoritmos de otimização na alocação de times ágeis que atuam em projetos de desenvolvimento de *software*.

1.2.2 Objetivos específicos

Como objetivos específicos tem-se:

1. Identificar as técnicas computacionais que auxiliem na alocação de times ágeis que atuam no desenvolvimento de *software*;

2. Comparar o desempenho das técnicas identificadas em um cenário real de times ágeis de desenvolvimento de *software*.

1.3 Justificativa

A indústria de *software* tem buscado evoluir seus processos e ferramentas com o objetivo de disponibilizar produtos de alta qualidade, com tecnologia de ponta e ao maior custo benefício. Por outro lado, os clientes estão cada vez mais exigentes e esperam que os produtos sejam inovadores e que agreguem valor ao seu negócio. O sucesso dessas organizações de *software* - especialmente aquelas que aplicam métodos ágeis - depende da sua capacidade de facilitar a melhoria contínua dos seus produtos, a fim de reduzir o custo, o esforço e *time-to-market*, mas também para restringir o aumento da complexidade e do tamanho dos sistemas de *software* (18). Atualmente, o desenvolvimento de *software* é uma atividade altamente dinâmica e complexa, que não é determinada apenas pela escolha adequada das tecnologias e metodologias, mas também pelo conhecimento e habilidades das pessoas envolvidas (18).

Neste estudo, o foco foi o planejamento de projetos, mais especificamente a otimização da composição de times de desenvolvimento de *software*. Ele consistiu na comparação de técnicas de otimização capazes de determinar o número de recursos humanos necessárias para o projeto de desenvolvimento de *software*, bem como as possíveis composições dos times para os projetos (dada a natureza das atividades do projeto, seu escopo e o tempo para sua realização), considerando sua execução com o time mais qualificado, com baixo custo e produtividade compatível com o prazo de entrega do projeto. Para as empresas de *software* se manterem competitivas no mercado elas precisam buscar maximar os resultados do seus projetos. Nesse âmbito, ter uma equipe de alta *performance*, corretamente alocada e com as competências adequadas aos desafios experimentados em seus projetos é essencial.

Há, portanto, a necessidade de se disponibilizar ferramentas que apoiem os gestores de projetos na composição dos times mais adequados e que garantam a entrega dos projetos de *software* dentro dos objetivos definidos.

1.4 Aderência ao programa

O estudo apresenta conceitos relacionados aos processos de Engenharia de *Software*, métodos e práticas ágeis e explora a aplicação de algoritmos de otimização, disponíveis na literatura, no campo do gerenciamento de projetos. Além disso, a pesquisa vem

preencher uma lacuna na literatura que carece de estudos empíricos com base na investigação do desempenho de algoritmos de otimização aplicados em problemas de Engenharia de *software*, mais especificamente no planejamento do projeto.

Esta é uma pesquisa interdisciplinar, abordando aspectos da Engenharia de *Software* e Sistemas de Informação. Os resultados poderão ser utilizados por pesquisadores e profissionais de Engenharia de *Software* e gestores interessados em alcançar melhores resultados em seus projetos, a partir da gestão do conhecimento dos times e lições aprendidas com os projetos executados, aumentando a assertividade e transparência em sua gestão.

Por um lado, os pesquisadores poderão usar esse material como uma fonte para identificar oportunidades de pesquisa. Por outro lado, engenheiros de *software* podem basear-se nos resultados apresentados para identificar necessidades de adaptação e possibilidades de utilização nas suas empresas, contribuindo para melhor eficiência nos seus projetos.

1.5 Estrutura do documento

Este trabalho está estruturado em seis capítulos: no capítulo 1 é apresentada a Introdução - em que estão descritos o problema de pesquisa, objetivos e motivação e aderência ao programa. No capítulo 2 relata-se a primeira pesquisa, que é uma revisão sistemática da literatura (RSL) sobre a técnica de *Search-based Software Engineering* adotada na otimização de problemas da Engenharia de *Software*, bem como todo o referencial teórico, incluindo os conceitos de metodologia ágil, algoritmos de otimização, a importância do planejamento adequado dos projetos de desenvolvimento de *software* e a metodologia utilizada para a realização da RSL e do experimento. No capítulo 3 descrevem-se o experimento realizado e a metodologia utilizada. No capítulo 4 são apresentados os resultados obtidos a partir do experimento. No capítulo 5 listam-se os trabalhos relacionados ao tema aqui pesquisado. Por fim, o capítulo 6 contém as considerações finais deste trabalho.

2 REVISÃO SISTEMÁTICA DA LITERATURA

2.1 Considerações iniciais

Para identificar as técnicas computacionais do campo de pesquisa do SBSE, aplicadas na otimização de processos da Engenharia de *Software*, foi realizada revisão sistemática da literatura seguindo as diretrizes propostas por Kitchenham (2004) (19).

Portanto, os objetivos deste estudo são: (i) levantar a literatura sobre aplicação de SBSE na Engenharia de *Software* e proporcionar compreensão sobre os tipos de pesquisas utilizadas nessas áreas; (ii) identificar as técnicas computacionais aplicáveis na solução do problema apresentado; (iii) identificar os algoritmos adotados nos estudos.

2.2 Fundamentação teórica

As metodologias ágeis de desenvolvimento de *software* têm o objetivo de otimizar o desenvolvimento de *software* (17). As práticas ágeis de desenvolvimento de *software* alcançam a agilidade promovendo times auto-organizados, a colaboração dos clientes, alta qualidade, menos documentação e redução do tempo de entrega da solução (14). Embora tenha começado anos antes, o movimento ágil iniciou oficialmente com a criação do Manifesto Ágil em fevereiro de 2001. Esse manifesto foi escrito e assinado por 17 "metodologistas leves", como eram chamados no momento. Os autores do Manifesto Ágil escreveram que eles valorizam (20):

- Indivíduos e interações mais que processos e ferramentas.
- *Software* em funcionamento mais que documentação abrangente.
- Colaboração com o cliente mais que negociação de contratos.
- Resposta a mudanças mais que seguir um plano.

Os métodos ágeis utilizam várias técnicas para lidar com as mudanças constantes nos requisitos de *software*. As mais notáveis são: (i) planejamento simples; (ii) iterações curtas; (iii) *releases* entregues mais cedo; e (iv) *feedbacks* frequentes dos clientes. Essas características permitem que os métodos ágeis entreguem os produtos mais rapidamente, em comparação à abordagem em cascata (21).

Nesta seção será explorado o papel de cada membro do time ágil, a relevância das pessoas e da colaboração entre elas nos projetos que adotam práticas ágeis, as principais metodologias ágeis adotadas pelo mercado, os critérios de qualidade de um *software* e as técnicas de otimização utilizadas na solução de problemas da Engenharia de *Software*.

2.2.1 A gestão ágil

No desenvolvimento ágil, é necessário ser otimista em relação às pessoas e dar-lhes espaço para trabalhar e realizar suas tarefas da melhor maneira, possibilitando que tomem as decisões profissionais corretas sobre o seu trabalho e garantam que o cliente esteja bem representado na equipe durante todo o projeto (18). O time é algo crítico em projetos ágeis (22). Pode haver uma variedade de pessoas envolvidas em um esforço de *software* - desenvolvedores, testadores, líderes de projeto, entre outros. Muitas vezes, há um cliente e um usuário final que querem o produto resultante. Há também gerentes executivos que estão interessados no orçamento, retornos de investimento e recursos humanos. Cada um deles tem um interesse no projeto ágil (15). A seguir estão descritos o papel e relevância de pessoas que participam do esforço de *software*.

Desenvolvedores: desenvolvedores escrevem testes e mantêm o código do programa tão simples e definido quanto possível (23). A abordagem ágil para gerenciar projetos de *software* baseia-se em dar mais valor aos desenvolvedores do que ao processo. Isso significa que a gerência deve se esforçar para tornar o ambiente de desenvolvimento propício (18). Os desenvolvedores devem ser dispostos a trabalhar em equipe, serem capazes de lidar com mudanças e engenhosos o suficiente para resolverem problemas (15). Os métodos ágeis são técnicas muito leves, não fornecendo diretrizes e processos para desenvolvedores seguir. Esse é um risco de gestão em que alguns desenvolvedores podem não se encaixar nesse ambiente ágil (15).

Testadores: os testadores verificam se o sistema que está sendo produzido atenderá aos requisitos do cliente. Pode ser uma equipe independente da equipe do projeto (23). O impacto nos testes (ou garantia de qualidade) do uso de um método ágil na organização está relacionado aos ciclos de desenvolvimento mais curtos, em que o teste ocorre durante todo o processo de desenvolvimento de *software*. Os testadores devem trabalhar em estreita colaboração com os desenvolvedores durante a escrita do código. Em métodos ágeis, recomenda-se que os testes sejam alterados antes do código ser modificado pelos desenvolvedores e o papel de um testador é significativamente reduzido (15). O desafio do gerente de projetos é realocar testadores que não se encaixam mais no grupo ágil e encontrar testadores com habilidades apropriadas de desenvolvimento/ teste. Isso representa uma oportunidade de iniciação para os desenvolvedores novatos ganharem experiência no sistema e no método ágil. Tal abordagem requer desenvolvedores mais experientes (15).

Líderes de projetos: existem duas funções-chave do líder de projeto de desenvol-

vimento de *software* - gerentes de projeto e líderes de equipe. Cada um tem um conjunto diversificado de desafios de gestão na metodologia ágil, que difere de outras metodologias. Esta distinção é bem caracterizada como líder de pessoas e gerenciamento de recursos do processo (15). Uma vez que equipes ágeis envolvem pessoas experientes com responsabilidade considerável, uma abordagem de mentoria ou liderança de *coach* é mais eficaz. Os líderes da equipe devem estar dispostos a permitir que os membros tomem a iniciativa. A liderança é feita via colaboração em vez de comando e liderança do tipo controle. Isso pode representar uma mudança cultural para alguns, uma vez que eles devem estar dispostos a compartilhar a autoridade decisória. O trabalho de um líder de equipe é facilitar a tomada de decisão pela equipe (15).

Em contraste, os gerentes de projeto, em processos ágeis, são responsáveis por acompanhar o progresso e tomar decisões de negócio. Os gerentes de projeto têm necessidade de ajustes de equipe. A ênfase é dada em responder as mudanças, em vez de seguir um plano específico. Isso apresenta um desafio, uma vez que eles são geralmente chamados para detalhar o *status* do projeto (15). Os gerentes de projeto também têm muito mais função de engajar o time e os clientes visando à colaboração no projeto, em vez do foco usual na definição de produtos e contratos de serviços (15).

Clientes: o impacto dos métodos ágeis é ter clientes mais envolvidos do que o métodos habituais. Clientes em metodologias tradicionais podem estar envolvidos no início do projeto - ajudando a definir requisitos e obrigações contratuais - e no final do projeto com testes alfa, beta e de aceitação (15). Os clientes em métodos ágeis estão envolvidos com mais frequência e com mais influência (15). Eles ajudam na definição das prioridades para os requisitos a serem desenvolvidos ou evoluídos, apoiando na sua elaboração e na definição das restrições do produto final desejado.

Gestão executiva: como o apoio da gerência executiva na seleção de qualquer novo processo organizacional é essencial, para os métodos ágeis, isso é particularmente desafiador. Os gerentes executivos são focados em riscos e oportunidades - relutantes em induzir risco sem visibilidade. Para justificar despesas, eles querem datas de entrega comprometidas para funcionalidade específica, progresso em tarefas e detalhamento de horários e planos (15). Métodos ágeis representam uma importante mudança cultural para eles. Com pouca documentação para acompanhar o progresso, os recursos em uma determinada versão podem mudar conforme o processo ágil prossegue no curso (15). Além disso, a estimativa de custo para um projeto, com um conjunto de funções específicas, é difícil em métodos ágeis. Uma vez que os requisitos não são fixos, não há como saber, *a priori*, o que estará em um produto acabado e, portanto, quando pode ser terminado. A gerência executiva é confrontada com a impossibilidade de garantir datas de entrega, custo ou funcionalidade - uma situação que é contraditória à maioria das abordagens de gestão. Isso pode ser uma situação insustentável para a gestão e dificultar a adoção de

um método ágil na organização (15).

A chave para um gerente de projeto é convencer os executivos de que os métodos ágeis fornecerão mais agilidade com qualidade. Se a gerência executiva estiver disposta a experimentar, a taxa de sucesso de projetos usando os métodos ágeis determinará seu uso continuado. Para aqueles gerentes de projeto que constroem a confiança com a gestão executiva, para aplicar métodos ágeis inicialmente em projetos apropriados e ganhar sua confiança, o retorno de muitos projetos pode ser alto para a empresa (15).

A equipe: é a equipe do projeto que possui autonomia para definir as ações necessárias para o atingimento dos objetivos preestabelecidos para o projeto. Como os métodos ágeis dependem substancialmente de colaboração e comunicação, a equipe é fundamental para o seu sucesso. Um único desenvolvedor forte, desenvolvedores que não trabalham bem juntos, um cliente que não se envolve com a equipe, cada um desses fatores pode destruir a natureza colaborativa de um grupo. A falta de sinergia da equipe representa um risco significativo para o projeto ágil (15).

A **rotatividade** é outro significativo fator de pessoal para ser considerado em uma equipe ágil. Sem documentação formal, a alta rotatividade em um projeto pode levar à perda de conhecimento crítico. Embora isso possa ser mitigado por revisões de código e ter desenvolvedores girando o trabalho em diferentes áreas funcionais, a perda de um importante membro da equipe pode ser catastrófico. O gerente de projeto deve considerar esse risco ao examinar se a equipe (e a organização) está adequada para o método ágil (15).

Uma das decisões a serem tomadas por um gerente de projeto de *software* é quanto à equipe adequada para o projeto. Definir as pessoas que atuarão no projeto de *software* não é tarefa simples, existem vários aspectos a serem observados e avaliados (24). A composição da equipe deve estar diretamente relacionada a natureza e complexidade das atividades a serem realizadas no projeto.

Fatores de desempenho: Sudhakar *et al.* listam quatro classes de fatores que influenciam desempenho da equipe: (i) técnico; (ii) não técnico; (iii) organizacional; e (iv) ambiental. Os fatores técnicos incluem características específicas dos projetos como tamanho, complexidade e processos, bem como características do produto. No nível individual, os fatores cognitivos incluem habilidade, conhecimento, competência e raciocínio lógico. A motivação é um fator relativo que tem recebido muita atenção em pesquisa de Engenharia de *Software*. Os valores pessoais, crenças e personalidade também foram investigados como fatores de desempenho diretos ou indiretos. Além disso, fatores afetivos foram examinados, mostrando que os desenvolvedores experimentam várias emoções em seu trabalho e que elas mudam com o tempo (2). O humor pode influenciar as tarefas de programação, como a depuração. Entusiasmo, valência e dominância emocional podem ter efeito positivo sobre o desempenho, enquanto a frustração é fator de risco negativo

para o desempenho. No nível de grupo ou equipe, alguns dos fatores relatados incluem coesão, confiança, clareza de objetivos e metas, estrutura de grupo e comunicação, troca de conhecimento, relações de equipe, diversidade, liderança e processos de coordenação. Os fatores organizacionais incluem cultura organizacional, clima, estrutura e valores. Finalmente, os fatores ambientais incluem características da indústria e volatilidade e também fatores relacionados aos clientes e concorrentes (2).

A seguir apresentam-se os fatores que impactam a produtividade, segundo o trabalho realizado em (25):

Características individuais e do grupo: descrevem os membros e tipos de equipe. A maioria dos quadros teóricos de desempenho de equipe inclui características de *design* da equipe, como tamanho e composição da equipe. Equipes bem projetadas para executar suas atividades são mais propensas a receber autoridade adicional sobre seu trabalho, mais recursos de apoio e metas mais desafiadoras. O *design* da equipe pode estar relacionado ao desempenho da equipe, pois afeta a quantidade de conhecimento e habilidades; os membros da equipe devem se inscrever na tarefa da equipe (25).

As características mais relevantes dos membros da equipe são conhecimento, habilidades e personalidade, enquanto as características da equipe incluem tamanho, diversidade, rotatividade de pessoal e crenças compartilhadas. Capacidades de equipe e habilidades são as características pessoais mais significativas que influenciam produtividade do desenvolvimento de *software* (25).

O desenvolvimento ágil é centrado nas pessoas e reconhece o valor das competências dos membros da equipe ao trazer agilidade aos processos de desenvolvimento de *software*. Obter as pessoas certas, com habilidades apropriadas e capacitá-las é fundamental para o sucesso do desenvolvimento ágil. A diversidade de equipes é primordial para o desenvolvimento ágil. Equipes com experiência mais ampla estão positivamente associadas ao desempenho do projeto. Essas alegações sugerem que as características do grupo são fatores que impactam na produtividade da equipe ágil (25).

Estágio do desenvolvimento da equipe: está relacionado à maturidade da equipe. Os membros da equipe devem aprender novos comportamentos e habilidades para trabalhar melhor e criar uma equipe de alto desempenho (25).

A natureza da tarefa: inclui o desenho da tarefa, a duração da tarefa, o grau de autonomia para executar as tarefas e interdependência de tarefas. *Cohen e Ledford* argumentam que tarefas enriquecidas permitem variedade, significância, autonomia e *feedback*, que resultam em alta responsabilidade, motivação, satisfação e desempenho da equipe. No desenvolvimento de *software*, a adequada atribuição de tarefas impacta a produtividade, porque pode influenciar a motivação do membro da equipe. As equipes ágeis devem ter autonomia suficiente para determinar quais tarefas devem ser executadas, demonstrando

resultados no final de cada iteração (25).

Contexto organizacional: inclui variáveis como recompensas, cultura, treinamento e recursos. Recompensas coletivas ajudam a motivar grupos cujas tarefas se tornaram interdependentes, enquanto que as recompensas individuais reconhecem membros cujas tarefas foram executadas refletindo as responsabilidades individuais (25).

Estilo de liderança: comportamentos de supervisão dependem do estilo de liderança, seja transacional ou transformacional, e orienta a equipe diretamente ou incentiva o autogerenciamento. Líderes transacionais geralmente estabelecem metas, obtêm o acordo da equipe sobre o que deve ser realizado e monitora o desempenho da equipe (25). Líderes transformacionais são inspiradores e estimulantes, proporcionando aos seguidores a sensação de objetivo, articulando objetivos compartilhados e compreensão mútua e um futuro atraente. Para isso, eles consideram o nível de maturidade, capacidades e necessidades dos subordinados, tratando os funcionários como indivíduos (25).

Além disso, o desenvolvimento de *software* é um trabalho mental que envolve a criação de conhecimento ou, pelo menos, o uso do conhecimento como parte dominante do trabalho. Esse conceito pode mudar a maneira como entendemos e interpretamos a produtividade do *software*, uma vez que o conhecimento é complexo e difícil de avaliar (26). Produtividade em equipes ágeis está, portanto, mais relacionada à capacidade de gerenciar entregas e cumprimento de prazos, mantendo a qualidade e satisfazendo o cliente (26). Embora a produtividade tenha sido estudada, ela permanece uma questão controversa. Primeiro, existem vários conceitos envolvidos em sua definição, como eficácia, eficiência e desempenho, gerando mal-entendidos e sobrecarga de termo. Em segundo lugar, a medição da produtividade do *software* é definida como a relação de saída (por exemplo, linhas de código, pontos de função ou recursos implementados) divididos por entrada (por exemplo, esforço de tempo).

O risco de baixa produtividade de desenvolvimento de *software* tem três fontes: ter uma equipe ineficiente, ter um equipe que não é coesa e ter uma equipe com moral pobre - tudo o que pode surgir devido a confrontos, personalidade ou escolhas de atribuição de papel pobres (27). Se a capacidade de determinados integrantes da equipe for maior que a complexidade das atividades do projeto, isso pode resultar em desperdício de talentos. Se as habilidades de determinadas pessoas da equipe não satisfazem a necessidade do projeto, isso afetará o progresso dos trabalhos (28) e, consequentemente, o resultado do produto a ser entregue.

Serão descritos a seguir os conceitos de competência, relevante de serem entendidos no contexto da discussão sobre composição de times de desenvolvimento de *software*.

2.2.1.1 Knowledge, Skills, Attitudes (KSA)

O uso da abordagem de competência como base para a gestão de recursos humanos tornou-se difundida nos Estados Unidos e está ganhando posição em práticas internacionais de recursos humanos (29). Competência descreve as habilidades, conhecimento, comportamentos, características pessoais e motivações associadas ao sucesso em um trabalho (30). A seguir apresentam-se as mais citadas definições de competências incluem (30):

- “Motivo, característica, habilidade, aspecto da autoimagem ou papel social ou corpo do conhecimento” (31).
- “Uma ferramenta de decisão validada, correlacionada a um grupo específico de atividades que descreve os conhecimentos e habilidades para realizar atividades” (32).
- “Uma mistura de conhecimento, habilidades, motivação, crenças, valores e interesses” (33).
- "Uma descrição por escrito de hábitos de trabalho mensuráveis e habilidades pessoais usadas para atingir os objetivos do trabalho” (34).
- “As competências são características subjacentes das pessoas e indicam maneiras de se comportar ou pensar, generalizando por meio das situações e perdurando por um período de tempo razoavelmente longo” (35).
- “Um conhecimento, habilidade ou característica associada a desempenho em um trabalho” (36).
- “Competências podem ser motivos, traços, autoconceitos, atitudes ou valores, conhecimento de conteúdo ou habilidades cognitivas ou comportamentais - qualquer característica individual que possa ser medida ou contada de forma confiável e que pode ser mostrada para diferenciar entre superior e desempenhos médios ou entre artistas eficazes e ineficazes” (37).

A força motriz por trás da introdução de uma abordagem baseada em competências é a necessidade de traduzir os imperativos dos negócios em novos requisitos de aprendizado e desempenho dos times visando ajudar as organizações a enfrentar os crescentes desafios do mercado (29). Modelos de competência, portanto, funcionam como veículos para dirigir mudança organizacional (38). Os profissionais de gestão de pessoas são confrontados com muitas novas oportunidades para utilizar modelos de competência visando melhorar indivíduos e equipes (30). Um conjunto gerenciável de características pessoais pode servir como uma estrutura conceitual para seleção, avaliação, desenvolvimento de talentos, gerenciamento de desempenho e outros programas de recursos humanos (30).

Considerando a importância da equipe nas metodologias ágeis, é importante que o conceito de competência seja muito bem entendido. Em síntese, o conceito de competência pode ser entendido como a capacidade de o desenvolvedor executar um trabalho corretamente, mas também é definido como uma combinação de conhecimentos, habilidades e atitudes (KSA) usados para melhorar a produtividade (39). É o conjunto de habilidades e conhecimentos de que um indivíduo necessita para executar determinado trabalho (40).

Conhecimento: todas as fontes de educação que compõem o conhecimento do desenvolvedor (educação formal e informal) e ele os utiliza na realização de suas tarefas diárias (39). É a informação e compreensão sobre um assunto que uma pessoa tem em mente (41).

Habilidades: as características adquiridas a partir da experiência são consideradas habilidades. Fornece informações sobre o uso do conhecimento (39). É a capacidade de dominar os conceitos de uma disciplina ou domínio e aplicar esse conhecimento adequadamente em novas situações (42).

Atitudes: a maneira como a pessoa pensa, sente e faz é considerada como suas atitudes. Está relacionada ao comportamento registrado nas tarefas realizadas diariamente na empresa de *software* (39).

Habilidade, conhecimento e competência são identificados como recursos essenciais da economia do conhecimento. Estes não podem ser comprados e/ ou entregues instantaneamente. Geralmente leva um tempo para desenvolver e apoiar a infraestrutura necessária para nutrir as habilidades e competências (43). Identificar o que a equipe sabe ou não sabe é conhecido como **gestão de competências**. Estudos têm mostrado que as pessoas muitas vezes não estão cientes dos conhecimentos que podem ser relevantes para elas (43).

O gerenciamento de conhecimento envolve uma variedade de assuntos, como Economia, Psicologia, Informática e Tecnologia. Assim, existem várias definições de conhecimento e gestão do conhecimento. Esse modelo de aprendizagem categoriza o conhecimento em tácito e formas explícitas (44). Uma boa definição de **gestão do conhecimento (KM)** é dada por Swan *et al.* que definiu como "qualquer processo ou prática de criar, adquirir, capturar, compartilhar e usar conhecimento, em que quer que ele esteja, para melhorar a aprendizagem e o desempenho das organizações" (43).

Os métodos ágeis sugerem que a maioria das documentações escritas seja substituídas por comunicações informais entre os membros da equipe e entre a equipe e os clientes, com ênfase mais forte no conhecimento tácito do que no conhecimento explícito. Os métodos ágeis sugerem diariamente reunião em pé durante a qual cada desenvolvedor (ou um par) precisa apresentar seu trabalho desde a última reunião. Os membros da equipe também podem expressar suas dúvidas durante as reuniões. Essa apresentação

fornece visibilidade do trabalho do apresentador para outros desenvolvedores e gerentes de projetos. Todos na equipe sabem quem trabalhou ou conhece bem quais partes do sistema. Eles sabem quem contatar quando precisam trabalhar partes do sistema com as quais não estão familiarizados. A criação e o compartilhamento de conhecimento são partes cruciais no processo de desenvolvimento de *software* ágil (44). Nos métodos ágeis argumenta-se que o custo de criar e atualizar documentos contra mudanças frequentes nos requisitos e código-fonte supera os benefícios de documentar o conhecimento do sistema e o domínio de detalhes. Dito isto, os métodos ágeis não deixam de fora a documentação, mas sim promovem a autodocumentação de projetos e código autodescritivo que está em conformidade com os padrões e diretrizes de indústria ou interna (44).

Nos times ágeis, a informação é comunicada informalmente e é simplesmente mantida como parte do conhecimento coletivo da organização. Enquanto reduzir a quantidade de documentação pode aumentar a produtividade, ela tem algum risco e custo. A documentação serve como uma maneira de trazer mais rapidamente novos membros até a velocidade do time. É útil ao fazer a transição do projeto para uma equipe de manutenção. De uma perspectiva de negócios, documentos formam a base para auditorias que asseguram que procedimentos de qualidade são seguidos. A documentação serve como um repositório de conhecimento de domínio. Se a organização muda drasticamente, esse conhecimento pode ser perdido (15). Esse processo de documentação leve evita esforço desperdiçado em que os documentos são escritos uma vez e depois se tornam obsoletos, pois não são atualizados para refletir as alterações (15).

Organizações perceberam que o gerenciamento adequado de suas habilidades e competências é base fundamental para a sua sobrevivência e rentabilidade na economia do conhecimento (43).

2.2.1.2 A alocação de recursos em projetos

Os recursos humanos são um dos fatores que determinam a sobrevivência e o desenvolvimento de uma empresa (28). A alocação adequada dos recursos humanos considerando o seu grau de conhecimento e o cargo na empresa tem impacto direto no uso racional dos recursos da empresa e, consequentemente, em sua eficiência (28). É um fator-chave que decide se a empresa pode ter um desenvolvimento sustentável, estável e rápido ou não (28). Para empresas, prever a demanda por recursos humanos pode ajudar a detectar os cargos com escassez de pessoal e fornecer orientações precisas para a alocação racional de recursos humanos (28). Portanto, hoje em dia, vários sistemas de informação de recursos humanos começam a se concentrar no desenvolvimento de funções de apoio à decisão e no estudo de como fazer uso dos dados existentes sobre recursos humanos para otimizar a relação de alocação entre o pessoal interno e os projetos da empresa e resolver problemas de alocação de recursos de forma a fornecer suporte a decisões racionais sobre

as posições ideais para os recursos (28).

Alocação dos recursos humanos na empresa significa que, por meio do processo de avaliação, seleção, recrutamento e treinamento, as empresas alocam diferentes tipos de pessoal que atendam às necessidades do cargo e do desenvolvimento dos negócios, aos postos de pessoal necessários, de forma oportuna e razoável (28). Por um lado, pode ajudar as pessoas a alcançarem suas metas de se desenvolverem e maximizarem seus talentos pessoais; por outro lado, permite que os recursos alcancem o seu melhor no cargo (28). A operação de cada cargo deve ser realizada por pessoal apropriado (28). Alocação racional dos recursos humanos da empresa, em combinação a outros recursos desta, pode ajudar a formar um movimento econômico virtuoso, o que pode aumentar a produtividade dos recursos humanos (28). Portanto, a alocação de recursos humanos é a questão-chave da gestão de recursos humanos (28). Por último, o objetivo da alocação dos recursos humanos nas empresas é melhorar a sua eficácia global, de modo a criar mais benefícios econômicos e sociais para os empreendimentos (28).

2.2.2 Métodos ágeis

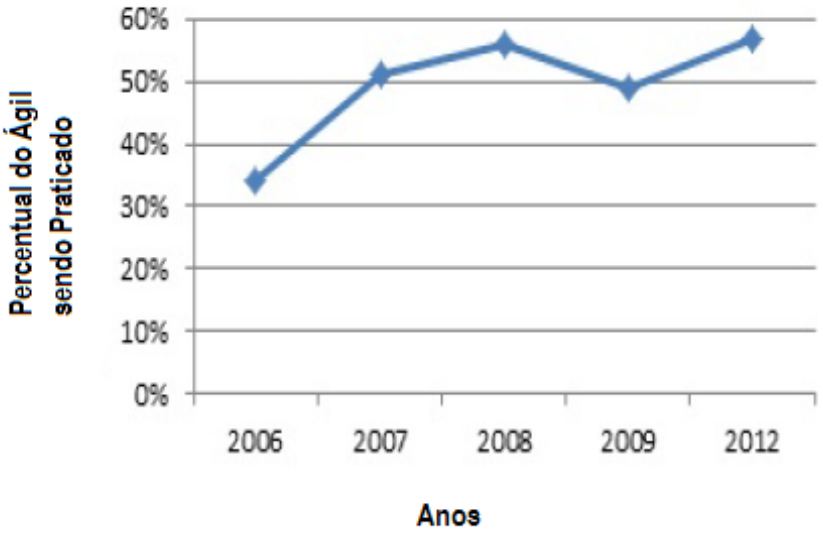
Metodologias ágeis de desenvolvimento de *software* fornecem uma maneira eficiente e mais leve de desenvolvimento, orientada para a construção do *software* de forma iterativa e incremental (45), substituindo o *design* de alto nível por redesenho frequente (46). As tarefas são realizadas em pequenas fases de trabalho, contribuindo para a modificação frequente do *software* de acordo com as mudanças de requisitos solicitadas pelos clientes (46).

Embora vários estudos tenham sido conduzidos por equipes individuais, poucos dados empíricos estão disponíveis em apoio ao sucesso e alta adoção de metodologias ágeis de desenvolvimento de *software*. O relatório CHAOS 2011 do *Standish Group* concluiu que as metodologias ágeis são três vezes mais bem-sucedidas do que as abordagens tradicionais de desenvolvimento de *software* (46). Pesquisa conduzida por Murphy *et al.* mostra aumento na prática de métodos ágeis na *Microsoft* de 2006 a 2012, conforme demonstrado na Figura 1 (47).

O Manifesto Ágil definiu 12 princípios ágeis que compõem o alicerce do desenvolvimento de *software* ágil. Esses princípios incluem (45):

- Satisfação do cliente pela entrega antecipada e contínua de *software* viável.
- A mudança de requisitos é bem-vinda, mesmo depois das etapas de desenvolvimento.
- Colaboração e comunicação frequentes entre os clientes e desenvolvedores.
- Entrega de *software* funcionando com frequência.

Figura 1 – Percentual do método ágil sendo praticado na *Microsoft* ao longo dos anos



- Apoio e motivação de pessoas envolvidas no desenvolvimento de *software*.
- Uso da comunicação face a face.
- O *software* funcional é a principal medida de progresso.
- O ritmo constante é mantido por meio de desenvolvimento sustentável.
- Atenção ao bom *design* continuamente.
- Manutenção das coisas simples.
- Equipes auto-organizadas podem desenvolver melhor arquitetura, requisitos e *design*.
- A equipe regularmente reflete como se tornar mais eficaz.

O guarda-chuva "ágil"fornece abrigo para metodologias de desenvolvimento de *software* incluindo *Scrum*, *Extreme Programming*, *Crystal*, *Kanban*, FDD (desenvolvimento orientado a recursos), DSDM (método dinâmico de desenvolvimento de sistemas), etc. (47). O relatório *Agile* Survey 2012 de Xebia revelou que 80% dos entrevistados concordaram em usar metodologias ágeis para desenvolvimento de *software*. Além disso, a pesquisa concluiu que 90% dos entrevistados seguem *Scrum* e *Extreme Programming* (XP), enquanto 30% seguem *Kanban* (46). A seguir será descrita a metodologia *Scrum* com suas principais características.

2.2.2.1 *Scrum*

A indústria de *software* tem enfrentado desafios de demandas dinâmicas de clientes, requisitos de *software* complexos, cronogramas apertados de projeto e restrições de recursos e orçamento. Isso exige uma abordagem pragmática do desenvolvimento de *software* que ofereça produtos de *software* de alta qualidade dentro do orçamento e cronograma alocados. O *Scrum* é uma dessas metodologias que gerencia o desenvolvimento de *software* em várias iterações curtas conhecidas como *Sprints*. Cada *sprint* inclui todas as fases de um modelo de ciclo de vida de desenvolvimento de *software*, como *design*, implementação, teste, revisão de clientes, etc. (46). A metodologia *Scrum* é orientada por três funções principais:

Dono do produto: responsável por definir, priorizar e comunicar os requisitos do produto e orientar o processo de desenvolvimento do produto (46). É responsável pela visão do produto, maximizando o retorno do investimento e também por decidir pela continuidade do desenvolvimento ou não (47).

Equipe de desenvolvimento: responsável pela execução das tarefas alocadas pelo dono do produto no prazo final da *Sprint*. Normalmente, uma equipe multifuncional

de três a nove pessoas implementa as tarefas de desenvolvimento do produto imaginadas pelo dono do produto (47).

Scrum Master: é um novo papel de gerenciamento introduzido pelo *Scrum*. Ele é responsável por garantir que o projeto seja realizado de acordo com as regras e princípios do desenvolvimento baseado em *Scrum* (23). O *Scrum Master* é também responsável por remover impedimentos ao desenvolvimento e ajuda a melhorar o processo, a equipe de desenvolvimento e o produto de *software* que está sendo desenvolvido (47).

A metodologia ágil é caracterizada por:

Colaboração: o desenvolvimento baseado em *Scrum* promove a colaboração, pois é conduzido por equipes multifuncionais, nas quais todas as pessoas com suas habilidades e experiências contribuem para a melhor solução de *design*. Uma equipe multifuncional inclui uma combinação de programadores, arquitetos de *software*, analistas de *software* e especialistas em controle de qualidade (46).

Reuniões diárias: são reuniões diárias organizadas para acompanhar o progresso da equipe *Scrum* (23). Elas têm curta duração e é nelas que a equipe de desenvolvimento de produto se comunica e avalia o *status* de progresso do desenvolvimento de *software*, aumentando assim a produtividade dos membros da equipe (46). O *Scrun Master* conduz as reuniões (23) realizadas pelo time ágil.

***Product Backlog*:** define tudo que é necessário para que o produto de *software* seja entregue com sucesso (23). Ele define o trabalho a ser feito no projeto (46). O *Product Owner* é responsável pelo conteúdo, priorização e disponibilidade do *Backlog* do produto (48). O *Backlog* do produto nunca é concluído e evolui à medida que o produto e o ambiente em que ele será usado evoluem, ou seja, ele é dinâmico e gerido constantemente para identificar o que precisa ser apropriado ao produto para que ele se mantenha competitivo e útil (48). Desde que exista um produto, o *Backlog* do produto também existe (48).

***Backlog* da *Sprint*:** define o trabalho, ou a lista de tarefas a serem executadas pela equipe de desenvolvimento durante a próxima *Sprint*, considerando a capacidade de trabalho e os desempenhos anteriores da equipe de desenvolvimento de *software* (46). O *Backlog da Sprint* define o trabalho que uma equipe estabelece para transformar os itens do *Product Backlog* selecionados para esse *sprint*, em um incremento de funcionalidades do produto potencialmente utilizável (48).

No entanto, existem alguns desafios que as organizações podem encontrar ao adotar a metodologia *Scrum* (47):

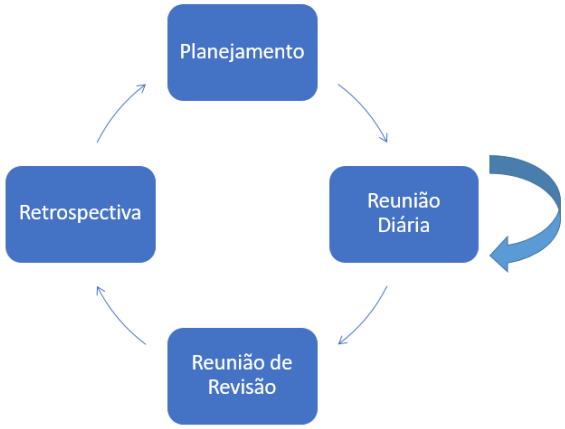
- O custo para a empresa pode aumentar devido à rotatividade de pessoal.
- Disponibilidade de recursos qualificados.

- Capacidade de criar aplicativos a partir de vários locais em projetos com equipes distribuídas geograficamente.
- O desenvolvimento orientado por testes é difícil, pois às vezes leva ao excesso de geração de código.

Sprint Planning

O *Scrum* é geralmente composto de quatro grandes etapas, planejamento de *Sprint*, *Sprints*, reuniões de revisão de *Sprint* e reuniões de retrospectivas de *Sprint*. Contudo, mais ênfase foi colocada nos componentes de gerenciamento de projetos do *Scrum*. Uma visão do *Scrum* divide seu modelo de gerenciamento de projetos em duas fases amplas, planejamento inicial e o ciclo de *Sprint*. A subfase do planejamento inicial consiste em uma sessão de descoberta quando os projetos são iniciados, define o escopo e o organiza. Nisso também consiste a subfase de planejamento de *release* quando um *backlog* de projeto é formado contendo as prioridades das necessidades do usuário e um cronograma geral para múltiplas *Sprints* de desenvolvimento. A fase do ciclo de *Sprint* consiste em uma subfase de planejamento de *Sprint*, a *Sprint* de desenvolvimento em si, reuniões diárias de equipe, análises de *Sprint* e retrospectivas (49). O ciclo de reuniões do *Scrum* está demonstrado na Figura 2.

Figura 2 – Fluxo de reunião do *Scrum*



Fonte: (47) adaptado.

2.2.2.2 Gerenciamento de projetos ágeis

O gerenciamento de projetos ágeis trata-se de gerenciar o impacto da complexidade e da incerteza em um projeto, reconhecendo (50):

- A necessidade de um período de tempo dramaticamente mais curto entre planejamento e execução.

- Que planejar uma ação não fornece todos os detalhes da sua implementação.
- Que a criatividade e a aprendizagem são necessárias para fazer sentido nesse ambiente.

Ao contrário da sequência linear de atividades bem definidas no gerenciamento tradicional de projetos, o gerenciamento ágil é caracterizado pela entrega de funcionalidades do produto em ciclos curtos de interação iterativa e incremental e integração contínua de alteração de código. O gerenciamento de projetos ágeis rompe com o sequência linear de atividades bem definidas do projeto de gestão tradicional e muda o foco do planejamento inicial para execução. Ao fazer isso, o gerenciamento de projetos ágeis muda a estrutura tradicional de comando e controle para decisões compartilhadas, autogestão e aprendizagem em equipes de *software*, para lidar com a complexidade e imprevisibilidade dos problemas de solução atividades de projetos de *software* (50). O gerenciamento de projetos ágeis introduz mudanças nos papéis de gestão, bem como nas práticas; muda a natureza de colaboração, coordenação e comunicação em projetos de *software* (50).

O planejamento de projetos de *software* constitui uma tarefa vital para muitas atividades de Engenharia de *Software*. O planejamento ruim pode causar custos e atrasos inaceitáveis, impondo restrições financeiras e de tempo ao projeto (51). Em qualquer planejamento de um projeto de *software*, em função da escala e da complexidade das atividades relacionadas, faz-se necessária a sua execução por uma equipe em vez de por um único indivíduo. A composição inadequada da equipe tem potencial de resultar em falhas do projeto, do portfólio de projetos e até mesmo comprometer a continuidade da organização do *software* em si (27).

É durante a fase de planejamento de *software* que se deve realizar a tarefa de compor a equipe que irá atuar no projeto. Para isto, deverão ser consideradas as características já mencionadas na seção 2.2.1.1: conhecimento, habilidade e atitude (KSA) (39) dos recursos que serão alocados, bem como a natureza das atividades que serão executadas no projeto. Planejar o desenvolvimento de um novo produto de *software* ou evoluir e manutenir um produto já existente não é tarefa fácil. As equipes ágeis usam três níveis de planejamento: planejamento de *release*, planejamento de iteração e planejamento diário (20).

O **planejamento de *releases*** no desenvolvimento de *software* incremental atribui funcionalidades de tal modo que as restrições técnicas, recursos, riscos e orçamento sejam atendidos (16). Uma funcionalidade é definida como um conjunto de requisitos logicamente relacionados que agregam valor ao produto e atendem aos objetivos de negócio dos seus usuários (16). Cada funcionalidade é parte da *release* e somente será disponibilizada ao mercado se todas as tarefas necessárias para sua conclusão forem executadas antes da data de lançamento da *release*. Diferentes tarefas requerem diferentes habilidades (16). Nesse

contexto, a escolha dos profissionais, dentro de um *pool* de profissionais disponíveis, que irão compor a equipe que atuará em determinada *release* está diretamente relacionada às tarefas que compõem a *release* e que, por sua vez, podem exigir diferentes níveis de habilidades (16).

O **plano de *release*** aguarda a duração da *release* tipicamente três a seis meses. Um plano de iteração olha em frente apenas a duração de uma iteração, geralmente de uma a quatro semanas. Um plano diário é o resultado de compromissos dos membros da equipe feitos uns com os outros geralmente durante um encontro rápido (em pé) diário (20).

A cada nova *release* novos objetivos são estabelecidos e se faz necessário avaliar a composição mais adequada do time, especialmente em se tratando de times autogerenciáveis que estão em constante evolução e se tornam cada vez mais preparados para novos desafios. Outro ponto é que, muitas vezes, talentos devem ser constantemente identificados para fazerem diferença nos projetos e não se desmotivarem (16).

As *releases* e recursos dependem um do outro (16). Definir o escopo da *release* sem verificar os recursos necessários pode implicar a inviabilidade do planejamento proposto. Por outro lado, o planejamento de recursos sem considerar o impacto comercial da *release* para seus usuários e para o mercado pode resultar na perda da oportunidade de gerar valor à solução de *software* (16).

O impacto dos métodos ágeis no gerenciamento de projetos

Os métodos ágeis têm impactos sobre as pessoas, processos e elementos de esforço do projeto. A seguir estão detalhados alguns desses impactos (15):

Pessoas: como dito na seção 2.2.1, as pessoas são peça fundamental para o sucesso do projeto.

Processo: como os métodos ágeis representam um novo princípio, processa atividades e subobjetivos, eles têm um impacto em muitos dos processos de uma organização. Os velhos processos (por exemplo, planejamento, desenvolvimento, entrega, operações) devem ser substituídos por processos ágeis. As mudanças culturais na organização, em função dos métodos ágeis, levam ao fim as velhas maneiras de pensar, gerando resistência (15).

Planejamento: processos ágeis são caracterizados por colocar menos ênfase no planejamento formal. Isso não quer dizer que o planejamento não ocorre. Com tantas pequenas tarefas, argumenta-se que os processos ágeis exigem mais planejamento, mas ao contrário de outras metodologias, o planejamento não é direto seguido de microajustes. Pelo contrário, é um tarefa constante para garantir resultados de entregas (15). Um bom processo de planejamento resulta em (39):

1. Redução dos riscos;

2. redução das incertezas;

3. apoio à tomada de decisão;

4. estabelecimento de confiança;

5. melhor comunicação.

O planejamento ágil é um processo relativamente informal. Por exemplo, decidir o que vai entrar em cada caixa de tempo é realizado por meio das reuniões diárias, discutindo problemas pendentes, priorizando o trabalho e atribuição de recursos para os problemas. Em outro métodos ágeis, mesmo esse nível de planejamento pode não ser considerado. É importante levar em conta a informalidade (15).

2.2.2.3 Agilidade e qualidade

A agilidade abre novos horizontes na área de garantia da qualidade de *software* (18). As metodologias ágeis de desenvolvimento de *software*, desde a sua criação, têm sido desenvolvidas para melhorar a qualidade do produto final (52). Os praticantes de metodologias ágeis também alegam que o uso da abordagem ágil melhorou muito a qualidade dos produtos deles. No entanto, o conceito de qualidade do *software* é complexo; na verdade, alguns definiram toda a disciplina de Engenharia de *Software* como a produção de *software* de qualidade (52). Qualidade é um conceito bastante abstrato que é difícil de definir, mas onde existe pode ser reconhecido (52). No gerenciamento formal de qualidade de *software*, as atividades de garantia da qualidade são cumpridas, assegurando que cada um dos parâmetros listados a seguir foram considerados no desenvolvimento do *software* (18). Uma definição de cada um desses parâmetros é dada segundo Meyer (2000) (18):

Exatidão: é a capacidade de um sistema executar de acordo com a especificação definida.

Robustez: desempenho adequado de um sistema sob condições extremas. Isto é complementar à correção.

Extensibilidade: um sistema que é fácil de adaptar a uma nova especificação.

Reutilização: o *software* é composto de elementos que podem ser usados para construir diferentes aplicações.

Compatibilidade: o *software* que é composto por elementos que podem ser combinados com outros elementos.

Eficiência: é capacidade de um sistema colocar o menor número possível de demandas de recursos de *hardware*, como memória, largura de banda usada em comunicação e tempo do processador.

Portabilidade: é a facilidade de instalar o produto de *software* em diferentes *hardwares* e plataformas de *software*.

Pontualidade: liberação do *software* antes ou exatamente quando é necessário pelos usuários.

Integridade: quão bem o *software* protege seus programas e dados contra acesso a recursos não autorizados.

Verificabilidade: como é fácil testar o sistema.

Facilidade de uso: a facilidade com que as pessoas podem aprender e usar o programa.

2.2.3 *Search-Based Software Engineering* (SBSE)

A Engenharia de *Software* (ES) considera problemas que envolvem encontrar um equilíbrio entre objetivos que competem e que são conflitantes. Normalmente, há um conjunto de escolhas e encontrar boas soluções pode ser difícil. A seguir, estão listadas algumas questões de ES (10):

1. Qual é o menor conjunto de casos de teste que abrange todos os fontes desse programa?
2. Qual é a melhor maneira de estruturar a arquitetura desse sistema para melhorar sua manutenção?
3. Qual é o conjunto de requisitos que equilibra o custo de desenvolvimento de *software* e a satisfação do cliente?
4. Qual é a melhor alocação de recursos para esse projeto de desenvolvimento de *software*?
5. Qual é a melhor sequência de etapas de refatoração para aplicar a esse sistema?

Nessa linha, a Engenharia de *Software* preocupa-se com a gestão de atividades complexas que estão sendo realizadas em diferentes estágios do ciclo de vida do *software*, buscando otimizar os processos de produção de *software*, bem como os produtos feitos por esse processo (10). Todas essas questões são de otimização. Eles são típicos problemas para os quais a SBSE está bem adaptada (53). Embora o conceito de Engenharia de Software Baseada em Pesquisa (SBSE) exista há décadas, o termo SBSE foi cunhado e o campo foi formalizado em 2001 (12) por Harman e Jones. A mais antiga tentativa, atualmente conhecida, de aplicar a otimização para um problema de ES foi relatada por Miller e Spooner em 1976 na área de testes de *software* (10). A Engenharia de Software Baseada em Pesquisa (SBSE) provou ser uma maneira muito eficaz de otimizar os problemas de

ES com aplicações em todo o espectro de atividades, desde requisitos, planejamento do projeto, estimativa de custos de testes de regressão e evolução progressiva (10). A SBSE tem sido aplicada em problemas durante todo o ciclo de vida de desenvolvimento de *software*. A abordagem da SBSE procura otimizar os processos de ES e produtos usando pesquisa computacional genérica, robusta, flexível, expansível e com ricos *insigths* (11). A SBSE fornece um mecanismo para o gerenciamento automatizado de atividades de Engenharia de *Software* (11). A abordagem é atraente porque oferece um conjunto de soluções automáticas e semiautomáticas adaptáveis (10). O SBSE é um prática que vem sendo utilizada pela comunidade de pesquisadores em problemas de ES e com sucesso (12).

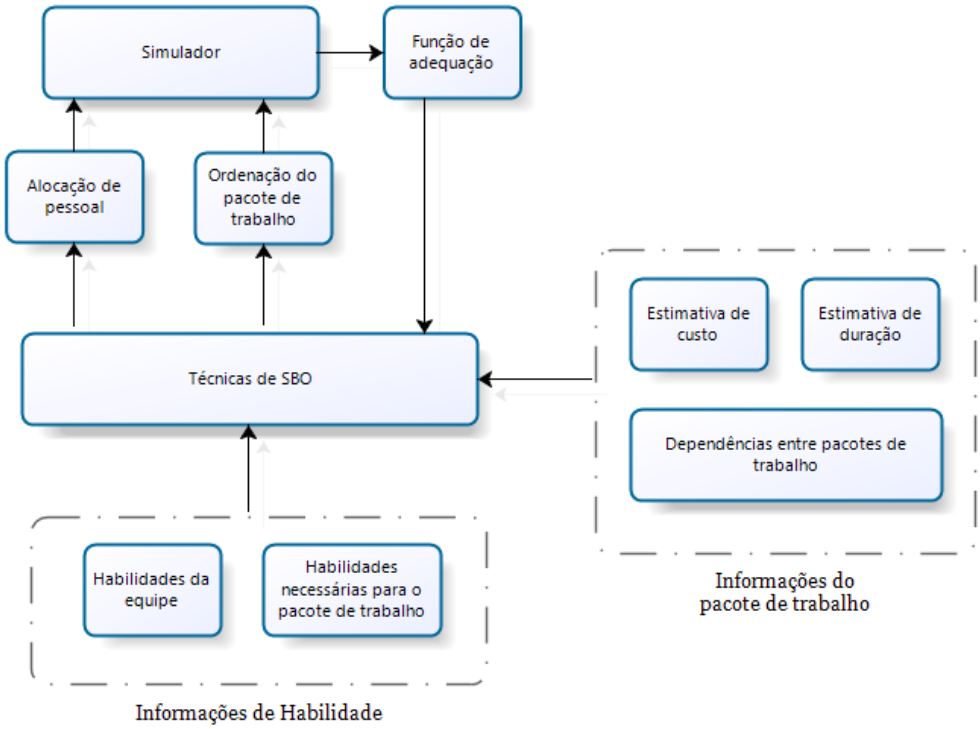
No SBSE, o termo "pesquisa" é usado para se referir ao uso de técnicas de otimização meta-heurística baseada em busca - *Search-based optimization* (SBO). A SBSE procura reformular os problemas da Engenharia de *Software* como problemas de SBO. Para a SBSE, um problema de pesquisa é aquele em que soluções ótimas ou quase ótimas são procuradas em um espaço de busca de soluções candidatas, guiados por uma função que distingue entre melhores e piores soluções (10).

Inicialmente, a SBSE estava focada principalmente na solução de problemas de otimização (SOPs) de um único objetivo, usando algoritmos de pesquisa, por exemplo, algoritmos genéticos (GA) (53). No entanto, muitos problemas de ES são tipicamente problemas multiobjetivos. Os objetivos envolvidos são muitas vezes conflitantes e contraditórios (10). A direção para a otimização por objetivos pode ser tanto maximizar ou minimizar seus valores. Por exemplo, na ES, pode-se querer maximizar as funcionalidades de entrega e ao mesmo tempo minimizar o custo de realizar a entrega. Se o modelo de entrega oferece apenas um objetivo, então se pode chamá-lo de um problema de otimização de único objetivo *(single-objective)*. Por outro lado, quando há muitos objetivos chamamos de problema de otimização de multiobjetivo *(multi-objective)* (54). Em tal espaço de objetivos concorrentes, não podemos ser ótimos em todos os objetivos, simultaneamente (54).

A primeira aplicação da SBSE no gerenciamento de projetos de *software* foi proposta para agendamento de projetos e alocação de recursos. A Figura 3 fornece uma visão geral do esquema genérico da abordagem SBSE para o planejamento do projeto. Uma vez fornecidas as informações de entrada sobre os pacotes de trabalho (por exemplo, custo, duração, dependências) e habilidades da equipe, a abordagem baseada em pesquisa busca uma ordem ideal de pacote de trabalho e alocação de pessoal orientada por uma função de adequação (*fitness function*) de um ou vários objetivos. Um objetivo natural da abordagem baseada em pesquisa para o gerenciamento de projetos é encontrar planos de projeto que minimizem o tempo de conclusão do projeto. Outro objetivo que tem sido levado em consideração é minimizar os riscos associados ao processo de desenvolvimento (por exemplo, atrasos no tempo de conclusão do projeto ou orçamentos reduzidos disponíveis) (55). A SBSE é aplicada para apoiar gerentes de projetos em áreas como o

gerenciamento de tempo, gerenciamento de custos, gerenciamento de qualidade, gestão de recursos humanos, gerenciamento de riscos (55), entre outros.

Figura 3 – Um esquema genérico de gerenciamento de projetos baseado em pesquisa



Fonte: (55) adaptado.

Fitness Function

O SBSE inicia com dois integrantes básicos (56):

1. A escolha da representação do problema.
2. A escolha da função de adequação (*Fitness Function*)

A simplicidade faz com que a SBSE seja atrativa. Com justamente dois ingredientes, o engenheiro de *Software* poderá implementar o algoritmo de otimização baseado em pesquisa e obter resultados. Normalmente, a Engenharia de *Software* terá a representação adequada para o problema. Muitos problemas em Engenharia de *Software* também têm, associadas a eles, métricas de *software* que naturalmente formam bons candidatos iniciais para as funções *fitness*. Pode ser que o engenheiro de *software* já terá em mãos a implementação de algumas métricas de seu interesse. Com um pouco de esforço isso pode ser transformado em uma função *fitness* e assim a "curva de aprendizado"e o investimento em infraestrutura necessária para iniciar a SBSE estão entre os menores que se pode encontrar (56).

Com esses dois ingredientes, torna-se possível implementar algoritmos de otimização baseados em pesquisa. Esses algoritmos usam diferentes abordagens para localizar soluções ótimas ou quase ótimas. No entanto, todos eles são essencialmente uma pesquisa por meio de muitas possíveis instâncias candidatas da representação, orientadas pela função *fitness*, que permite ao algoritmo comparar soluções candidatas de acordo com a sua eficácia na resolução do problema (medido pela função de *fitness*) (57).

Algoritmos de otimização

Para diferentes tipos de problemas de otimização, geralmente se faz necessário usar diferentes técnicas de otimização (58). Em geral, os algoritmos de otimização podem ser divididos em duas categorias: algoritmos determinísticos e algoritmos estocásticos (58). Algoritmos determinísticos seguem um procedimento rigoroso e seu caminho e valores de ambas as variáveis de projeto e as funções são repetíveis (58). Por exemplo, *hill-climbing* é um algoritmo determinístico e para o mesmo ponto de partida, eles seguirão o mesmo caminho se executar o programa em diferentes momentos. Por outro lado, os algoritmos estocásticos sempre têm alguma aleatoriedade (58). Algoritmos genéticos são um bom exemplo de algoritmos estocásticos, as sequências de caracteres ou soluções na população será diferente cada vez que você executa um programa, pois os algoritmos usam alguns números pseudoaleatórios, embora os resultados finais possam não ser uma grande diferença, mas os caminhos de cada indivíduo não são exatamente repetíveis. Além disso, existe um terceiro tipo de algoritmo, que é uma mistura ou híbrido de determinístico e algoritmos estocásticos (58). Grande parte dos algoritmos convencionais ou clássicos é determinística (58).

Para algoritmos estocásticos, temos em geral dois tipos: heurístico e meta-heurístico, embora a diferença seja pequena. Heurístico significa "encontrar"ou "descobrir por tentativa e erro" (58). Soluções de qualidade para um problema de otimização difícil podem ser encontradas em um período de tempo razoável, mas não há garantia de que soluções ótimas sejam alcançadas (58). É esperado que esses algoritmos encontrem soluções interessantes na maioria das vezes, mas não sempre. Isso é geralmente suficiente quando não queremos necessariamente as melhores soluções, mas boas soluções que são facilmente alcançáveis (58). Uma evolução sobre os algoritmos heurísticos são os chamados algoritmos meta-heurísticos (58). Meta significa "além"ou "nível superior", e eles geralmente têm melhor desempenho do que simples heurística (58). Além disso, todos os algoritmos meta-heurísticos usam certa troca de aleatoriedade e pesquisa local. Vale a pena ressaltar que não existem definições acordadas de heurísticas e meta-heurísticas na literatura, alguns usam "heurística"e "meta-heurística"de maneira intercambiável (58). No entanto, tendências recentes tendem a nomear todos os algoritmos estocásticos com randomização e pesquisa local como meta-heurística (58). Grande parte dos algoritmos meta-heurísticos é inspirada na natureza, pois foi desenvolvida com base em alguma abstração dela (58).

Dois componentes principais de qualquer algoritmo meta-heurístico são: seleção das melhores soluções e randomização. A seleção do melhor garante que as soluções convergirão para a otimização, enquanto a aleatorização evita que as soluções permaneçam em ótimos locais e, ao mesmo tempo, aumenta a diversidade das soluções (58). A boa combinação desses dois componentes contribuem para que a otimização global seja alcançável (58).

Otimização multiobjetivo (MOP)

Parte dos problemas de otimização no mundo real envolve a minimização e/ou maximização de mais de uma função objetivo (59). De modo geral, a otimização multiobjetivo não se limita a encontrar uma única solução de determinado problema de otimização multiobjetivo (MOP), mas um conjunto de soluções chamadas **não dominadas** (59). Cada solução nesse conjunto é considerada um Pareto ótimo, e quando elas são plotadas no espaço objetivo, elas são coletivamente conhecidos como frente de Pareto. Obter a frente de Pareto de um dado MOP é o principal objetivo da otimização multiobjetivo (59). No entanto, identificar todo o conjunto ótimo de Pareto, para muitos problemas multiobjetivos, é praticamente impossível devido ao seu tamanho (60). Em geral, os espaços de busca em MOPs costumam ser muito grandes, e a avaliação das funções-objetivo pode exigir significativa quantidade de tempo (59). Os algoritmos mais conhecidos para resolver MOPs pertencem à classe de algoritmos evolutivos (EAs) (por exemplo, NSGA-II, PAES e SPEA2) (59).

Os EAs são estratégias de busca inspirados na natureza, baseados na seleção natural da evolução, e são especialmente adequados para combater os MOPs, por causa de sua capacidade de encontrar várias opções de soluções em uma única execução (59). As subclasses bem-aceitas de EAs são algoritmos genéticos (GA), programação genética (GP), programação evolutiva (EP) e estratégias de evolução (EV). Esses algoritmos funcionam em um conjunto (**população**) de soluções potenciais (**indivíduos**) que são submetidas a operadores estocásticos para buscar melhores soluções. Esses operadores são a recombinação de operadores, que permitem a cooperação entre indivíduos, e o operador de mutação, consistindo de alterações aleatórias das variáveis do problema (59).

Algoritmo genético (GA) é talvez a mais conhecida de todas as técnicas de busca baseada na evolução. GA é um algoritmo de busca baseado na seleção natural que transforma um conjunto de indivíduos dentro da população de soluções em um novo conjunto para a próxima geração, usando operadores genéticos tais como cruzamento e mutação. A estratégia de sobrevivência mais forte é adotada para identificar a melhor solução e, subsequentemente, operadores genéticos são usados para criar novas soluções para a próxima geração. Esse processo é repetido geração após geração até que uma solução satisfatória seja encontrada. Algoritmos genéticos foram usados com sucesso para obter soluções para muitos problemas de otimização combinatória. Algoritmo genético para problema de otimização combinatória funciona da seguinte forma: o espaço de pesquisa contém todos os

nós de pesquisa para o problema determinado. GA começa com uma população inicial de nós de pesquisa do espaço de pesquisa. Cada nó de pesquisa tem um valor de aptidão atribuído a ele usando a função objetiva. Novos nós de pesquisa são gerados para a próxima geração com base no valor da aptidão e aplicação de operadores genéticos para os nós de pesquisa atuais. Esses processos são repetidos, geração após geração, até o algoritmo convergir (61).

Algoritmos evolutivos multiobjetivo (MOEA)

Os MOEAs criam a população inicial primeiro e depois executam o cruzamento e a mutação repetidamente até ficarem "cansados ou felizes"; ou seja, até que se tenha esgotado a limitação de tempo da CPU ou até que se tenha soluções alcançadas que são suficientes para os propósitos em questão. A estrutura básica para MOEAs é a seguinte (54):

1. Gera a geração 0 usando alguma política de inicialização.
2. Avalia todos os indivíduos na geração 0.
3. Repete até cansar ou:
    a) Cruza itens da geração atual para fazer nova população;
    b) muda a população fazendo pequenas mudanças;
    c) avalia indivíduos na população;
    d) seleciona algum subgrupo de elite da população para formar uma nova geração

Uma maneira simples de entender os MOEAs é compará-los com a teoria da evolução de Darwin. Para encontrar boas pontuações para os objetivos, partem de um grupo de indivíduos. Com o passar do tempo, indivíduos cruzam dentro do grupo. Os descendentes que têm melhores pontuações de aptidão tendem a sobreviver (na etapa de seleção). Durante a evolução, a operação de mutação pode aumentar a diversidade do grupo e evitar que a evolução fique presa no local ótimo (54).

Existem muitos operadores de cruzamento e mutação, como cruzamento de um/dois ponto(s), mutação gaussiana, mutação FlipBit, cruzamento uniforme parcialmente combinado (UPMX), etc. Determinados domínios podem exigir operadores de cruzamento especializados (54).

Os Parâmetros em algoritmos evolutivos

Durante os anos 80, muitos pesquisadores basearam suas escolhas no ajuste dos parâmetros dos algoritmos evolutivos, experimentando-os com diferentes valores e selecionando os que davam melhores resultados (62). A visão contemporânea sobre algoritmos

evolutivos, no entanto, reconhece que problemas específicos exigem configurações específicas dos EAs para um desempenho satisfatório (62). Isso enfatiza necessidade de técnicas eficientes que ajudem a encontrar boas configurações de parâmetros para um dado problema, ou seja, a necessidade de bons métodos de ajuste de parâmetros (62).

O ajuste de parâmetros à mão é uma prática comum na computação evolucionária. Normalmente, um parâmetro é ajustado de cada vez, o que pode causar baixa qualidade nas escolhas, já que os parâmetros geralmente interagem de maneira complexa. O ajuste simultâneo de mais parâmetros, no entanto, leva a representativa quantidade de experimentos. As desvantagens das técnicas de ajuste de parâmetro baseado na experimentação podem ser resumidas do seguinte modo (62).

- Os parâmetros não são independentes, mas tentar sistematicamente todas as diferentes combinações é computacionalmente inviável.
- O processo de ajuste de parâmetros é demorado.
- Para um dado problema, os valores dos parâmetros selecionados não são necessariamente ótimos.

Outras opções para projetar um bom conjunto de parâmetros estáticos por um método evolutivo para resolver um problema em particular incluem "parametrização por analogia" e o uso de análise teórica (62). A parametrização por analogia equivale ao uso de configurações de parâmetro que foram comprovadas com sucesso para problemas "semelhantes" (62).

A eficácia de um algoritmo evolutivo depende muito dos seus componentes e as interações entre eles (62). A variedade de parâmetros incluídos nesses componentes, as escolhas possíveis e a complexidade das interações entre vários componentes e parâmetros fazem a seleção de um algoritmo evolucionário "perfeito" para um dado problema muito difícil, se não impossível (62). Para maximizar sua eficiência, três parâmetros que devem ser otimizados são a probabilidade de mutação, a probabilidade de cruzamento (*crossover*) e o tamanho de população (63).

Tamanho da população (N): o tamanho da população afeta ambos, o desempenho final e a eficiência dos GAs. GAs em geral performam mal com populações muito pequenas, porque a população fornece um tamanho de amostra insuficiente para a maioria dos hiperplanos. Uma grande população é mais propensa a conter representantes de elevado número de hiperplanos. Como resultado, uma grande população desencoraja a convergência prematura para soluções subótimas. Por outro lado, uma grande população requer mais avaliações por geração, possivelmente resultando em uma taxa inaceitavelmente lenta de convergência (64).

Taxa de cruzamento (C): a taxa de cruzamento controla a frequência com a qual o operador de cruzamento é aplicado. Em cada nova população, estruturas C * N passam por cruzamento. Quanto mais alta a taxa de cruzamento, mais rapidamente novas estruturas são introduzidas na população. Se a taxa de cruzamento é muito alta, estruturas de alto desempenho são descartadas mais rápido que a seleção pode produzir melhorias. Se a taxa de cruzamento é muito baixa, a pesquisa pode estagnar devido a menor taxa de exploração (64).

Taxa de mutação (M): mutação é um operador de busca secundário que aumenta a variabilidade da população. Após a seleção, cada posição de *bit* de cada estrutura na nova população sofre uma mudança aleatória, com probabilidade igual à taxa de mutação M. Baixo nível de mutação serve para prevenir que qualquer *bit* em dada posição permaneça para sempre e convirja para um único valor em toda a população. Alto nível de mutação produz uma busca essencialmente aleatória (64).

Existem muitas abordagens para resolver problemas de otimização multiobjetivo. O algoritmo genético de ordenação não dominada -II (NSGA-II), MOCell e o algoritmo evolutivo de Pareto de força 2 (SPEA-2) tornaram-se abordagens-padrão (59). A Figura 4 descreve as categorias para os algoritmos de busca.

Figura 4 – Classificação dos algoritmos de busca

Categoria do Algoritmo			Algoritmo
Algoritmos Evolutivos (EAs)	Algoritmo Genético (GAs)	Base Classificada	NSGA II
		Base Celular	MOCell
	Força de Pareto EA		SPEA2
	Estratégias de Evolução		PAES
Algoritmos de Enxames	Teoria de Enxame de Partículas		SIMPSO
Algoritmo Híbrido	Algoritmo Genético Celular + Evolução Diferencial		CellDE

Fonte: (65) adaptado.

2.2.3.1 *Non-dominated Sorting Genetic Algorithm II* (NSGA-II)

O algoritmo NSGA-II foi proposto por Deb *et al.* em 2000 (66) e é um dos mais populares algoritmos de otimização de múltiplos objetivos (59). O procedimento NSGA-II é um dos procedimentos de otimização multiobjetivo evolutiva (EMO), que tenta encontrar múltiplas soluções ótimas de Pareto em um problema de otimização multiobjetivo e tem as três seguintes características (67):

1. Ele usa um princípio elitista.
2. Usa um mecanismo explícito de preservação da diversidade.
3. Enfatiza soluções não dominadas.

O NSGA-II classifica a população em várias frentes não dominadas usando um algoritmo de classificação seguido pela seleção de indivíduos dessas frentes não dominadas e gera nova população por aplicação de operadores de seleção, cruzamento e mutação. Além disso, o NSGA-II define uma métrica chamada distância de aglomeração para medir a distância entre uma solução individual e as outras. Se duas soluções individuais estão na mesma frente não dominada, a solução com um valor mais alto da distância de aglomeração é selecionada. O objetivo do indicador de distância de aglomeração é maximizar a diversidade das soluções não dominadas produzidas (65).

Geralmente, o NSGA-II pode ser detalhado conforme as etapas a seguir (68):

Etapa 1: inicialização da população - inicialize a população com base no intervalo e na restrição do problema.

Etapa 2: classificação não dominada - processo de classificação baseado em critérios de não dominação da população que foi inicializada.

Etapa 3: distância de aglomeração - quando a classificação estiver concluída, o valor da distância de aglomeração será atribuído à frente. Os indivíduos da população são selecionados com base na classificação e distância de aglomeração.

Etapa 4: seleção - a seleção de indivíduos é realizada usando uma seleção de torneio binário com operador de comparação cheia.

Etapa 5: operadores genéticos - GA codificado real usando cruzamento binário simulado e mutação polinomial.

Etapa 6: recombinação e seleção - população de descendentes e população de geração atual são combinadas e os indivíduos da próxima geração são definidos por seleção. A nova geração é preenchida por cada frente subsequentemente até que o tamanho da população exceda o tamanho atual da população.

O meta-heurística NSGA-II começa gerando uma população P0 de tamanho N e é classificada de acordo com o operador de dominância. Uma segunda população, chamada Q0, com o mesmo tamanho de P0, é gerada aplicando os operadores de cruzamento e mutação ao longo de P0 (69).

As duas populações foram combinadas em uma terceira população chamada R0, com tamanho 2N. A população R0 é classificada usando o operador de dominância. Então, uma população chamada P1 é preenchida com as frentes de Pareto. As frentes de Pareto são geradas usando o operador de distância de aglomeração sobre a população R0. A população P1 é utilizada como população inicial para a próxima geração da execução do algoritmo. Esse processo continua até que o critério de parada seja alcançado (69).

O operador de dominância, que é uma das características do NSGA-II, trabalha avaliando e numerando cada solução de agenciamento de acordo com o número de outras

soluções que domina cada uma delas. Em seguida, inicia-se um processo no qual as soluções não dominadas são removidas do Rt. Em cada ciclo dessa fase, uma nova frente de Pareto é criada usando as soluções removidas da Rt. O processo continua até que todas as soluções de uma geração sejam avaliadas. O operador de distância de aglomeração calcula o número de soluções próximas de cada solução da frente. A avaliação é melhor quando o número de soluções vizinhas é pequeno (69).

2.2.3.2 *Improved Strength Pareto Evolutionary Algorithm* (SPEA2)

Como o SPEA (*Strength Pareto Evolutionary Algorithm*) (70) forma a base para SPEA2, é dado a seguir um resumo do algoritmo (71). O SPEA usa uma população regular e um arquivo (conjunto externo). Começando com uma população inicial e um arquivo vazio, as etapas a seguir são executadas por iteração (71).

Primeiro, todos os membros da população não dominada são copiados para o arquivo; qualquer indivíduo dominado ou duplicado (em relação aos valores objetivos) são removidos do arquivo durante essa operação de atualização. Se o tamanho do arquivo atualizado exceder um valor limite predefinido, outros membros do arquivo são excluídos por uma técnica de *clustering* que preserva as características da frente não dominada. Depois, os valores de aptidão são atribuídos para os membros do arquivo e da população (71).

O próximo passo representa a fase de seleção de acasalamento, em que os indivíduos da união de população e arquivo são selecionados por meio de torneios binários. Observe que a aptidão deve ser minimizada aqui, ou seja, cada indivíduo no arquivo tem mais chances de ser selecionado do que qualquer membro da população. Finalmente, após a recombinação e mutação, a população idosa é substituída pela população de descendentes resultante (71).

Embora a SPEA tenha tido um bom desempenho em diferentes estudos comparativos (70) e (72), ainda há espaço para melhorias relacionadas a possíveis fraquezas do SPEA associadas ao trabalho de Fitness, a estimativa de densidades e truncamento de arquivo (71). A seguir está descrito o algoritmo aprimorado denominado SPEA2 (71).

O algoritmo SPEA2, proposto por Zitzler, é a versão aprimorada do SPEA, que pode obter a solução Pareto distribuída em ordem por truncamento e controle do conjunto de arquivos. O SPEA2 é considerado um algoritmo evolucionário multiobjetivo bem-sucedido, possui poucos parâmetros de configuração, velocidade convergente rápida, boa robustez e conjuntos de soluções distribuídas ordenadamente (73).

O SPEA2 foi projetado para superar os problemas identificados no SPEA. Em contraste com a SPEA, o SPEA2 usa uma estratégia de atribuição de exercícios, incorpora informações de densidade. Além disso, o tamanho do arquivo é fixo, ou seja, sempre que o número de indivíduos não dominados é menor do que o tamanho de arquivo predefinido,

o arquivo é preenchido por indivíduos dominados. Além disso, a técnica de agrupamento, que é invocada quando a frente não nomeada excede o limite de arquivo, foi substituída por um método de truncamento alternativo que possui características semelhantes, mas não perde limites pontos (71).

O SPEA2 demonstra constituir significativa melhoria em relação ao seu antecessor, o SPEA, à medida que alcança melhores resultados em todos os problemas considerados (71).

2.2.3.3 *MultiObjective Cellular genetic algorithm* (MOCell)

MOCell (*MultiObjective Cellular genetic algorithm*) é uma família de algoritmos genéticos celulares (cGAs) para otimização multiobjetivo, que é o resultado da combinação de diferentes estratégias (síncrona vs assíncrona e dois esquemas de retroalimentação de arquivos). Em cGAs, o conceito de vizinhança (pequena) é intensivamente utilizado; isso significa que um indivíduo pode cooperar apenas com os seus vizinhos próximos no ciclo de reprodução. A sobreposição de pequenas vizinhanças de cGAs ajuda a explorar o espaço de busca, porque a lenta difusão induzida de soluções por meio da população fornece uma espécie de exploração (diversificação), enquanto a exploração (intensificação) ocorre dentro de cada vizinhança por operações genéticas. Além disso, a vizinhança é definida entre soluções provisórias no algoritmo, sem relação com a definição de vizinhança geográfica no espaço do problema (59).

Nesta seção, detalhamos primeiro uma descrição de um cGA canônico, para só então descreveremos o algoritmo MOCell.

Algoritmos genéticos celulares

Figura 5 – Funcionamento de um algoritmo genético celular canônico

Algorithm 1 Pseudocode for a Canonical cGA.

```
1.  proc Steps_Up(cga)      //Algorithm parameters in 'cga'
2.  while not Termination_Condition() do
3.    for individual ← 1 to cga.popSize do
4.      n_list←Get_Neighborhood(cga,position(individual));
5.      parents←Selection(n_list);
6.      offspring←Recombination(cga.Pc,parents);
7.      offspring←Mutation(cga.Pm,offspring);
8.      Evaluate_Fitness(offspring);
9.      Insert(position(individual),offspring,cga,aux_pop);
10.   end for
11.   cga.pop←aux_pop;
12. end while
13. end proc Steps_Up;
```

Fonte: (59).

Um cGA canônico segue o pseudocódigo incluído no algoritmo 1 demonstrado na Figura 5. Neste cGA básico, a população é geralmente estruturada em uma série regular de d dimensões (d = 1, 2, 3), e uma vizinhança é definida nela. O algoritmo considera iterativamente como atual cada indivíduo na grade (linha 3). Um indivíduo só poder interagir com indivíduos pertencentes à sua vizinhança (linha4), então seus pais são escolhidos entre seus vizinhos (linha 5) com determinado critério. Operadores de recombinação e mutação são aplicados aos indivíduos nas linhas 6 e 7, com probabilidades Pc e Pm, respectivamente. Posteriormente, o algoritmo calcula o valor de adequação do novo indivíduo da prole (ou indivíduos) (linha 8) e insere (ou um deles) no local equivalente ao indivíduo atual na nova população (auxiliar) (linha 9) após determinada política de substituição. Depois de aplicar esse ciclo reprodutivo a todos os indivíduos da população, a população auxiliar recém-gerada é assumida como a nova população para a próxima geração (linha 11). Esse *loop* é repetido até que uma condição de terminação seja encontrada (linha 2). As condições de término mais comuns são atingir o valor ideal (se conhecido), para realizar um número máximo de avaliações da função de aptidão ou combinação de ambos (59).

O multiobjetivo cGA: MOCell

O multiobjetivo celular (MOCell) é baseado no modelo celular de algoritmos evolutivos (AGCs) com a suposição de que apenas um indivíduo interage com seus vizinhos durante o processo de busca. Além disso, o MOCell armazena um conjunto de soluções individuais de objetos não dominados em um arquivo externo. Depois de cada geração, o MOCell substitui um número fixo de soluções escolhidas aleatoriamente da população, selecionando o mesmo número de soluções do arquivo até que as condições de rescisão sejam atendidas. Tal substituição só ocorre quando soluções recém-geradas da população são piores do que as do arquivo (65).

Seu pseudocódigo é dado no algoritmo 2 demonstrado na Figura 6. Podemos observar que os algoritmos 1 e 2 são muito semelhantes. Uma das principais diferenças entre eles é a existência de uma frente de Pareto (definição 5) no caso multiobjetivo. A frente de Pareto é apenas uma população adicional (um arquivo externo) composta de um número de soluções não dominadas encontradas, já que tem tamanho máximo. Para gerenciar a inserção de soluções na frente de Pareto e com o objetivo de obter um conjunto diversificado, utiliza-se um estimador de densidade na distância de superposição (proposta para NSGA-II). Essa medida também é usada para remover soluções do arquivo quando ele fica cheio. O MOCell pode ser considerado o próximo passo para usar o modelo celular canônico de GAs no campo multiobjetivo (59).

O MOCell começa criando uma frente de Pareto vazia (linha 2 no algoritmo 2). Indivíduos estão dispostos em uma grade toroidal bidimensional e os operadores genéticos são sucessivamente aplicados a eles (linhas 7 e 8) até que a condição de término seja

Figura 6 – Funcionamento do MOCell

MOCELL: A CELLULAR GENETIC ALGORITHM

Algorithm 2 Pseudocode of MOCell.

```
1.  proc Steps_Up(mocell)        //Algorithm parameters in 'mocell'
2.  Pareto_front = Create_Front() //Creates an empty Pareto 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: (59).

encontrada (linha 3). Assim, por cada indivíduo, o algoritmo está na escolha de dois pais de sua vizinhança, recombinando-os para obter uma descendência, mutando-a e avaliando o resultado individual. Esse indivíduo é então solicitado a inserir-se na população auxiliar no arquivo externo. No primeiro caso, a prole resultante substitui o indivíduo na posição atual, se este último for pior do que o anterior, mas, como é comum na otimização multiobjetivo, precisamos definir o conceito de "melhor indivíduo". Nossa abordagem é substituir o indivíduo atual, se ele for dominado pelos descendentes, ou ambos são nã -dominados e o indivíduo atual tem a pior distância de superposição (como definido no NSGA-II) em uma população composta de C9 ou vizinhos Compact9 (59).

Para inserir o indivíduo na frente de Pareto, as soluções no arquivo também são classificadas de acordo com a distância de superlotação; portanto, ao inserir um solução não dominada, se a frente de Pareto já estiver cheia, a solução com o pior valor de distância de superposição é removida. Finalmente, após cada geração, a antiga população é substituída pela auxiliar, e um procedimento de *feedback* é invocado (linha 14) para substituir um número fixo de indivíduos escolhidos aleatoriamente da população pelo mesmo número de soluções retiradas do arquivo (59).

O MOCell foi projetado com o mesmo mecanismo de controle de restrições do NSGA-II para lidar com MOPs restritos. Sempre que dois indivíduos são comparados, suas restrições são avaliadas. Se ambos são viáveis, um domínio de teste de Pareto (definição 3) é aplicado diretamente. Se um é viável e o outro é inviável, o possível domina. Caso contrário, se ambos os indivíduos são inviáveis, então aquele com a menor quantidade de violação de restrição domina o outro (59).

Tabela 1 – Questões de pesquisa da RSL

Id	Questões
Q1.	Quais as técnicas computacionais que podem auxiliar na alocação de times ágeis de desenvolvimento de *software*?
Q2.	A qual área da Engenharia de *Software* o estudo se refere?
Q3.	O estudo foi aplicado em um processo ágil?

Fonte: dados da pesquisa.

2.3 Metodologia da revisão sistemática da literatura

Para alcançar os objetivos deste trabalho, uma revisão sistemática de literatura (RSL) foi realizada com o intuito de identificar, compreender e comparar as literaturas existentes, a fim de obter relevante fundamentação e identificar técnicas computacionais aplicadas nos problemas da Engenharia de *Software* (no campo da pesquisa da técnica SBSE), bem como a identificação de lacunas e oportunidades para investigações posteriores.

2.3.1 Questões de pesquisa da revisão sistemática da literatura:

A definição das questões de pesquisa é o passo mais crítico ao realizar uma revisão sistemática da literatura (74). Essas questões orientam a construção do protocolo da revisão e auxiliam na identificação do escopo do trabalho (19). As questões de pesquisa definidas para este estudo são apresentadas na Tabela 1 e pretendem ajudar a identificar e analisar a literatura sobre o uso de técnicas computacionais nos problemas de Engenharia de *Software*.

2.3.2 Estratégias de pesquisa

Como estratégia para realizar esta pesquisa, no primeiro momento foram definidos os termos para realizar busca nas bases de dados. A segunda etapa consistiu na seleção das bibliotecas digitais. Em sequência, foram definidos os critérios de inclusão e exclusão dos documentos relevantes para a pesquisa.

2.3.3 Termos para pesquisa

A pesquisa foi realizada entre os dias 10 e 14 de fevereiro de 2018 e um termo de busca comparável entre as bases foi definido para garantir a consistência da pesquisa, observando a possibilidade de combinações e inserção de critérios de seleção, como título, *abstract* e palavras-chave. O objetivo foi encontrar estudos que colaborassem para responder as questões propostas.

Termo de busca: SBSE AGILE

2.3.4 Recuperação de dados

Os bancos de dados digitais escolhidos para recuperação de dados foram:

- IEEE *Xplore*
- *Science Direct*
- *Springer*
- *Wiley online*
- ACM Digital

O processo de recuperação de dados consistiu na execução de pesquisa usando os termos propostos em cada uma das bibliotecas digitais selecionadas. A ferramenta de gerenciamento de referências **Zotero** foi utilizada para armazenar os arquivos com suas respectivas informações. Os resultados do número de artigos retornados por cada fonte de dados escolhida estão demonstrados na Tabela 2.

Tabela 2 – Número de artigos retornados de cada fonte

Fonte	URL	Número de Artigos
ACM Digital	http://dl.acm.org/	1.857
IEEE	http://ieeexplore.ieee.org/	22
Science Direct	http://www.sciencedirect.com/	21
Springer Link	https://link.springer.com/	21
Wiley	http://onlinelibrary.wiley.com/	7
TOTAL		1.928

Fonte: dados da pesquisa.

2.3.5 Critérios de seleção

Os critérios de inclusão e exclusão foram utilizados para selecionar os artigos relevantes para o estudo. Procurou-se incluir todos os trabalhos de pesquisa publicados até o dia 14 de fevereiro de 2018, no domínio da Engenharia de *Software*, baseada em pesquisa (SBSE) e que adotaram técnicas de otimização para solução de problemas multiobjetivos. Em cada artigo procuramos o número de objetivos do problema proposto, os algoritmos utilizados, em qual etapa do ciclo de desenvolvimento de *software* foi aplicada a técnica de otimização e o detalhamento da técnica utilizada. Somente foram considerados artigos publicados em inglês e português.

Critérios de inclusão

- Foram incluídos na pesquisa artigos científicos publicados em conferências e jornais.

- Foram incluídos artigos científicos disponíveis nas bases de dados IEEE *Xplore*; *Science Direct*; *Springer link*; *Wiley Library Online*; ACM.

- Foram adotados estudos publicados em inglês e português.

- Foram considerados para o estudo artigos científicos que abordam a aplicação da técnica de SBSE independentemente da área de aplicação na Engenharia de *Software*.

Critérios de exclusão

- Os estudos que não abordaram a aplicação da técnica SBSE em problemas de Engenharia de *Software* ou que não aplicaram algoritmos para otimização do problema estudado ou que não detalharam a aplicação da técnica de otimização.

- Livros, teses, editoriais, prefácios, resumos de artigos, entrevistas, notícias, comentários, correspondência, debates, comentários, cartas do leitor, resumos de tutoriais, *workshops*, painéis, dissertações, sessão de pôsteres e publicações em outros idiomas que não inglês e português.

- Documentos corrompidos ou indisponíveis nas bases de dados.

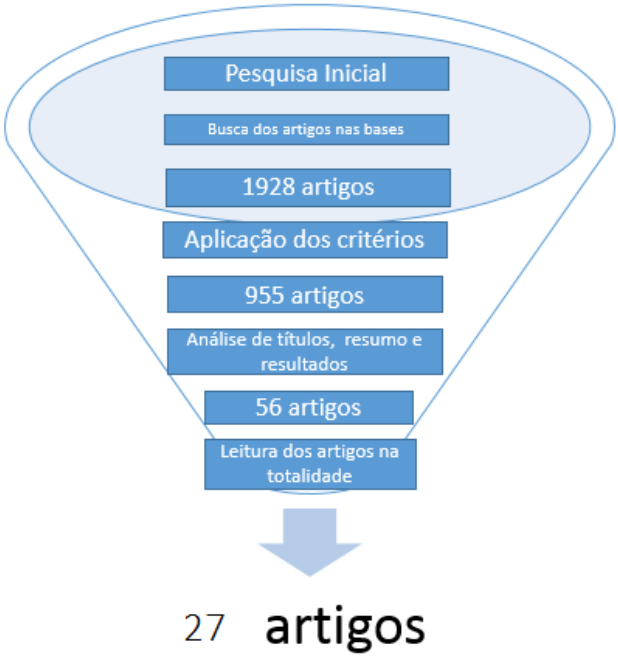
- Documentos duplicados e/ou redundantes.

2.3.6 Triagem dos artigos

A pesquisa foi realizada de forma minuciosa e para garantir sua validade as seções seguintes descrevem as etapas de procura, seleção da literatura, análise dos documentos e a discussão sobre os resultados obtidos.

A investigação foi realizada nas bases por meio da ferramenta de busca avançada e dos campos de códigos que estavam disponíveis nos diferentes bancos de dados. Na **fase 01** foram inseridos os termos nas bases, resultando em 1.928 documentos. Em seguida, na **fase 02**, foram aplicados em cada base os critérios de exclusão, resultando em 955 artigos para serem analisados. Na **fase 03**, o título, resumo e os resultados dos artigos foram lidos e analisados, obtendo-se 56 artigos. Na **fase 04**, os 56 artigos foram lidos em sua totalidade, sendo excluídos 28. Restaram 27 artigos que abordam SBSE e apresentaram os algoritmos utilizados na solução dos problemas apresentados. Nem todos os 27 artigos citam a metodologia ágil. Esses artigos foram classificados por área da Engenharia de *Software* e a técnica de otimização utilizada. Os passos para a triagem dos artigos e os respectivos artigos resultantes de cada passo estão demonstrados na Figura 7.

Figura 7 – Processo de triagem dos artigos da RSL



Classificação: os 27 artigos selecionados foram integralmente lidos, analisados e classificados de acordo com as categorias definidas. As classificações propostas neste trabalho foram baseadas nos trabalhos de Sayyad *et al.* (2013) (75):

1. Os estudos se referem a qual área da Engenharia de *Software*?
2. Quais são os algoritmos utilizados para solucionar o problema apresentado?
3. O estudo foi aplicado em um processo ágil?
4. O problema apresentado é multiobjetivo?

2.3.7 Resultados da revisão sistemática da literatura

Nesta seção são relatados os resultados e as discussões a respeito dos dados extraídos das atividades de classificação dos 27 artigos selecionados para a pesquisa. Os resultados são representados em uma visão gráfica.

Inicialmente, o ano de publicação dos artigos foi considerado com o intuito de identificar a evolução das investigações relacionadas à aplicação da técnica SBSE. A Tabela 3 demonstra o número de artigos publicados por ano. Ressalta-se que neste estudo não houve critério de seleção de artigos por ano, sendo considerados, portanto, todos os artigos publicados e que apresentaram em seu resumo o tema em questão.

Tabela 3 – Ano de publicação dos artigos selecionados na RSL

Ano	Qtde
2008	1
2009	1
2010	2
2012	2
2013	1
2014	5
2015	7
2016	2
2017	4
2018	2
	27

Mesmo apresentando oscilação entre os anos de 2014 e 2015, conforme apresentado na Tabela 4, o número de artigos publicados, ou seja, 12 (44,44%) supera as publicações dos anos anteriores, o que evidencia o crescente interesse dos pesquisadores pelo tema.

Considerando os critérios de classificação:

1. Os estudos se referem a qual área da Engenharia de *Software*?

Tabela 4 – Área da Engenharia de *Software* dos artigos selecionados da RSL

Ano	Qtde
Requisitos	11
Projeto	7
Testes	3
Gerenciamento	5
Vários	1

2. Quais são os algoritmos utilizados para solucionar o problema apresentado?

Tabela 5 – Algoritmos utilizados nos artigos selecionados na RSL

Algoritmo	Qtde
MOCell	00
SPEA2	03
NSGAII	16
ACO	04
GRASP	03
Hill Climbing	2
OUTROS	14

3. O estudo foi aplicado em um processo ágil? Sete artigos foram aplicados em processos ágeis.

4. O problema apresentado é multiobjetivo? Não foi identificado artigo algum que apresentou problema com um único objetivo.

A seguir, na Tabela 6 são listados os 27 artigos que foram selecionados na revisão sistemática da literatura.

Tabela 6 – Lista dos artigos selecionados na RSL

Artigo	Área	Algoritmo
(54)	Estimativa de esforço, Gerenciamento de projetos, Gerenciamento de requisitos	SWAY
(12)	Gerenciamento de requisitos	KEYS2
(39)	Gerenciamento - alocação de recursos	NSGAII
(69)	Gerenciamento de requisitos	NSGAII
(76)	Teste	NSGAII
(77)	*Design*	NSGAII
(78)	*Design*	NSGAII
(79)	*Design*	GA
(80)	Teste	MOEA, NSGAII
(81)	Gerenciamento de requisitos	SWAY, NSGAII , SPEA2
(82)	Gerenciamento	*RANDOM SEARCH*,GA, *HILL CLIMBING*
(4)	Gerenciamento de requisitos	MONRP, NSGAII (USADO PARA COMPARAR)
(83)	Gerenciamento de requisitos	OUTROS
(84)	Gerenciamento/ Alocação de recursos	NSGAII
(85)	Gerenciamento de requisitos	ACO
(86)	Gerenciamento de requisitos	ACO, NSGAII, GRASP
(87)	*Design*	MOREX
(88)	*Design*	MOGP
(89)	Gerenciamento	NSGAII
(90)	Gerenciamento/ Planejamento	NSGAII
(91)	Gerenciamento de requisitos	SPEA2, NSGAII , OUTROS
(92)	*Design*	GA - FERRAMENTA
(93)	*Design*	*simulated annealing* (SA), *genetic algorithms* (GAs) e *multiple ascent hill-climbing* (HCM)
(94)	Gerenciamento de requisitos	NSGAII , ACO, GRASP, MOABC
(95)	Gerenciamento de Requisitos	ACO, NSGAII, GRASP, SPEA2
(96)	Teste	GP
(97)	Gerenciamento de Requisitos	NSGAII

2.3.8 Discussão sobre a revisão sistemática da literatura

Esta seção descreve e discute as descobertas a partir da extração de dados e procedimentos de classificação no contexto das questões de pesquisa.

Q1. Quais as técnicas computacionais que podem auxiliar na alocação de times ágeis de desenvolvimento de *software*?

O estudo identificou, a partir dos artigos selecionados, que os algoritmos NSGAII, SPEA2 estão sendo utilizados fortemente na solução de problemas da Engenharia de *Software* e vêm apresentando resultados bem consistentes. Além disso, apesar de o algoritmo MoCell não ter sido adotado nos artigos selecionados na RSL, ele vem sendo utilizado em comparação a NSGA e SPEA2, como pode ser comprovado no artigo do Nebro *et. al.* (59).

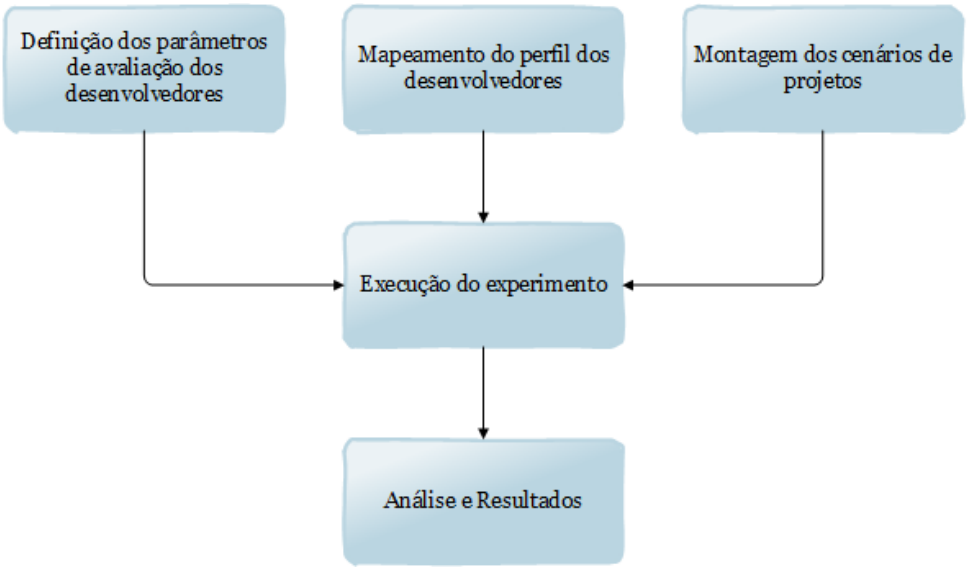
3 MÉTODO DO EXPERIMENTO

Com a realização da revisão sistemática da literatura foram selecionadas as técnicas de otimização a serem utilizadas no experimento. Com essa etapa finalizada, iniciou-se a preparação para a realização do experimento e, na sequência, sua execução.

O experimento foi realizado em dezembro de 2018 em uma empresa de *software* brasileira de grande porte que há cerca de três anos adotou a metodologia ágil - SCRUM. O experimento foi realizado em uma de suas plantas cujo time é composto de 400 analistas entre desenvolvedores, testadores, *Scrum Master*, POs, entre outros. A empresa adota *Sprints* de 15 dias e a *release* tem duração de três meses.

O objetivo principal do experimento foi comparar o desempenho dos algoritmos de otimização multiobjetivos (NSGAII, SPEA2 e o MOCell), selecionados a partir da RSL, no âmbito de gerenciamento de projetos, mais especificamente na alocação de times ágeis de desenvolvimento de *software*. Todos os algoritmos usados neste trabalho (NSGA-II, SPEA2 e MOCELL) foram implementados usando o *framework* jMetal (Versão 4.5.2). jMetal significa algoritmos meta-heurísticos em Java e é uma estrutura baseada em Java orientada a objetos que visa facilitar o desenvolvimento de meta-heurísticas para resolver os MOPs. jMetal fornece um rico conjunto de classes, que podem ser usadas como blocos de construção de meta-heurísticas multiobjetivas; assim, aproveitando a reutilização de código, os algoritmos compartilham os mesmos componentes básicos, tais como implementações de operadores genéticos e estimadores de densidade, fazendo assim a comparação justa de meta-heurísticas diferentes para MOPs possíveis.

Figura 8 – Fluxo de atividades do experimento



Fonte: dados da pesquisa.

O experimento, conforme ilustrado na Figura 10, foi realizado em cinco etapas:

1. Definição dos parâmetros de avaliação dos desenvolvedores.
2. Mapeamento do perfil dos times de desenvolvimento: nesta etapa foram envolvidos os gestores de pessoas e o PMO.
3. Montagem dos cenários de projeto para a realização dos testes: nesta etapa foram envolvidos os *Master POs*.
4. Execução do experimento: nesta etapa foram envolvidos os *Scrum Masters* .
5. Compilação e análise dos resultados.

3.1 Definição dos parâmetros de avaliação dos desenvolvedores

O primeiro passo foi identificar os parâmetros a serem considerados no mapeamento do perfil dos analistas.

Os desenvolvedores foram avaliados com base nas seguintes características:

- Competências: conhecimento, habilidades, atitudes, cultura.
- Remuneração.
- Produtividade e assertividade.

As definições desses parâmetros baseou-se nos conceitos de KSA abordados na seção 2.2.1.1 e nas características dos time ágil descritas na seção 2.2.1. A escolha está em linha com os objetivos propostos no trabalho em que na comparação das técnicas de otimização buscou-se identificar o número de recursos humanos necessários para o projeto, considerando melhor qualificação e baixo custo.

3.1.1 Parâmetros de competência da avaliação dos desenvolvedores

Para o Conhecimento, foram consideradas as fontes de conhecimento formal e informal utilizadas na realização das tarefas diárias. Nesta área, considerou-se o grau de conhecimento do desenvolvedor relacionado a vários temas: tecnologia, processos, produto, projeto, entre outros. O conhecimento foi classificado em categorias conforme a seguir:

1. Tecnologia: linguagem C#, *WebServices*, *Javascript*, desenvolvimento *Mobile*, Asp .Net MVC, .Net *Core*, CSS3, Mensageria (integrações), *Framework* .Net proprietário, HTML 5, *Winforms*, TDD, Angular JS, *Webforms*, Xamarim, PHP, *Framework* HTML proprietário, ADVPL.

2. Testes: teste de unidade, teste de instalação, teste de regressão, automação de teste, teste funcional, estratégia de teste, teste de desempenho.
3. Banco de dados: *Oracle* (administração), *SQL Server* (administração), *Oracle* (sintaxe), *SQL Server* (sintaxe), DML, DDL.
4. Ferramentas: TFS, Jira, *Visual Studio*
5. Design: *Photoshop*, *Design Think*, *Ilustrator*, *InDesign*, Pacote Adobe
6. Processos: metodologia ágil, Canvas, *Scrum*
7. Produto: módulos de produto desenvolvidos pela empresa

Cada categoria recebeu um peso e o conhecimento foi avaliado em cinco níveis: muito baixo, baixo, médio, alto, muito alto. Os dados foram obtidos a partir do sistema de gestão de capital humano da empresa e complementado com a avaliação realizada pelos gestores de pessoas dos times estudados juntamente com o *Scrum Master* e o dono do produto.

Para as habilidades foram consideradas as características adquiridas a partir da experiência. Elas foram classificadas em categorias conforme a seguir:

1. Comunicação.
2. Colaboração.
3. Criatividade.
4. Autonomia.
5. Inteligência emocional.
6. Curiosidade.
7. Lógica de programação.
8. Visão estratégica.
9. Liderança.
10. Auto gerenciamento.

Para cada categoria, o desenvolvedor foi avaliado em cinco níveis: muito baixo, baixo, médio, alto, muito alto. Os dados foram obtidos a partir da avaliação realizada pelos gestores de pessoas dos times estudados, juntamente com o *Scrum Master* e o dono do produto.

As atitudes são relacionadas ao comportamento registrado nas tarefas realizadas diariamente na empresa. As atitudes foram classificadas em categorias conforme a seguir:

1. Disciplina.
2. Relacionamento.
3. Iniciativa.
4. Motivação.
5. Interesse e dedicação.
6. Protagonismo.
7. Comprometimento.
8. Maestria.
9. Facilitador.

Para cada categoria, o desenvolvedor foi avaliado em cinco níveis: muito baixo, baixo, médio, alto, muito alto. Os dados foram obtidos a partir da avaliação realizada pelos gestores de pessoas dos time estudados, juntamente com o *Scrum Master* e o dono do produto.

No quesito cultura foi levada em consideração a nota resultante da **avaliação de desempenho** utilizada na empresa de *software* estudada, como aspecto organizacional que influencia o desempenho. As informações foram obtidas por meio da extração automática dos dados no *software* de gestão de capital humano da empresa.

3.1.2 Parâmetros de remuneração da avaliação dos desenvolvedores

Representa a remuneração do desenvolvedor. Os dados relacionados a salário foram obtidos a partir da ferramenta de gestão de folha de pagamento, mediante autorização expressa da diretoria responsável. Entretanto, para fins de confidencialidade das informações, elas foram estratificadas em níveis de carreira e graduadas de zero a 12.

3.1.3 Parâmetros de produtividade e assertividade da avaliação dos desenvolvedores

Os dados relacionados à produtividade e assertividade foram coletados com base nas informações por *Squad* e *Sprint* fornecidas pelos *Scrum Masters* para a equipe de escritório de projetos (PMO).

3.2 Mapeamento do perfil dos times de desenvolvimento

Esta etapa teve como objetivo mapear o perfil dos profissionais que atuam no desenvolvimento, nos segmentos de negócios escolhidos para o estudo. Este mapeamento foi realizado a partir dos parâmetros detalhados na seção anterior: competências (conhecimento técnico, habilidades, atitude), remuneração, produtividade e assertividade na atuação dos projetos.

3.2.1 Coleta das competências dos desenvolvedores

Dois gestores de pessoas mapearam 84 competências de 86 analistas, distribuídos em 15 *Squads*, além da assertividade e produtividade dos times na atuação nos projetos que participaram entre setembro de 2017 e outubro de 2018.

- Foram considerados os *Scrum Masters* na amostra.
- Os *POs* não foram considerados na amostra.

As categorias mapeadas foram descritas na seção 3.1.

3.2.2 Coleta da remuneração dos desenvolvedores

Foi coletado o salário de cada analista no sistema de gestão de folha de pagamento e realizada a correspondência com o nível de carreira considerando uma escala de zero a 12.

3.2.3 Coleta da produtividade dos desenvolvedores

A equipe de PMO disponibilizou informações do período de setembro de 2017 ao início de outubro de 2018, referentes a:

- *Squad* - identificando a composição de cada *Squad* (equipe).
- *Sprint* - identificando cada *Sprint* realizada, seu período e dados relacionados.
- Analista - identificação do colaborador .
- Dias úteis - o número de dias úteis que o analista participou da *Sprint*.
- Planejado e velocidade - o número de pontos planejados e entregues pela *Squad*.
- A participação do analista (colaborador) na *Sprint* (poderia haver férias ou folga durante a *Sprint*, por exemplo, que tornava essa informação variável).

Considerando a produtividade como:

$$Produtividade = \frac{PontosEntregues}{Dias} \tag{3.1}$$

Assim, pode-se definir a produtividade de cada *Sprint*, como sendo a aplicação da Fórmula 3.3 para a agregação dos dados disponíveis por *Sprint* e a produtividade por *Squad* a aplicação da mesma para a agregação de dados disponíveis por *Squad*.

Um ponto de atenção: cada *Squad* tem sua própria escala de pontos, uma vez que a pontuação para cada tarefa é dada com base em referências de pontuação distintas, como: ajuste de um rótulo, criação de um cadastro simples ou implementação de um serviço *Web API* simples, por exemplo. Tais definições não possuem a mesma complexidade e esforço de trabalho. Desta forma, não se pode considerar, por exemplo, que uma determinada *Squad* é a mais produtiva da empresa, tomando como parâmetro somente a quantidade de pontos que entregam.

Nesse sentido, há falta de informações para comparar a produtividade de equipes distintas (como, por exemplo, o número de pontos de função). Dessa forma, para considerar informações na mesma escala, dividiu-se a velocidade de cada *Sprint* pela velocidade média da equipe. É como se considerássemos que cada equipe possuísse uma média de produtividade de 1 ponto (real) e trabalhássemos somente com a informação de variabilidade de produtividade de cada analista conforme essa referência. Assim, temos a produtividade da *Sprint* pela média da equipe, que será chamada de produtividade da *Sprint* normalizada.

$$P_Normal(s) = \frac{P_{Sprint}(s)}{P_{Squad}} \tag{3.2}$$

Calculando a média da produtividade da *Sprint* normalizada, ponderada pela participação (em dias úteis) de cada analista, obtém-se a produtividade de cada analista em relação à produtividade média da equipe, conforme descrito as primeiras 10 linhas na Figura 9. Desta forma, este número representa a produtividade do analista em relação a média da equipe.

$$P_Analista(i) = \frac{\sum_{j=1}^{n}(P_Normal(j) * D_j)}{\sum_{j=1}^{n} D_j} \tag{3.3}$$

Tomando um exemplo didático: se uma equipe entrega em média 50 pontos por *Sprint* de 10 dias úteis e as *Sprints* que o Analista A participou foram entregues em média 45 pontos e as que o analista B participou foram entregues em média 60 pontos, considerando que ambos são da mesma *Squad*, os analistas terão a produtividade calculada em 0,9 e 1,2 respectivamente.

Figura 9 – Produtividade média por analista

Colaborador	PROD_MEDIA
1	1.036331
2	0.978308
3	0.997330
4	1.013673
5	1.013673
6	0.987088
7	0.994470
8	1.003580
9	1.010217
10	1.017039

Fonte: dados da pesquisa.

O analista menos produtivo possui 68% de produtividade em relação à média da equipe, e o mais produtivo 118%. Esses valores foram utilizados como produtividade pelo sistema. Tendo em vista que não há informações sobre a produtividade dos *Scrum Masters*, a produtividade atribuída a eles foi a média (um).

3.2.4 Coleta da assertividade dos desenvolvedores

A equipe de PMO disponibilizou informações, também referentes ao período de setembro de 2017 ao início de outubro de 2018, relativos à assertividade, dada como:

$$A_{PMO} = \frac{PontosEntregues}{PontosPlanejados}$$

tal que:

$$0 \leq A_{PMO} \leq 200$$

Entretanto, tendo em vista a possibilidade dos pontos entregues superarem os pontos planejados, existe a possibilidade de o número se apresentar superior a 100%. Nesse sentido, tem-se a necessidade de ajustar os dados de maneira a serem mais compatíveis com o conceito de assertividade, desta forma:

$$Assertividade = \begin{cases} A_{PMO}, & \text{se } 0 \leq A_{PMO} \leq 100 \\ 200 - A_{PMO}, & \text{senão} \end{cases}$$

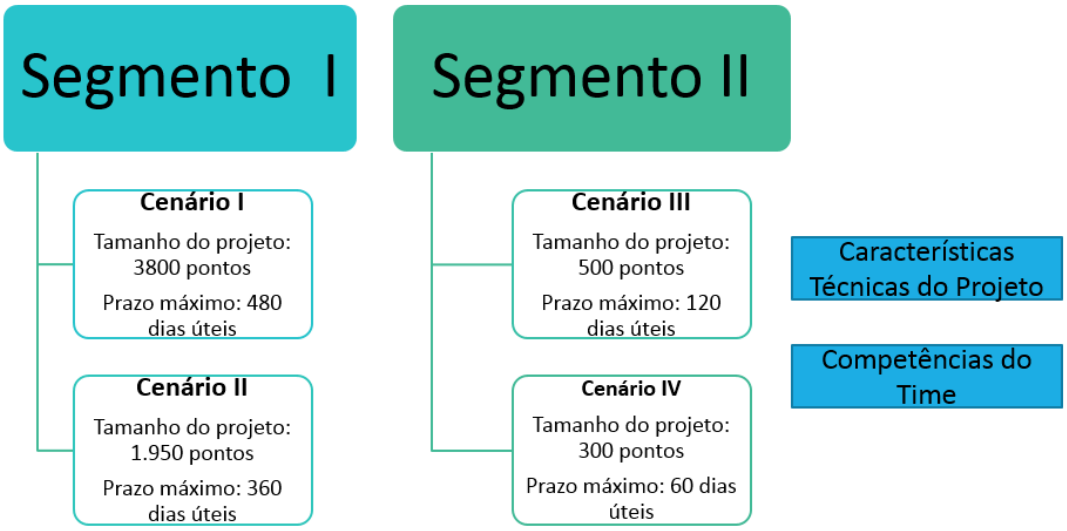
de forma com que:

$$0 \leq Assertividade \leq 100$$

3.3 Montagem dos cenários de projeto para a realização dos testes

Todo o contexto do experimento foi explicado aos dois MPOs dos segmentos escolhidos para o estudo, que selecionaram dois cenários reais de projeto já realizados na empresa. Para cada cenário, levantaram o esforço gasto para a realização do projeto (esforço medido em pontos), o tempo que tiveram para realizar o projeto (data limite para entrega aos clientes), os requisitos técnicos e comportamentais necessários aos recursos humanos participantes do projeto (dentre os descritos na seção 3.1.1). A pontuação já foi normalizada, como descrito na coleta de produtividade. Os cenários propostos por eles são os descritos a seguir e demonstrados na Figura 10.

Figura 10 – Cenários de projeto para realização dos testes



Fonte: dados da pesquisa.

3.3.1 Cenário I dos testes

Construção de indicadores de contabilidade para o segmento *Core* a serem utilizados na solução de BI. O contexto é de um projeto de construção de indicadores, métricas e relatórios para o segmento *Core*. Pode-se dividir o projeto em três partes: **extratores** consultas SQL no banco de dados *SQL Server* e *Oracle*; **modelagem de *Business Inteligence* (BI):** modelagem das tabelas referentes aos fatos e dimensões, tomando por base a granularidade e a relevância dos dados; ***Dashboards*:** construção de métricas, relatórios e gráficos para visualização das informações. Conhecimentos desejados: conhecimento médio no *Core*, banco de dados, conceitos de BI, plataforma *Gooddata*. Requer profissional com boa lógica de programação e que trabalhe de forma colaborativa, pois foi necessário obter informações de clientes e de várias áreas. Nas Tabelas 7 e 8 estão descritos os parâmetros específicos deste cenário I com o detalhamento. A Tabela 7 descreve o

tamanho e o prazo do projeto e a Tabela 8 mostra o nível de conhecimento, em uma escala de zero a cinco, exigido para cada área/subárea mapeadas como requisitos do projeto.

Tabela 7 – Características do cenário I dos testes

Tamanho do projeto:	3800 pontos
Prazo máximo:	480 dias úteis

Tabela 8 – Competências necessárias no cenário I dos testes

Área	Subárea	Conhecimento
Banco de dados	*Oracle* (sintaxe)	4
	SQL Server (sintaxe)	4
	DDL	2
	DML	2
Produto	*Core*	3
	Gooddata	2
Habilidade	Autonomia	2
	Lógica de programação	2
	Curiosidade	1
Atitude	Iniciativa	2
	Relacionamento	2
	Motivação	1

Fonte: dados da pesquisa.

3.3.2 Cenário II dos testes

Construção de uma solução de análise preditiva. O contexto é de uma solução que pode ser desenvolvida em três partes:

1. Modelagem da solução de acordo com suas as características específicas e que servirá de insumo para a construção do algoritmo de *machine e-learning*.

2. Modelagem e configuração do processo de MDM e dos *data models* de armazenagem dos dados de entrada e saída do processo.

3. Construção do portal *Web* para exibição dos indicadores de predição e da gestão.

Não está contemplado neste escopo o desenvolvimento do algoritmo de *machine e-learning*, que é realizado exclusivamente pelo time de *data science*. O time apenas modela o problema/solução que servirão de insumos para a construção do algoritmo. Os conhecimentos desejados: Angular JS, *Type Script*, HTML, CSS, conhecimento médio do *Core*, plataforma de inteligência artificial proprietária, banco de dados. Requer profissional com boa lógica de programação, com muita iniciativa, autodidata, facilidade de

relacionamento e automotivado. Isso porque é um projeto inovador que requer bastante investigação, experimentação. Nas Tabelas 9 e 10 estão descritos os parâmetros específicos desse cenário. A Tabela 9 descreve o tamanho e o prazo do projeto e a Tabela 10 relata o nível de conhecimento exigido, em uma escala de zero a cinco, para cada área/ subárea mapeadas como requisitos do projeto.

Tabela 9 – Características do cenário II dos testes

Tamanho do projeto:	1.950 pontos
Prazo máximo:	360 dias úteis

Tabela 10 – Competências necessárias do cenário II dos testes

Área	Subárea	Conhecimento
Tecnologia	AngularJS	2
	JavaScript	3
	CSS3	3
	HTML5	3
Banco de dados	*Oracle* (sintaxe)	2
	SQL Server (sintaxe)	2
	DDL	2
	DML	2
Produto	*Core*	3

Fonte: dados da pesquisa.

3.3.3 Cenário III dos testes

Um cliente do mercado internacional solicitou evoluções do portal de vendas. Foram identificadas necessidades de melhorias de processos e alterações de interface. O contexto é de um projeto de evolução do portal de vendas do parceiro. O portal foi desenvolvimento na tecnologia PHP (*Web, Mobile*). O portal de vendas é *Kernel* da solução *Core* do cliente. As interfaces precisam ser remodeladas de modo a trazer mais usabilidade aos usuários. Os conhecimentos desejados são: PHP, *Mobile*, WEB, conhecimentos de *User Experience* (UX), conhecimentos do produto *Core*. Nas Tabelas 11 e 12 estão descritos alguns parâmetros específicos desse cenário. A Tabela 11 aborda o tamanho e o prazo do projeto e a Tabela 12 exibe o nível de conhecimento exigido, em uma escala de zero a cinco, para cada área/subárea mapeadas como requisitos do projeto.

Tabela 11 – Características do cenário III dos testes

Tamanho do projeto:	500 pontos
Prazo máximo:	120 dias úteis

Tabela 12 – Competências necessárias do cenário III dos testes

Area	Subárea	Conhecimento
Tecnologia	*JavaScript*	2
	Mobile	1
	AngularJS	2
	CSS3	2
	HTML5	2
	PHP	4
Design	*Design Think*	2
Processos	*Scrum*	1

Fonte: dados da pesquisa.

3.3.4 Cenário IV dos testes

O contexto é de um projeto de migração da integração do produto *Core* com o *Backoffice*, para uma *framework* de integração com o modelo de recorte, em que a solução financeira fica responsável pela gestão da carteira de recebíveis da solução *Core*. Os conhecimentos desejados são: *Backoffice*, já ter participado de um projeto de migração de integração no mesmo âmbito, conhecimento da *framework* envolvido, conhecimentos aprofundados da arquitetura de integração atual, conhecimento de arquitetura, senioridade, proatividade, conhecimentos da solução financeira. Nas Tabelas 13 e 14 estão ressaltados alguns parâmetros específicos deste cenário. A Tabela 13 demonstra o tamanho e o prazo do projeto e a Tabela 14 descreve o nível de conhecimento exigido, em uma escala de zero a cinco, para cada área/ subárea mapeadas como requisitos do projeto.

Tabela 13 – Características do cenário IV dos testes

Tamanho do projeto:	300 pontos
Prazo máximo:	60 dias úteis

Tabela 14 – Competências necessárias do cenário IV dos testes

Area	Subárea	Conhecimento
Tecnologia	Mensageria (integrações)	3
	Framework	2
Produto	Financeiro	3
	Backoffice	3
	Core	3
Habilidades	Autonomia	3
	Lógica de programação	2
	Liderança	2
	Autogerenciamento	2
	Colaboração	3
Atitude	Facilitador	3

Fonte: dados da pesquisa.

3.4 Execução dos experimentos

Um grupo de 10 *Scrum Masters*, dos segmentos selecionados para o estudo, foi convidado a uma agenda de três horas para a realização do experimento. Foi elaborado um passo a passo para a realização do experimento. Inicialmente, foram explicados aos participantes todo o contexto da pesquisa e os objetivos do experimento. Foram apresentados os cenários de projeto a serem simulados e os parâmetros do projeto definidos pelo MPO do segmento em questão. Com essas informações, o grupo foi convidado a repassar todas as informações e validá-las. A partir do consenso do grupo, quanto aos parâmetros do projeto, a simulação foi iniciada. O MPO participou dessa reunião para esclarecer as dúvidas dos SM, quanto ao escopo do projeto, etc.

Cada participante tinha como objetivo determinar o número de analistas necessários para o projeto, bem como escolher os analistas mais adequados para os cenários de projeto estudados, considerando os analistas com mais competência para tal, produtividade suficiente, que atendesse ao tempo solicitado de entrega e que totalizasse o menor custo (menor nível salarial). Os participantes foram orientados quanto ao uso da ferramenta de apoio à seleção do time. Para as simulações, participaram cinco *Scrum Masters* de cada segmento, totalizando 10 pessoas. Assim, cada participante, a partir dos cenários apresentados, realizou individualmente a simulação. Cada um escolheu um time para cada cenário de projeto apresentado. Foram realizadas as seguintes observações:

- O próprio MPO que elaborou o cenário também realizou a simulação em momento anterior, mas seus dados não foram considerados nos resultados, somente para avaliar a adequação do experimento.
- Foram realizadas 20 simulações (quatro cenários e cinco pessoas por cenário).
- Os dados gerados a partir de cada simulação foram gravados em um *log* para posterior análise.
- Após a simulação manual, foram executados os algoritmos de otimização e os resultados também gravados no *log*.

3.5 Modelagem do problema

A solução do problema é modelada como um vetor de binários em que:

- O tamanho do vetor é o tamanho do *pool* de analistas.
- 0 ou 1 em uma posição do vetor representa se o analista naquela posição está presente ou ausente.

O problema possui duas funções-objetivo, minimizar o custo e maximizar a competência, que são descritas na Figura 11.

Figura 11 – Funções-objetivo do problema deste trabalho

$$(1)\ Min \sum_{i=1}^{n} (W_i * D_i + PF)$$

$$(2)\ Max \sum_{i=1}^{n} (K_i * D_i - PF)$$

$$PF = \frac{TamanhoProjeto - Produtividade}{TamanhoProjeto} x\ 100$$

$$Produtividade = DuracaoProjeto * \sum_{i=1}^{n} (P_i * D_i)$$

Fonte: dados da pesquisa.

Em que:

- i é o índice do analista.
- Di é 1 se o analista está presente na solução ou 0 se não está.
- N é a quantidade de analistas no *pool*.
- W é a remuneração do desenvolvedor.
- K é a competência do desenvolvedor.
- H é a habilidade do desenvolvedor.
- AT é a atitude do desenvolvedor.
- PF é o fator de penalização.
- TamanhoProjeto é o tamanho do projeto (em pontos).
- DuracaoProjeto é o número de dias úteis disponível para a realização do projeto.

A produtividade é definida como a soma das produtividades individuais dos analistas selecionados, multiplicado pelo número de dias úteis. O fator de penalização reduz os valores da função-objetivo de forma proporcional a quão distante a produtividade do time está da produtividade-alvo (TamanhoProjeto) em razão desta. Dessa forma, a solução converge para soluções que atendam ao tamanho do projeto demandado no tempo disponível. Esse comportamento é diferente de uma programação linear, em que há restrições

e as soluções que não satisfaçam as restrições são eliminadas. Nos algoritmos evolutivos estudados e implementados no jMetal, as restrições são critério de seleção dos mais aptos para a próxima geração. Assim, entre dois indivíduos, um que apresente restrição e outro que não, aquele que não apresenta restrições será escolhido como mais apto para a próxima geração, ainda que suas funções-objetivo sejam piores. Entre dois indivíduos que apresentem restrições, será escolhido aquele que tiver menor valor de restrição. Assim, a restrição que foi utilizada neste experimento foi:

$$Restrição = \begin{cases} 0, & \text{se} Produtividade \geq TamanhoProjeto \\ TamanhoProjeto - Produtividade, & \text{senão} \end{cases}$$

3.6 Algoritmos utilizados no experimento

Os algoritmos utilizados para a otimização dos times foram descritos em detalhes nas seções 2.2.3.1, 2.2.3.2 e 2.2.3.3. Em resumo, o funcionamento prático deles é descrito a seguir.

3.6.1 NSGA II

1. Gera a população inicial aleatoriamente
2. Realiza o seguinte laço por <Número de Gerações> vezes:
   a) Duplica a população cruzando-a, selecionando a cada vez dois indivíduos por torneio binário e misturando os genes conforme o *crossover* selecionado. Há a possibilidade de mutação do *bit*, com uma probabilidade configurável.
   b) Ranqueia a população conforme a dominância.
   c) Descarta a metade da população menos apta.
   d) Em caso de empate, seleciona aleatoriamente o indivíduo.
3. Ranqueia a população e tem como solução a melhor ranqueada (não dominados). Em caso de mais de uma solução, exibem-se todas.

3.6.2 SPEA2

1. Gera a população inicial Q aleatoriamente.
2. Cria a população P vazia.
3. Realiza o seguinte laço por <Número de Gerações> vezes:

a) Calcula o *fitness* (funções-objetivo) para os indivíduos da população.

b) Q recebe os indivíduos não dominados de P + Q. Caso o número seja maior que a população definida, realiza um corte dos piores *fitness*.

c) É realizado um torneio binário em Q para selecionar os pais, que gerarão a nova população P.

4. Ranqueia a população e tem como solução a melhor ranqueada (não dominados) da população Q. Em caso de mais de uma solução, exibem-se todas.

3.6.3 MOCELL

1. Gera a população inicial aleatoriamente.

2. Cria a população P vazia.

3. Realiza o seguinte laço por <Número de Gerações> vezes:

   a) Para cada indivíduo I da população:

      i. Define oito vizinhos deste indivíduo.

      ii. Escolhe um pai entre os vizinhos, por torneio binário.

      iii. Escolhe um pai na população P, por torneio binário. Se P for vazia, escolhe o outro dentre os vizinhos.

      iv. Cria um novo indivíduo cruzando os pais.

      v. Compara o novo indivíduo com I: adiciona o dominante em P e assume a posição corrente na população (podendo ser escolhido pelos vizinhos).

4. Ranqueia a população P e tem como solução a melhor ranqueada (não dominados) da população Q. Em caso de mais de uma solução, exibem-se todas.

3.6.4 Parâmetros dos algoritmos e do experimento

Os principais parâmetros de cada algoritmo no jMetal são:

- Tamanho da população.

  O tamanho da população inicial, que também é o tamanho da população após cada iteração.

- Número de gerações.

  A quantidade de gerações no ciclo de cruzamento e mutação.

- Formas de representação:

  Binário. Cada gene é representado por um *bit*. Que simboliza a presença ou ausência do analista na solução.

- *Crossover*

  Para a forma de representação binária, no JMetal, podem ser utilizados os seguintes métodos de*crossover*:

  PointCrossOver: uma única posição é selecionada aleatoriamente. Na cópia de ambos os pais, todas as posições após o ponto são trocadas e, dessa forma, são resultados os filhos.

  HUXCrossover (*Half Uniform Crossover*): nesta abordagem, metade dos *bits* diferentes entre os pais são trocados. Para tal, faz-se necessário calcular a quantidade de *bits* diferentes. Divide-se este número por dois e o número resultante é a quantidade de *bits* diferentes que foram trocados entre as cópias dos pais. De igual forma, resultam-se os filhos.

Na execução dos algoritmos, durante o experimento, foram utilizados os parâmetros máximo de gerações, tamanho da população, tamanho do conjunto de variáveis, probabilidade de cruzamento, probabilidade de mutação, listados na Tabela 15 apresentada a seguir:

Tabela 15 – Parâmetros do experimento

Máximo de gerações:	Parametrizável
Tamanho da população:	Parametrizável
Tamanho do conjunto de variáveis:	Número de analistas disponíveis.
Probabilidade de cruzamento:	90%
Probabilidade de mutação:	50% / Número de analistas disponíveis

Fonte: dados da pesquisa.

Neste estudo, não se realizou qualquer tentativa de encontrar a melhor configuração para os parâmetros. Os parâmetros foram definidos com base em trabalhos similares realizados, que tiveram resultados consistentes, e a partir da análise da teoria. A escolha dos parâmetros de mutação e cruzamento baseou-se nos trabalhos apresentados em (84), (86), (98), (99) e (100). Identificou-se que não há consenso na área quanto à configuração dos parâmetros dos algoritmos evolutivos.

Na simulação, os parâmetros número máximo de gerações e o tamanho da população variará conforme o experimento de estudo que é explorado no capítulo 4. O tamanho do conjunto de variáveis é o mesmo do número de analistas disponíveis para formar-se o

time, visto que cada gene representa a presença ou ausência do analista na solução. No capítulo 4 são apresentados os resultados obtidos com essa metodologia.

4 RESULTADOS

Neste capítulo, os resultados são apresentados nas seções: **variação do tamanho da população**, em que o número de gerações é afixado em 250 e o tamanho da população varia para os algoritmos estudados; **variação do número de gerações**, em que o tamanho da população é fixo e o número de gerações varia; **medindo o desempenho dos algoritmos**, em que o tempo de execução é medido, **tamanho da população ou número de gerações**, em que é analisado se é mais interessante aumentar o tamanho da população ou o número de gerações; **escolha do melhor de cada execução**, em que é feita análise considerando somente a melhor solução dada pelo algoritmo em cada execução (uma execução pode retornar mais de uma solução). Por fim, os resultados são discutidos.

4.1 Variação do tamanho da população

Foram executados cada um dos três algoritmos (NSGAII, SPEA2 e MOCell) em suas variantes (com *Single Point Crossover* e *Hux Crossover*) 20 vezes, para o número de gerações de 250 e população variando entre: 100, 200, 300, ..., 800, 900, 1000; para os quatro cenários. Os dados de coleta das execuções dos gestores foram adicionados a esses dados. Posteriormente, o experimento foi complementado com dados da execução do NSGAII e SPEA2 com população variando de 1100, 1200, ..., 1500 e o MOCell com população variando de 1100, 1200, ..., 1900, 2000.

Nas próximas seções são relatados os resultados de todos os cenários. Conforme já exposto na Figura 11, as funções-objetivo do problema proposto são a maximização dos pontos de competência e minimização do custo. Nesse sentido, estas duas informações são apresentadas em destaque. Entretanto, com a finalidade de simplificar a análise nesse contexto, foi calculada a razão de qualidade (pontos de competência) por custo, que representa a quantidade de pontos de competência por nível de remuneração média do time escolhido como solução. A escala dessa razão não é relevante, visto que a escala da qualidade varia de acordo com o cenário. Assim, por exemplo, uma qualidade/custo de 15 no cenário III, representado na Figura 14, não é necessariamente melhor que uma qualidade/custo de 6 no cenário IV, representado na Figura 15, dessa comparação nada se pode concluir. A informação de fato relevante nesta análise é a comparação entre os diversos algoritmos e o resultado dos gestores, dentro de cada cenário. Em síntese, nas Figuras 12, 13, 14 e 15 é apresentada a comparação da média dos resultados para os algoritmos configurados com a maior população, ordenados pelo valor da **qualidade** dividido pelo **custo**, para os quatro cenários do experimento:

Figura 12 – Comparação dos resultados no cenário I variando o tamanho da população

Algoritmo	Qualidade/Custo	Qualidade	Custo	Produtividade
MOCELL_Binary	10.782493	572.023599	54.56 1947	38 18.306254
SPEA2_Binary	10.63 1735	6 17.750000	58. 142857	3800.862857
NSGAII_Binary	10. 126990	662.727273	65.5 15 152	3853.703030
GESTOR	9.502051	6 13.400000	65.400000	3903.954000
NSGAII_Binary_HUXCrossover	8.027 127	10 15.446429	126.982 143	7055.40 1786
MOCELL_Binary_HUXCrossover	7.900434	99 1.764706	125.588235	7005. 17 1176
SPEA2_Binary_HUXCrossover	7.872259	1027.563636	130.527273	722 1.774 182

Fonte: dados da pesquisa.

Figura 13 – Comparação dos resultados no cenário II variando o tamanho da população

Algoritmo	Qualidade/Custo	Qualidade	Custo	Produtividade
SPEA2_Binary	11.884401	429.600000	36.200000	1944.450400
NSGAII_Binary	11.631291	443.764706	38.205882	1945.6 13824
MOCELL_Binary	11. 147568	405.767760	37.40 1639	1942.472732
GESTOR	10.5 18975	458.400000	44.000000	2115.604000
NSGAII_Binary_HUXCrossover	7.543934	952. 125000	126.200000	5348.226750
MOCELL_Binary_HUXCrossover	7.456709	886.485714	118.97 1429	5 104.342857
SPEA2_Binary_HUXCrossover	7.443610	846.000000	114.029412	4836.977647

Fonte: dados da pesquisa.

Comparando a qualidade/custo do melhor algoritmo em relação aos gestores, nota-se que ele entregou, em média, 13,44% (MOCell), 13,03% (SPEA2), 60,11% (NSGAII) e 33,94% (NSGAII) mais qualidade/custo do que a média dos gestores, para os cenários I, II, III e IV, respectivamente.

Como pode ser visto nas Figuras 12, 13, 14 e 15, a qualidade/custo dos algoritmos executados com *Hux Crossover* ficaram abaixo dos que executaram com *Single Point Crossover* em todos os cenários. Assim, serão focados somente os resultados dos algoritmos em *Single Point Crossover*. Além disso, é interessante notar que, apesar de o MOCell ter média de resultados próxima dos demais algoritmos, ele apresenta alguns resultados muito expressivos. As Tabelas 16, 17, 18 e 19 disponibilizam os resultados com melhores

Figura 14 – Comparação dos resultados no cenário III variando o tamanho da população

Algoritmo	Qualidade/Custo	Qualidade	Custo	Produtividade
NSGAII_Binary	15.759171	395.351351	25.162162	301.431892
SPEA2_Binary	15.386973	405.913043	26.434783	301.546957
MOCELL_Binary	13.762170	380.288820	28.673913	300.832919
GESTOR	9.842172	432.200000	44.600000	301.156000
NSGAII_Binary_HUXCrossover	7.741285	917.138889	118.722222	866.447778
SPEA2_Binary_HUXCrossover	7.293457	885.250000	121.468750	860.554375
MOCELL_Binary_HUXCrossover	7.201363	866.833333	120.666667	856.605333

Fonte: dados da pesquisa.

Figura 15 – Comparação dos resultados no cenário IV variando o tamanho da população

Algoritmo	Qualidade/Custo	Qualidade	Custo	Produtividade
SPEA2_Binary	6.404832	136.958333	21.416667	484.972917
NSGAII_Binary	6.214730	152.521739	24.652174	485.479783
MOCELL_Binary	5.354286	124.527344	23.738281	489.093242
GESTOR	4.781672	172.600000	36.800000	576.936000
SPEA2_Binary_HUXCrossover	2.550990	307.440000	120.200000	1722.890000
NSGAII_Binary_HUXCrossover	2.504675	282.185185	112.703704	1653.634074
MOCELL_Binary_HUXCrossover	2.500024	300.312500	120.375000	1770.900625

Fonte: Dados da pesquisa.

qualidade/custo de cada cenário e o respectivo algoritmo. Em todos os cenários, na escolha dos melhores resultados, o Mocell foi o algoritmo selecionado.

Tabela 16 – Resultados com melhor qualidade/custo do cenário I

Algoritmo	Produtividade	População	Gerações	qualidade/custo
MOCell	3846,1	1800	250	15,0625

Fonte: dados da pesquisa.

Tabela 17 – Resultados com melhor qualidade/custo no cenário II

Algoritmo	Produtividade	População	Gerações	Qualidade/Custo
MOCell	1806,52	800	250	16,210526
MOCell	1806,52	900	250	16,210526
MOCell	1806,52	1700	250	16,210526
MOCell	1806,52	1900	250	16,210526
MOCell	1806,52	2000	250	16,210526

Fonte: dados da pesquisa.

Tabela 18 – Resultados com melhor qualidade/custo no cenário III

Algoritmo	Produtividade	População	Gerações	Qualidade/Custo
MOCell	300,85	700	250	18,105263
MOCell	300,66	1400	250	18,176471
MOCell	300,85	1400	250	18,105263
MOCell	300,85	1600	250	18,105263
MOCell	300,85	1600	250	18,105263
MOCell	300,64	1600	250	18,294118
MOCell	300,64	1700	250	18,294118
MOCell	300,85	1700	250	18,105263
MOCell	300,66	1900	250	18,176471
MOCell	300,85	1900	250	18,105263
MOCell	300,64	1900	250	18,294118
MOCell	300,66	1900	250	18,176471
MOCell	300,85	2000	250	18,105263

Fonte: dados da pesquisa.

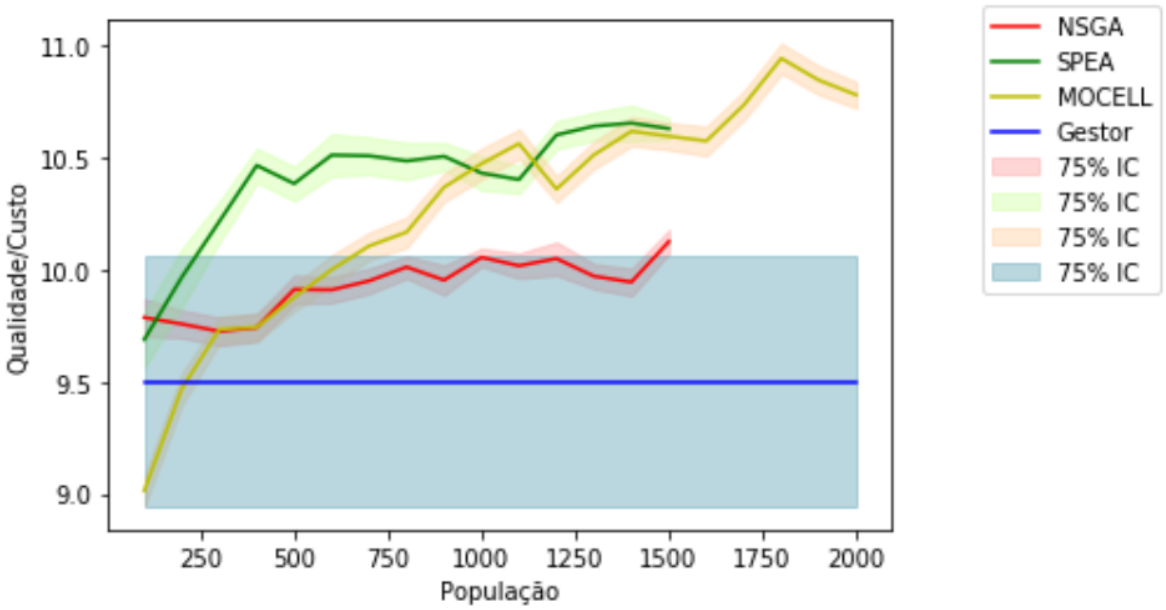
Tabela 19 – Resultados com melhor qualidade/custo no cenário IV

Algoritmo	Produtividade	População	Geração	Qualidade/Custo
MOCell	482,17	900	250	7,142857
MOCell	482,17	900	250	7,142857
MOCell	482,17	1000	250	7,142857
MOCell	482,17	1300	250	7,142857
MOCell	482,17	1300	250	7,142857
MOCell	482,17	1400	250	7,142857
MOCell	482,17	1500	250	7,142857
MOCell	482,17	1700	250	7,142857
MOCell	482,17	1800	250	7,142857
MOCell	482,17	1800	250	7,142857
MOCell	482,17	1900	250	7,142857
MOCell	482,17	1900	250	7,142857
MOCell	482,17	2000	250	7,142857

Fonte: dados da pesquisa.

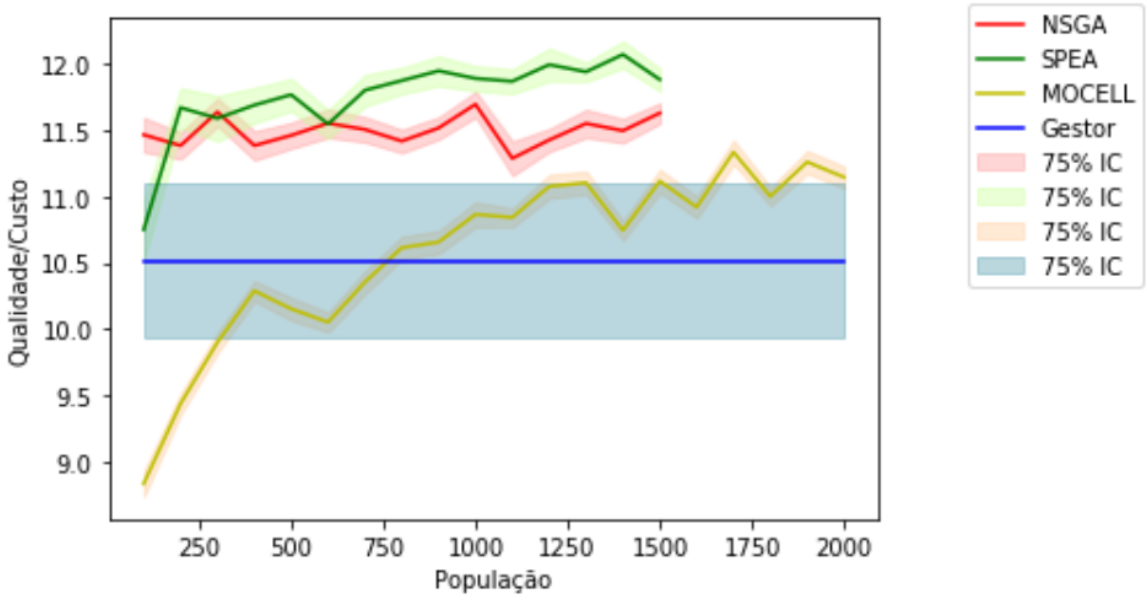
Os melhores resultados, nas Tabelas 16, 17, 18 e 19, foram 58,49%, 54,11%, 85,88% e 49,38% melhores que o resultado médio dos gestores. Resultados expressivamente maiores do que a média do melhor algoritmo do cenário. Além disso, o MOCell ofereceu as melhores soluções dos cenários.

Figura 16 – Evolução da qualidade/custo por população no cenário I



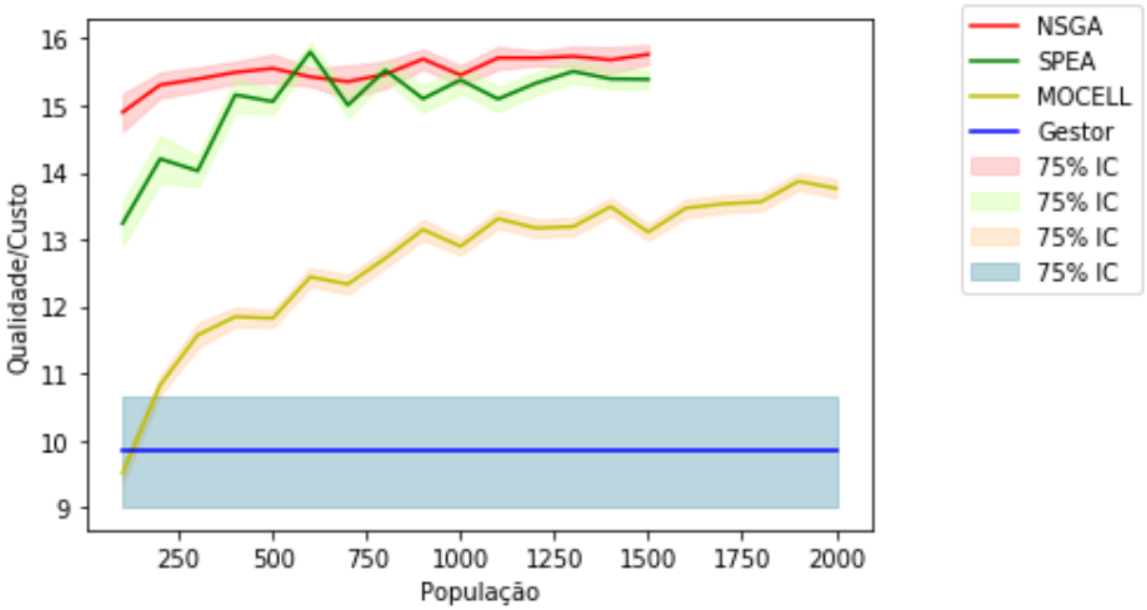
Fonte: dados da pesquisa.

Figura 17 – Evolução da qualidade/custo por população no cenário II



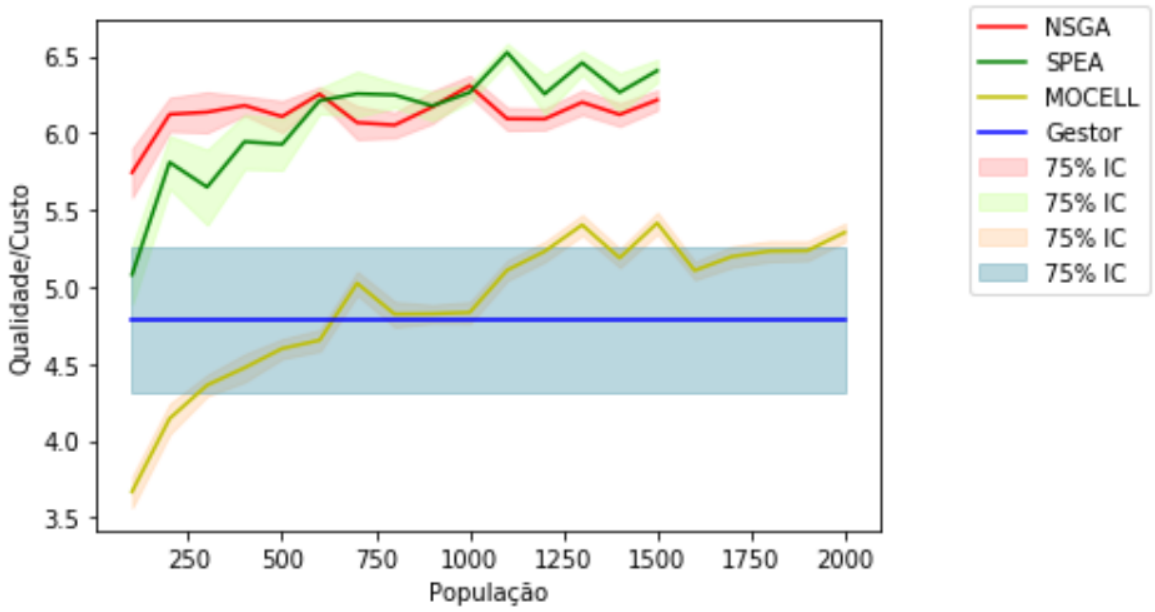
Fonte: dados da pesquisa.

Figura 18 – Evolução da qualidade/custo por população no cenário III



Fonte: dados da pesquisa.

Figura 19 – Evolução da qualidade/custo por população no cenário IV



Fonte: dados da pesquisa.

Na Figura 16 é registrado o resultado da qualidade/custo do cenário I. Nesse contexto, o SPEA2 se manteve à frente, sendo equivalente estatisticamente, com a população em 1.000 e posteriormente sendo superada pelo MOCell. A linha azul representa a média dos resultados das escolhas do gestor. A sombra que envolve a linha é o intervalo de confiança com nível de confiança de 75%. Assim, pode-se notar, com este nível de confiança,

que nesse cenário o SPEA2 e o MOCell tiveram resultados melhores que o gestor. Como foram apurados resultados de cinco gestores somente, inserindo no gráfico o intervalo de confiança dos gestores, ele toma conta de grande parte do gráfico. No que se refere à evolução da qualidade/custo na variação da população do cenário II, o SPEA2 se manteve entre os resultados mais interessantes. O MOCell foi pior (em média) que os gestores até 800 de população, obtendo uma melhora da média após os 1.000 de população, mas não atingiu o resultado dos demais algoritmos. No cenário III, como pode ser observado na Figura 18, o SPEA2 manteve-se entre os melhores, estando pior que o NSGAII para populações inferiores a 500 e estatisticamente empatado de 500 a 1.500 de população. O MOCell ainda evoluiu após 1.000 de população, entretanto, não chegou aos resultados dos outros dois algoritmos. A respeito do cenário IV, os resultados são descritos na Figura 19. O SPEA2 manteve um resultado interessante, equivalente estatisticamente ao NSGAII. O MOCell teve resultado pior até 600 de população e equivalente aos gestores em populações maiores.

4.2 Variação do número de gerações

Nesta seção é descrito o estudo quando se fixa o tamanho da população e varia o número de gerações. A execução variou o número de gerações de 100 a 1.000 e com as populações de 250. De forma sucinta, os resultados foram bem similares aos observados na seção 4.1 e são apresentados a seguir.

Nas Figuras 20, 21, 22 e 23 são feitas as comparações da média dos resultados para os algoritmos parametrizados com a execução com a maior população, ordenados pela qualidade/custo.

Figura 20 – Comparação dos resultados no cenário I variando o número de gerações

Algoritmo	Qualidade/Custo	Qualidade	Custo	Produtividade
MOCELL_Binary	11.049068	587.942697	55.013483	3827.270775
SPEA2_Binary	10.593716	617.545455	58.363636	3804.510909
NSGAII_Binary	10.142449	646.190476	63.809524	3821.165952
GESTOR	9.502051	613.400000	65.400000	3903.954000
MOCELL_Binary_HUXCrossover	8.023560	918.236364	114.872727	6410.448182
NSGAII_Binary_HUXCrossover	7.963580	883.654545	111.163636	6215.005455
SPEA2_Binary_HUXCrossover	7.914911	944.681818	119.590909	6509.203788

Fonte: dados da pesquisa.

Figura 21 – Comparação dos resultados no cenário II variando o número de gerações

Algoritmo	Qualidade/Custo	Qualidade	Custo	Produtividade
SPEA2_Binary	12.063162	426.095238	35.380952	1944.250476
NSGAII_Binary	11.454435	453.388889	39.666667	1944.368333
MOCELL_Binary	11.425208	412.392573	37.143236	1934.900477
GESTOR	10.518975	458.400000	44.000000	2115.604000
NSGAII_Binary_HUXCrossover	7.637796	801.228571	105.114286	4485.417143
SPEA2_Binary_HUXCrossover	7.566196	812.078947	107.263158	4501.394737
MOCELL_Binary_HUXCrossover	7.486541	785.714286	104.809524	4476.075952

Fonte: dados da pesquisa.

Figura 22 – Comparação dos resultados no cenário III variando o número de gerações

Algoritmo	Qualidade/Custo	Qualidade	Custo	Produtividade
NSGAII_Binary	15.818280	392.257143	24.857143	301.357429
SPEA2_Binary	15.317932	405.350000	26.550000	301.472000
MOCELL_Binary	14.270256	386.002331	28.109557	300.831259
GESTOR	9.842172	432.200000	44.600000	301.156000
MOCELL_Binary_HUXCrossover	7.560106	780.294118	103.735294	746.856765
NSGAII_Binary_HUXCrossover	7.309008	716.200000	98.100000	698.939333
SPEA2_Binary_HUXCrossover	7.305341	760.437500	104.500000	739.062187

Fonte: dados da pesquisa.

No cenário I, o melhor algoritmo foi em média 16,28%(MOCell) melhor que o resultado médio dos gestores. Nota-se também melhora no MOCell que nos demais algoritmos quando se aumenta o número de gerações. Nos cenários II, III e IV, o melhor algoritmo entregou 14,64% (SPEA2), 60,73% (NSGAII) e 31,21% (SPEA2) mais qualidade/custo do que a média dos gestores, respectivamente.

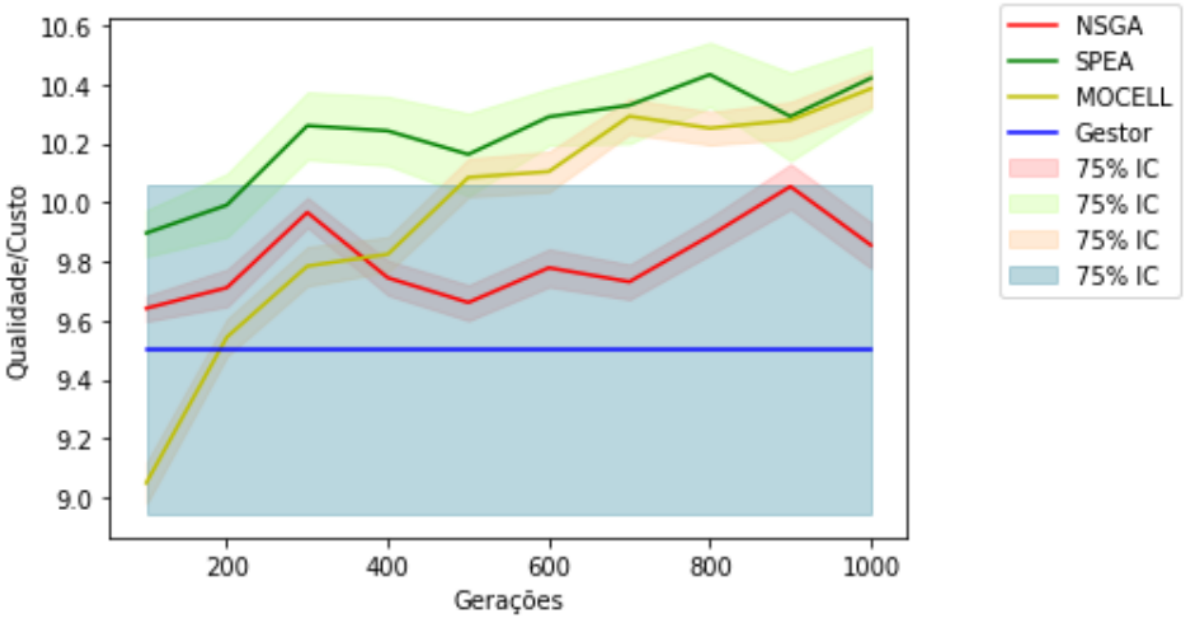
A evolução dos resultados de qualidade/custo em cada cenário, variando o número de gerações, foi similar aos já apresentados na seção 4.1, no que se refere a qual algoritmo teve o melhor desempenho.

Figura 23 – Comparação dos resultados no cenário IV variando o número de gerações

Algoritmo	Qualidade/Custo	Qualidade	Custo	Produtividade
SPEA2_Binary	6.273716	144.565217	23.130435	485.446957
NSGAII_Binary	6.259162	150.147059	24.058824	485.664706
MOCELL_Binary	5.597703	130.904615	23.806154	487.800431
GESTOR	4.781672	172.600000	36.800000	576.936000
MOCELL_Binary_HUXCrossover	2.614374	278.032258	106.419355	1501.169677
NSGAII_Binary_HUXCrossover	2.512408	260.892857	103.678571	1498.142143
SPEA2_Binary_HUXCrossover	2.470503	253.580645	102.193548	1465.225806

Fonte: dados da pesquisa.

Figura 24 – Evolução da qualidade/custo por população no cenário I

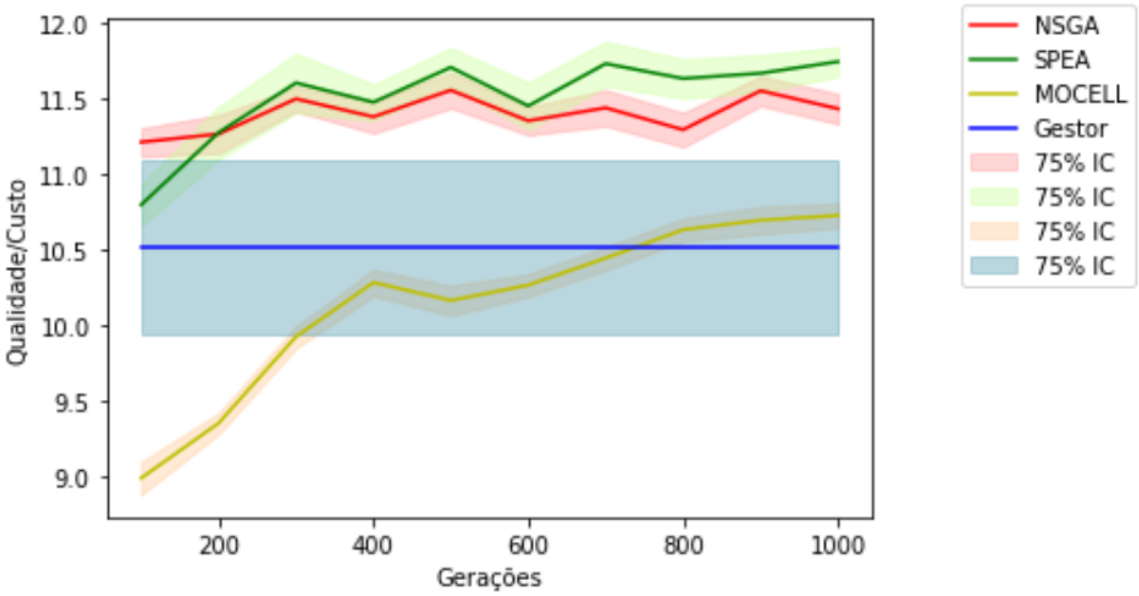


Fonte: dados da pesquisa.

4.3 Medida de desempenho dos algoritmos

A Tabela 20 traz as configurações da máquina em que os testes de desempenho foram executados. A Figura 28 indica a complexidade quadrática do NSGAII e SPEA2 e a complexidade linear do MOCell. O tempo de execução despendido com o MOCell com 2.000 população é aproximadamente o mesmo despendido no NSGAII e SPEA2 com 600 e 500 de população, respectivamente.

Figura 25 – Evolução da qualidade/custo por população no cenário II



Fonte: dados da pesquisa.

Tabela 20 – Configuração da máquina em que o tempo foi medido

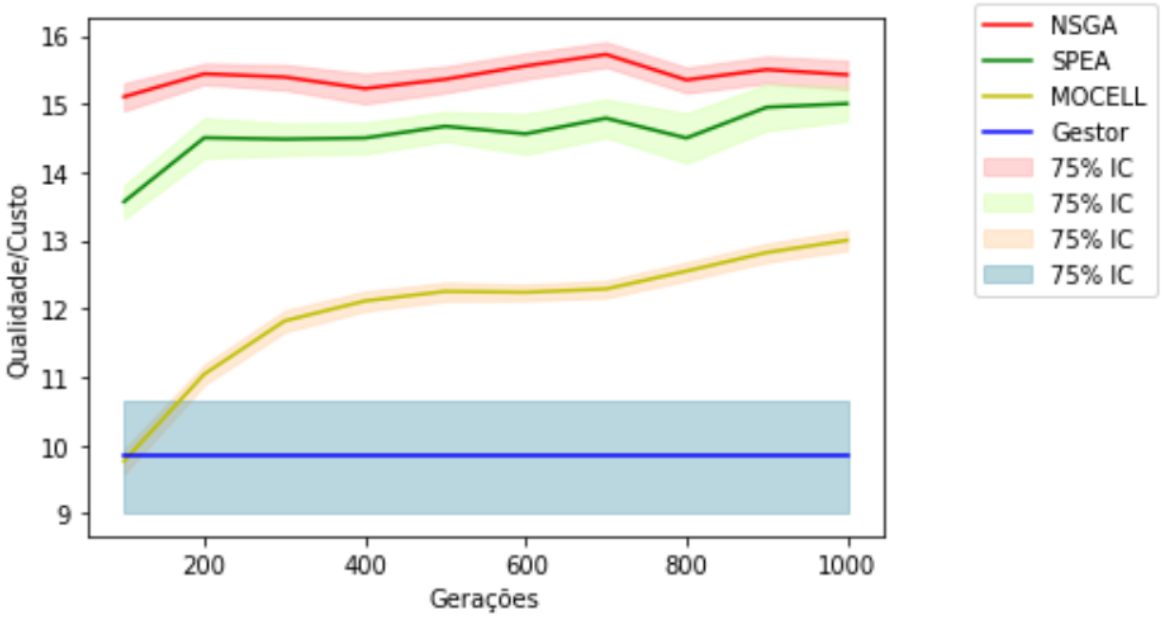
Processador:	Intel Core i5-7200U CPU 2.50GHz 2.71GHz
Memória RAM:	16 GB (Utilizável 15,9 GB)
Tipo de sistema:	*Windows* 64 *bits*

4.4 Tamanho da população ou número de gerações?

Em todos os cenários foi comparada a melhoria média percentual entre 100 e 1.000 de população (fixando-se o número de gerações em 250) com a melhoria média percentual entre 100 e 1.000 gerações (fixando-se o tamanho da população em 250). Os dados são apresentados a seguir.

A Tabela 21 refere que em todos os cenários a melhoria foi maior para o aumento de população do que o aumento do número de gerações. Em todos os cenários a melhoria foi maior para o MOCell, em relação aos outros dois algoritmos. Isso evidencia que o MOCell é mais ganancioso tanto por tamanho de população, quanto número de gerações maiores que os demais algoritmos. Em contrapartida, o NSGAII foi o menos ganancioso. Assim sendo, nota-se a vantagem de investir em população em vez do número de gerações. Entretanto, como pode ser visto na Figura 28, o aumento de população nos algoritmos SPEA2 e NSGAII reflete um aumento quadrático do tempo de execução, enquanto que o aumento do número de gerações é linear. Assim, faz sentido concluir a predileção por aumentar o número de gerações no SPEA2 e NSGAII em vez do tamanho da população e o inverso para o MOCell.

Figura 26 – Evolução da qualidade/custo por população no cenário III



Fonte: dados da pesquisa.

Tabela 21 – Tabela comparativa de melhoria entre incremento de população ou gerações

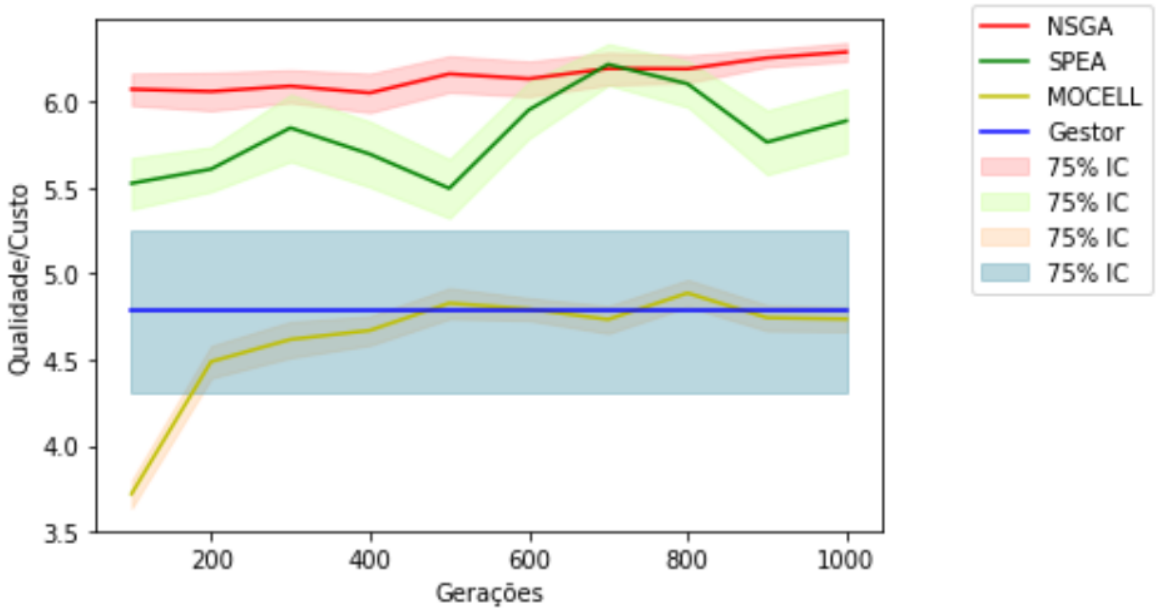
Cenário	Algoritmo	Melhoria (Pop.)	Melhoria (Ger.)
I	SPEA2	7,62	5,31
	NSGAII	2,75	2,22
	MOCell	16,20	14,77
II	SPEA2	10,60	8,77
	NSGAII	2,00	1,97
	MOCell	22,98	19,35
III	SPEA2	16,16	10,59
	NSGAII	3,71	2,14
	MOCell	35,48	33,08
IV	SPEA2	23,28	6,57
	NSGAII	9,81	3,56
	MOCell	31,71	27,30

Fonte: dados da pesquisa.

4.5 Selecionando o melhor resultado de cada execução

Neste estudo, cada uma das 20 execuções foi numerada (cada algoritmo pode retornar mais de uma solução). As soluções foram filtradas de modo a manter somente o melhor resultado de cada execução (o que possui maior valor de qualidade/custo). Esse procedimento tem especial efeito no MOCell, que retorna diversas soluções, visto que o objetivo dele é também fornecer uma amostragem maior e melhor da fronteira de Pareto (59). Nesse sentido, nas Figuras 29, 30, 31 e 32 encontram-se os resultados de evolução da qualidade/custo por algoritmo, escolhendo a melhor solução de cada execução

Figura 27 – Evolução da qualidade/custo por população no cenário IV



Fonte: dados da pesquisa.

(melhor qualidade/custo) para cada cenário. O MOCell foi, em média, superior aos demais algoritmos nos três primeiros cenários e equivalendo estatisticamente no quarto.

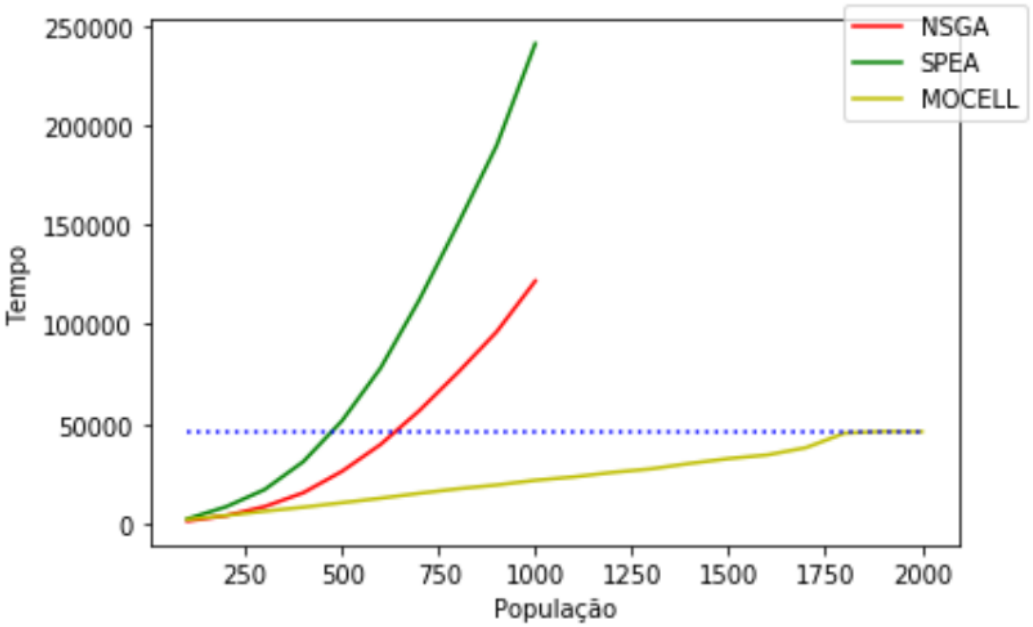
4.6 Discussão dos resultados

Conforme observado nas seções 4.1 e 4.2, o NSGAII e o SPEA2 tiveram resultados bastante próximos. Pode-se notar também que o NSGAII é menos ganancioso e, para tal, atinge um ponto de saturação com menor população que o SPEA2. Além disso, observa-se na seção 4.3 que o desempenho em esforço computacional do NSGAII é melhor do que do SPEA2. Dessa forma, entre os dois algoritmos, por meio destes resultados e neste contexto, a recomendação é de utilização do NSGAII.

Ao adicionar o MOCell na comparação, nota-se que ele possui o melhor desempenho em esforço computacional. Desse modo, executá-lo com 2.000 de população corresponde ao tempo de execução do NSGAII e do SPEA2 com 500 e 600 de população, respectivamente. Além disso, o MOCell é mais ganancioso e, assim, necessita de uma população maior para sua saturação.

Ao filtrar-se as soluções de modo a selecionar a melhor delas entre cada execução, por meio da melhor qualidade/custo, conforme observado na seção 4.5, o MOCell apresentou médias superiores nos três primeiros cenários e resultado similar aos demais algoritmos no quarto cenário. É característico do algoritmo a alta variância, tendo em vista que um de seus objetivos é representar bem a fronteira de Pareto (59). Nesse sen-

Figura 28 – Tempo de execução dos algoritmos (ms)



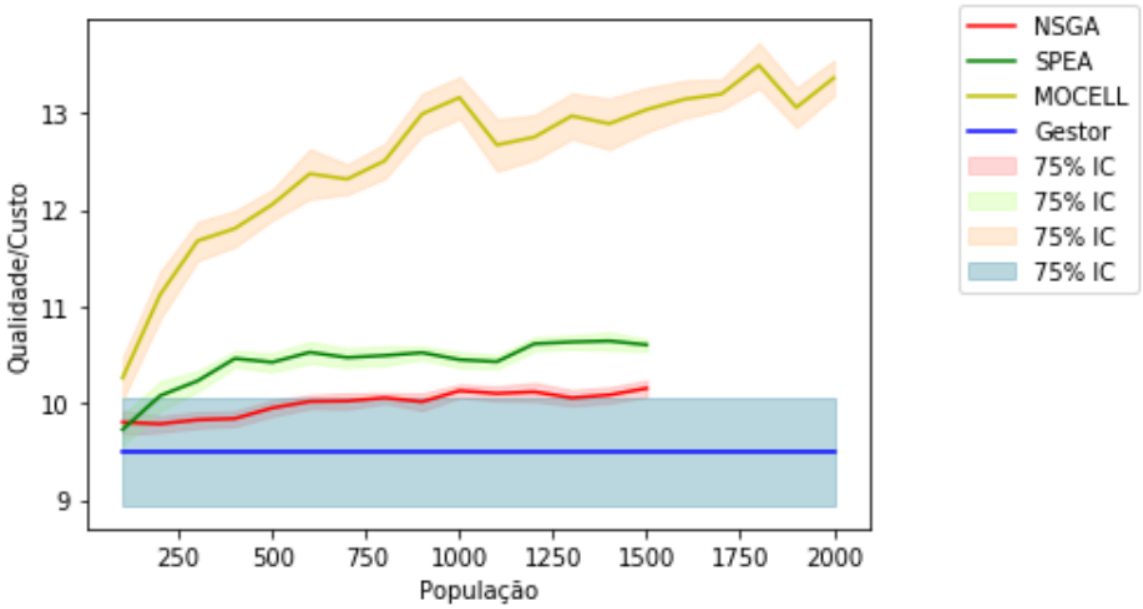
Fonte: dados da pesquisa.

tido, na prática, resultados interessantes poderiam ser apresentados se o algoritmo fosse executado algumas vezes e as melhores soluções fossem coletadas, já que em todos os cenários a melhor solução partiu deste algoritmo.

Além disso, no mesmo trabalho (59), nota-se a melhoria do MOCell utilizando uma abordagem de *feedback* da população não dominada arquivada, substituindo aleatoriamente a população da próxima iteração. Essa melhoria não foi contemplada neste trabalho, tendo em vista que na versão utilizada do JMetal ela não foi implementada no MOCell, mas sim em outras versões do algoritmo, nomeadas como sMOCell e aMOCell, que não foram testadas neste estudo. Então, ainda há a possibilidade de o algoritmo apresentar melhores resultados do que aqueles aqui observados.

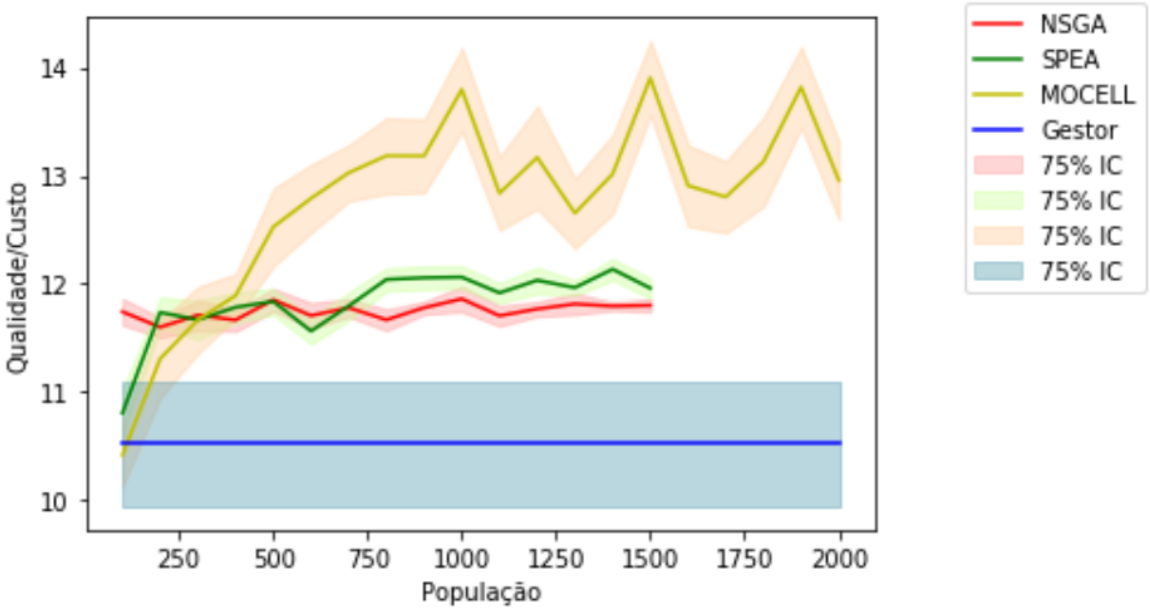
Por fim, com base nos dados verificados para este cenário, é interessante a utilização do MOCell com população de 2.000 e selecionando-se a melhor solução entre cada execução. Caso esteja inserido em um âmbito em que seja possível aguardar o resultado, pode-se ainda coletar as melhores soluções entre as várias execuções, tendo em vista que estas são melhores que a média, mas se apresentaram em número reduzido de vezes, em relação à quantidade total de execuções.

Figura 29 – Evolução do cenário I filtrando o melhor em cada execução



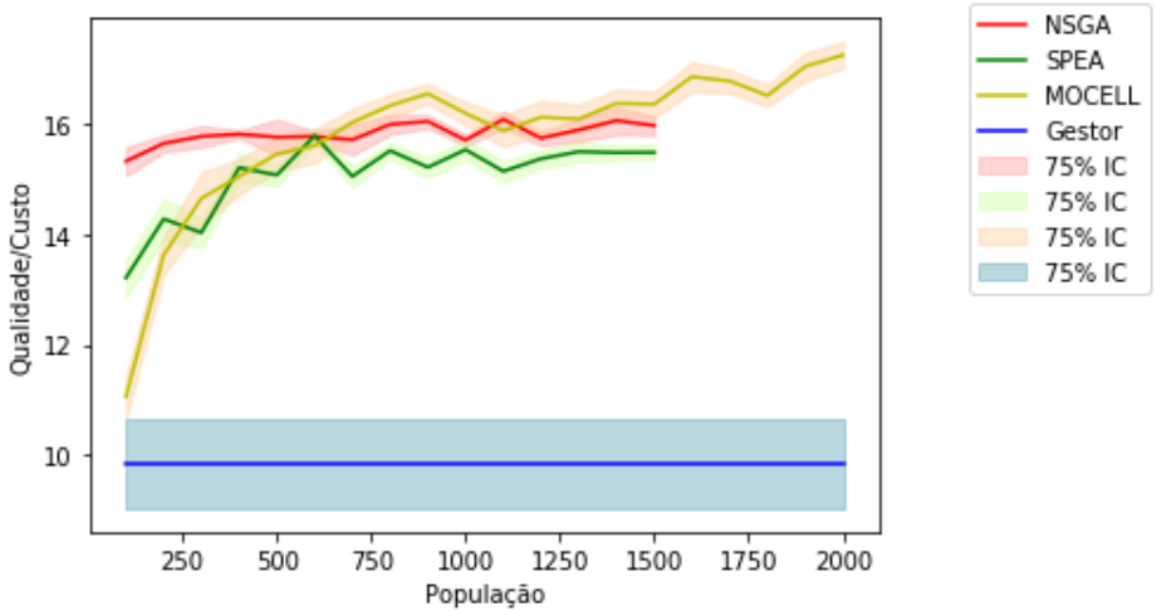
Fonte: dados da pesquisa.

Figura 30 – Evolução do cenário II filtrando o melhor em cada execução



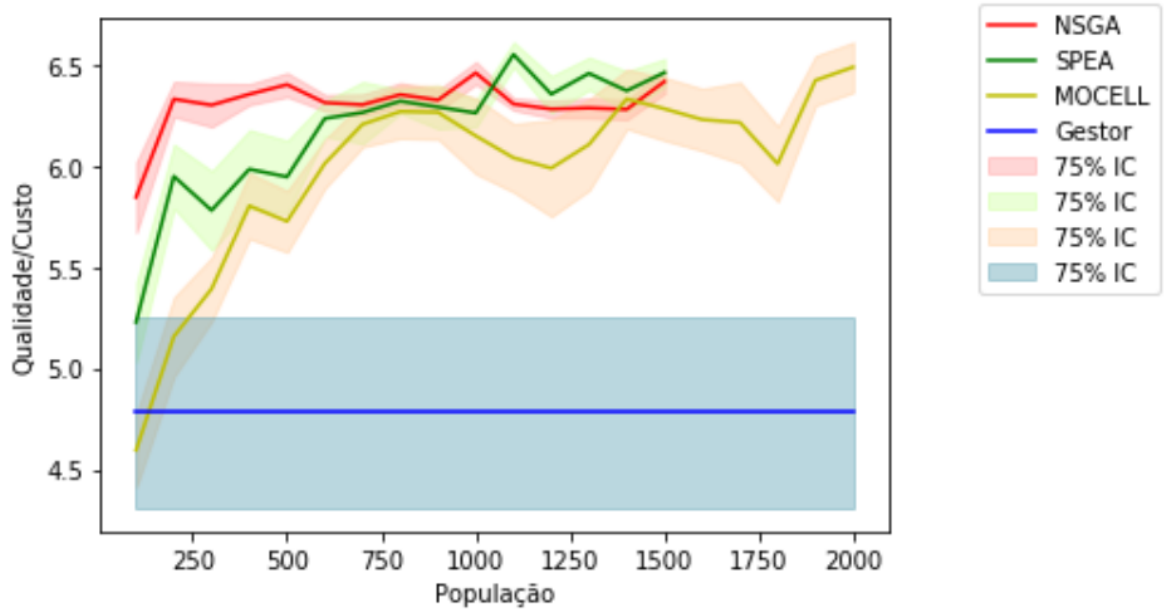
Fonte: dados da pesquisa.

Figura 31 – Evolução do cenário III filtrando o melhor em cada execução



Fonte: dados da pesquisa.

Figura 32 – Evolução do cenário IV filtrando o melhor em cada execução



Fonte: dados da pesquisa.

5 TRABALHOS RELACIONADOS

Neste capítulo são apresentados os trabalhos identificados durante o estudo e que têm relação com o problema proposto ou as técnicas utilizadas na solução do problema proposto.

Algoritmos de busca têm se tornado um meio eficiente para resolver problemas complexos de otimização em todas as fases ciclo de vida de desenvolvimento de *software* (65). Os estudos nesta área de pesquisa denominada SBSE aumentaram significativamente nas últimas duas décadas, por exemplo, o repositório SBSE hospedado pelo centro CREST (101) contém 1729 artigos publicados até 20 de janeiro de 2019. Há poucas abordagens na literatura que envolvem a problemática de alocação de recursos humanos em projetos de desenvolvimento de *software* (16).

Este estudo estende significativamente o trabalho dos autores Brito *et al.* (39) que em 2012 propuseram uma abordagem para a solução de problemas multiobjetivos com a combinação de métodos de otimização baseados em SBSE. Para resolver o problema eles utilizaram o algoritmo NSGAII multiobjetivo meta-heurística e o sistema de inferência Mandani Fuzzy. Nesta pesquisa foi realizada a análise utilizando os algoritmos NSGAII, SPEA2 e MOCell.

Sayyad *et al.* salientam os algoritmos NSGAII e SPEA2 como os algoritmos multiobjetivos mais utilizados em problemas de otimização (102). Foi realizada pesquisa bibliográfica da SBSE em artigos que utilizaram a pesquisa multiobjetivo para encontrar soluções, dando atenção aos algoritmos escolhidos, ferramentas e indicadores de qualidade, se houvesse. Concluíram que o campo da SBSE tem demonstrado tendência a adotar os algoritmos de otimização do modelo evolutivo multiobjetivo (MEOAs), que são amplamente utilizados em outros campos (como NSGA-II e SPEA2).

Nebro *et al.* propõem o algoritmo MOCell na solução de problemas de otimização multiobjetivos e comparam seu desempenho com os algoritmos NSGA-II and SPEA2 (59). Introduziram um novo algoritmo genético celular para a solução de problemas de otimização multiobjetivo. A proposta foi avaliar tanto com restrições quanto sem restrições problemas e compará-los com dois estados da arte de otimizadores evolutivos multiobjetivos, o NSGA-II e o SPEA2. O experimento realizado indicou que o MOCell obteve resultados competitivos em termos de convergência e hipervolume e superou claramente o outros dois algoritmos comparados quanto à diversidade das soluções ao longo da frente de Pareto.

Connor e Shah (103) apresentaram os resultados da aplicação de três algoritmos de busca meta-heurística na solução de problemas na área de gerenciamento de proje-

tos de *software*. Os algoritmos *Simulated annealing*, *tabu search* e algoritmos genéticos são avaliados em problemas de alocação e escalonamento de recursos (103). O objetivo desta pesquisa foi avaliar o desempenho de diferentes técnicas de busca meta-heurística em problemas de alocação e escalonamento de recursos que seriam típicos de projetos de desenvolvimento de *software*. Este trabalho relata um conjunto de experimentos que avaliam o desempenho de três algoritmos estudados e os resultados experimentais indicam que todas as técnicas de pesquisa heurística podem ser usadas para resolver problemas na alocação e no agendamento de recursos dentro de um projeto de *software*. A análise comparativa sugeriu que o algoritmo genético performou melhor que os outros dois algoritmos estudados. Este artigo, diferentemente do estudo em questão, se ocupou de além de selecionar os recursos humanos, atribuí-los às atividades do projeto, ou seja, agendar os recursos humanos no projeto.

Yi Zhang desenvolveu um modelo matemático de alocação de recursos e aplicou algoritmos genéticos para resolver o modelo (61). O artigo analisou e avaliou os elementos da organização e, em seguida, propôs um modelo de escolha de alocação ótima de recursos humanos. Foram usados algoritmos genéticos para resolver o modelo e o processo detalhado do algoritmo. O método forneceu uma nova ferramenta específica de gerenciamento quantitativo de alocação ótima de recursos humanos e um exemplo numérico foi calculado para comprovar a eficácia do método. Este trabalho utilizou um método subjetivo de pontuação para definir os requisitos dos cargos considerados no experimento e, diferente do estudo em questão, considerou uma escala de um a nove para o nível do conhecimento exigido para estes cargos. As categorias consideradas nos requisitos dos cargos foram qualidade básica, experiência, habilidades e educação. Foram avaliados no experimento apresentado no artigo 12 recursos de quatro diferentes cargos.

Feng Wen e Chi-Ming Lin (100) propõem um algoritmo genético multiobjetivo (moGA) para resolver um problema de alocação de recursos humanos em projetos de *software* ao considerar que a alocação de recursos humanos é um fator de influência mais importante entre os objetivos conflitantes entre minimizar o tempo e o custo do projeto. Esse moGA é baseado em um método de codificação chamado método de codificação de comprimento fixo aprimorado. Um mecanismo de atribuição de aptidão de peso adaptativo é usado para encontrar um conjunto de soluções de Pareto. Um método de ponderação de fator é proposto para encontrar a melhor solução comprometida de um conjunto de soluções de Pareto. Os gerentes de projeto podem atribuir peso a cada objetivo para decidir como organizar o *software* para o projeto.

Peixoto *et al.* (104) ampliaram e aprimoraram a discussão de uma RSL sobre as soluções de otimização baseadas em pesquisa do alocação de recursos e problema de agendamento. O foco do trabalho foi na análise da literatura em relação ao planejamento de projetos, mais especificamente nas pesquisas realizadas no agendamento de projetos de

software e alocação de recursos. Considerando os resultados de uma revisão sistemática da literatura, neste trabalho analisaram-se as questões de adotar esses algoritmos de otimização no que é considerado configurações típicas encontradas em organizações de desenvolvimento de *software*. Foram encontraram poucas evidências, sinalizando que as expectativas das organizações de desenvolvimento de *software* estão sendo atendidas.

Khalil *et al.* (84) adotaram abordagem meta-heurística baseada em pesquisa para a alocação de recursos humanos para o planejamento de iteração de correção de *bugs*, em processos ágeis. Levaram em conta a gravidade do *bug*, a prioridade, o nível de habilidade requerido, além do nível de habilidade do desenvolvedor, escolhas mais complicadas para um gerente que planeja tal iteração. Nesse estudo, foi otimizada a atribuição de recursos humanos para alcançar os objetivos mencionados, usando algoritmos evolutivos multiobjetivos. A lição é que complicar a formulação do problema multiobjetivo pode ajudar com a qualidade geral das soluções. O que difere esse artigo, entre outros aspectos, do presente estudo é que ele se limita à alocação de recursos humanos para atuação na manutenção do *software*, ao passo que este estudo aborda a alocação de recursos humanos independentemente de manutenção ou inovação do *software*. Outra diferença é que foi utilizado no experimento apenas o algoritmo NSGAII.

6 CONCLUSÕES E TRABALHOS FUTUROS

O objetivo desta pesquisa foi avaliar o desempenho de técnicas de otimização aplicadas na alocação de times ágeis de desenvolvimento de *software*. Foram avaliadas três técnicas de otimização (NSGAII, SPEA2 , MOCell). O primeiro objetivo específico desta investigação foi "identificar as técnicas de otimização adequadas para o problema de pesquisa proposto"que foi alcançado na revisão sistemática de literatura (capítulo 2). Para alcançar o segundo objetivo, foi realizado experimento em uma empresa de desenvolvimento de *software* de grande porte que adotou a metodologia ágil há cerca de três anos. No experimento foram testados quatro cenários reais da empresa. As técnicas testadas se mostraram aderentes ao problema proposto, conforme demonstrado no capítulo 4. Os algoritmos SPEA2 e NSGAII apresentaram resultados similares. Pode-se notar também que o NSGAII é menos ganancioso e, por isso, atinge um ponto de saturação com menor população que o SPEA2, além de exibir melhor desempenho computacional. Dessa forma, entre os dois algoritmos, por meio dos resultados apresentados no capítulo 4, recomenda-se a utilização do NSGAII. Já o MOCell, apesar de na média apresentar resultados próximos aos demais algoritmos, ele apresentou resultados expressivos, especialmente para população maiores e utilizou menos recursos computacionais. Pesquisas futuras podem investir na medição do desempenho do MOCell utilizando a abordagem de *feedback* da população não dominada arquivada, substituindo aleatoriamente a população da próxima iteração. Outra oportunidade para pesquisas futuras seria investir no estudo das configurações dos parâmetros dos algoritmos avaliando a abordagem de automação do conjunto de parâmetros para serem modificados durante sua execução e análise dos resultados alcançados.

Referências

1 NAN, N.; HARTER, D. E. Impact of budget and schedule pressure on software development cycle time and effort. *IEEE Transactions on Software Engineering*, IEEE, v. 35, n. 5, p. 624–637, 2009. Citado na página 13.

2 FAGERHOLM, F. et al. Performance alignment work: How software developers experience the continuous adaptation of team performance in lean and agile environments. *Information and Software Technology*, Elsevier, v. 64, p. 132–147, 2015. Citado 3 vezes nas páginas 13, 21 e 22.

3 ADENSO-DÍAZ, B.; GONZALEZ-TORRE, P.; GARCIA, V. A capacity management model in service industries. *International Journal of Service Industry Management*, MCB UP Ltd, v. 13, n. 3, p. 286–302, 2002. Citado na página 13.

4 CHAVES-GONZÁLEZ, J. M.; PÉREZ-TOLEDANO, M. A. Differential evolution with pareto tournament for the multi-objective next release problem. *Applied Mathematics and Computation*, Elsevier, v. 252, p. 1–13, 2015. Citado 2 vezes nas páginas 13 e 53.

5 KORKALA, M.; PIKKARAINEN, M.; CONBOY, K. A case study of customer communication in globally distributed software product development. In: ACM. *Proceedings of the 11th International Conference on Product Focused Software*. [S.l.], 2010. p. 43–46. Citado na página 13.

6 BARRETO, A.; BARROS, M. de O.; WERNER, C. M. L. Staffing a software project: a constraint satisfaction and optimization-based approach. *Computers & Operations Research*, v. 35, n. 10, p. 3073–3089, October 2008. Citado na página 13.

7 CORAM, M.; BOHNER, S. The impact of agile methods on software project management. In: IEEE. *Engineering of Computer-Based Systems, 2005. ECBS'05. 12th IEEE International Conference and Workshops on the*. [S.l.], 2005. p. 363–370. Citado na página 13.

8 HODA, R.; NOBLE, J.; MARSHALL, S. Organizing self-organizing teams. In: IEEE. *Software Engineering, 2010 ACM/IEEE 32nd International Conference on*. [S.l.], 2010. v. 1, p. 285–294. Citado na página 13.

9 STYLIANOU, C.; GERASIMOU, S.; ANDREOU, A. S. A novel prototype tool for intelligent software project scheduling and staffing enhanced with personality factors. In: IEEE. *Tools with Artificial Intelligence (ICTAI), 2012 IEEE 24th International Conference on*. [S.l.], 2012. v. 1, p. 277–284. Citado na página 13.

10 HARMAN, M.; MANSOURI, S. A.; ZHANG, Y. Search-based software engineering: Trends, techniques and applications. *ACM Computing Surveys (CSUR)*, ACM, v. 45, n. 1, p. 11, 2012. Citado 3 vezes nas páginas 13, 35 e 36.

11 HARMAN, M. et al. Dynamic adaptive search based software engineering. In: ACM. *Proceedings of the ACM-IEEE international symposium on Empirical software engineering and measurement*. [S.l.], 2012. p. 1–8. Citado 2 vezes nas páginas 13 e 36.

12 GAY, G. A baseline method for search-based software engineering. In: ACM. *Proceedings of the 6th International Conference on Predictive Models in Software Engineering*. [S.l.], 2010. p. 2. Citado 5 vezes nas páginas 13, 14, 35, 36 e 53.

13 KALYANMOY, D. et al. *Multi objective optimization using evolutionary algorithms*. [S.l.]: John Wiley and Sons, 2001. Citado na página 14.

14 AHMED, A. et al. Agile software development: Impact on productivity and quality. In: IEEE. *Management of innovation and technology (ICMIT), 2010 IEEE international conference on*. [S.l.], 2010. p. 287–291. Citado 2 vezes nas páginas 14 e 18.

15 CORAM, M.; BOHNER, S. The impact of agile methods on software project management. In: IEEE. *Engineering of Computer-Based Systems, 2005. ECBS'05. 12th IEEE International Conference and Workshops on the*. [S.l.], 2005. p. 363–370. Citado 7 vezes nas páginas 14, 19, 20, 21, 26, 33 e 34.

16 KAPUR, P. et al. Optimized staffing for product releases and its application at chartwell technology. *Journal of Software Maintenance and Evolution: Research and Practice*, Wiley Online Library, v. 20, n. 5, p. 365–386, 2008. Citado 4 vezes nas páginas 14, 32, 33 e 87.

17 INGOLD, D.; BOEHM, B.; KOOLMANOJWONG, S. A model for estimating agile project process and schedule acceleration. In: ACM. *Proceedings of the 2013 International Conference on Software and System Process*. [S.l.], 2013. p. 29–35. Citado 2 vezes nas páginas 15 e 18.

18 STAMELOS, I. G. *Agile software development quality assurance*. [S.l.]: Igi Global, 2007. Citado 3 vezes nas páginas 16, 19 e 34.

19 KITCHENHAM, B. Procedures for performing systematic reviews. *Keele, UK, Keele University*, v. 33, n. 2004, p. 1–26, 2004. Citado 2 vezes nas páginas 18 e 48.

20 COHN, M. *Agile estimating and planning*. [S.l.]: Pearson Education, 2005. Citado 3 vezes nas páginas 18, 32 e 33.

21 HUO, M. et al. Software quality and agile methods. In: IEEE. *Computer Software and Applications Conference, 2004. COMPSAC 2004. Proceedings of the 28th Annual International*. [S.l.], 2004. p. 520–525. Citado na página 18.

22 LAYTON, M. C.; OSTERMILLER, S. J. *Agile project management for dummies*. [S.l.]: John Wiley & Sons, 2017. Citado na página 19.

23 ABRAHAMSSON, P. et al. Agile software development methods: Review and analysis. *arXiv preprint arXiv:1709.08439*, 2017. Citado 2 vezes nas páginas 19 e 30.

24 BARRETO, A.; BARROS, M. d. O.; WERNER, C. M. Staffing a software project: A constraint satisfaction and optimization-based approach. *Computers & Operations Research*, Elsevier, v. 35, n. 10, p. 3073–3089, 2008. Citado na página 21.

25 MELO, C. D. O. et al. Interpretative case studies on agile team productivity and management. *Information and Software Technology*, Elsevier, v. 55, n. 2, p. 412–427, 2013. Citado 2 vezes nas páginas 22 e 23.

26 MELO, C. et al. Agile team perceptions of productivity factors. In: IEEE. *2011 Agile Conference*. [S.l.], 2011. p. 57–66. Citado na página 23.

27 LICORISH, S.; PHILPOTT, A.; MACDONELL, S. G. Supporting agile team composition: A prototype tool for identifying personality (in) compatibilities. In: IEEE COMPUTER SOCIETY. *Proceedings of the 2009 ICSE Workshop on Cooperative and Human Aspects on Software Engineering*. [S.l.], 2009. p. 66–73. Citado 2 vezes nas páginas 23 e 32.

28 CHANG, N. The application of neural network to the allocation of enterprise human resources. In: IEEE. *e-Business and Information System Security (EBISS), 2010 2nd International Conference on*. [S.l.], 2010. p. 1–4. Citado 3 vezes nas páginas 23, 26 e 27.

29 ATHEY, T. R.; ORTH, M. S. Emerging competency methods for the future. *Human Resource Management: Published in Cooperation with the School of Business Administration, The University of Michigan and in alliance with the Society of Human Resources Management*, Wiley Online Library, v. 38, n. 3, p. 215–225, 1999. Citado na página 24.

30 GANGANI, N.; MCLEAN, G. N.; BRADEN, R. A. A competency-based human resource development strategy. *Performance Improvement Quarterly*, Wiley Online Library, v. 19, n. 1, p. 127–139, 2006. Citado na página 24.

31 BOYATZIS, R. E. *The competent manager: A model for effective performance*. [S.l.]: John Wiley & Sons, 1982. Citado na página 24.

32 BUFORD, J. A.; LINDNER, J. R. *Human resource management in local government: Concepts and applications for HRM students and practitioners*. [S.l.]: South-Western Pub, 2002. Citado na página 24.

33 PETERSON, N. G. et al. Development of prototype occupational information network (o* net) content model. volume i: Report [and] volume ii: Appendices. ERIC, 1995. Citado na página 24.

34 FITZPATRICK, R. Building robust competencies: Linking human resource systems to organizational strategies. *Personnel Psychology*, Blackwell Publishing Ltd., v. 53, n. 1, p. 248, 2000. Citado na página 24.

35 GUION, R. M. Personnel assessment, selection, and placement. Consulting Psychologists Press, 1991. Citado na página 24.

36 MIRABILE, R. Implementation planning: Key to successful competency strategies. *HUMAN RESOURCES PROFESSIONAL-NEW YORK-*, LRP PUBLICATIONS, v. 10, p. 19–23, 1997. Citado na página 24.

37 SPENCER, L. M.; SPENCER, P. S. M. *Competence at Work models for superior performance*. [S.l.]: John Wiley & Sons, 2008. Citado na página 24.

38 SANCHEZ, J. I. The art and science of competency models: Pinpointing critical success factors. *Personnel Psychology*, Blackwell Publishing Ltd., v. 53, n. 2, p. 509, 2000. Citado na página 24.

39 BRITTO, R. et al. A hybrid approach to solve the agile team allocation problem. In: *Proceedings of IEEE Congress on Evolutionary Computation (CEC '12)*. Brisbane, Australia: IEEE, 2012. p. 1–8. Citado 5 vezes nas páginas 25, 32, 33, 53 e 87.

40 BAKER, J. et al. A hierarchical model of business competence. *Integrated Manufacturing Systems*, MCB UP Ltd, v. 8, n. 5, p. 265–272, 1997. Citado na página 25.

41 SINCLAIR, J. et al. *Collins COBUILD English language dictionary*. [S.l.]: Harper Collins Publishers,, 1987. Citado na página 25.

42 SANCHEZ, R.; HEENE, A.; THOMAS, H. Towards the theory and practice of competence-based competition, forthcoming. 1996. Citado na página 25.

43 HAFEEZ, K.; ABDELMEGUID, H. Dynamics of human resource and knowledge management. *Journal of the Operational Research Society*, Taylor & Francis, v. 54, n. 2, p. 153–164, 2003. Citado 2 vezes nas páginas 25 e 26.

44 CHAU, T.; MAURER, F.; MELNIK, G. Knowledge sharing: Agile methods vs. tayloristic methods. In: IEEE. *Enabling Technologies: Infrastructure for Collaborative Enterprises, 2003. WET ICE 2003. Proceedings. Twelfth IEEE International Workshops on*. [S.l.], 2003. p. 302–307. Citado 2 vezes nas páginas 25 e 26.

45 ANWER, F. et al. Agile software development models tdd, fdd, dsdm, and crystal methods: A survey. *International Journal of Multidisciplinary Sciences and Engineering*, v. 8, n. 2, p. 1–10, 2017. Citado na página 27.

46 MATHARU, G. S. et al. Empirical study of agile software development methodologies: A comparative analysis. *ACM SIGSOFT Software Engineering Notes*, ACM, v. 40, n. 1, p. 1–6, 2015. Citado 3 vezes nas páginas 27, 29 e 30.

47 TULI, A. et al. Empirical investigation of agile software development: cloud perspective. *ACM SIGSOFT Software Engineering Notes*, ACM, v. 39, n. 4, p. 1–6, 2014. Citado 5 vezes nas páginas 27, 28, 29, 30 e 31.

48 SCHWABER, K.; BEEDLE, M. *Agile software development with Scrum*. [S.l.]: Prentice Hall Upper Saddle River, 2002. v. 1. Citado na página 30.

49 RICO, D. F. Lean and agile project management: for large programs and projects. In: *Lean Enterprise Software and Systems*. [S.l.]: Springer, 2010. p. 37–43. Citado na página 31.

50 DYBÅ, T.; DINGSØYR, T. *May. Agile project management: from self-managing teams to large-scale development. In 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering (Vol. 2, pp. 945-946)*. [S.l.]: IEEE, 2015. Citado 2 vezes nas páginas 31 e 32.

51 PENTA, M. D.; HARMAN, M.; ANTONIOL, G. The use of search-based optimization techniques to schedule and staff software projects: an approach and an empirical study. *Software: Practice and Experience*, Wiley Online Library, v. 41, n. 5, p. 495–519, 2011. Citado na página 32.

52 MNKANDLA, E.; DWOLATZKY, B. Defining agile software quality assurance. In: IEEE. *Software Engineering Advances, International Conference on*. [S.l.], 2006. p. 36–36. Citado na página 34.

53 WANG, S. et al. A practical guide to select quality indicators for assessing pareto-based search algorithms in search-based software engineering. In: IEEE. *Software Engineering (ICSE), 2016 IEEE/ACM 38th International Conference on*. [S.l.], 2016. p. 631–642. Citado 2 vezes nas páginas 35 e 36.

54 CHEN, J. et al. " sampling" as a baseline optimizer for search-based software engineering. *IEEE Transactions on Software Engineering*, IEEE, 2018. Citado 3 vezes nas páginas 36, 40 e 53.

55 FERRUCCI, F.; HARMAN, M.; SARRO, F. Search-based software project management, in software project management in a changing world. In: _____. [S.l.]: Springer, 2014. p. 373–399. Citado 2 vezes nas páginas 36 e 37.

56 HARMAN, M. et al. Search based software engineering: Techniques, taxonomy, tutorial. In: *Empirical software engineering and verification*. [S.l.]: Springer, 2012. p. 1–59. Citado na página 37.

57 HARMAN, M.; MANSOURI, S. A.; ZHANG, Y. Search based software engineering: A comprehensive analysis and review of trends techniques and applications. *Department of Computer Science, King's College London, Tech. Rep. TR-09-03*, p. 23, 2009. Citado na página 38.

58 YANG, X.-S. *Engineering optimization: an introduction with metaheuristic applications*. [S.l.]: John Wiley & Sons, 2010. Citado 2 vezes nas páginas 38 e 39.

59 NEBRO JUAN J. DURILLO, F. L. B. D. E. A. A. J. Mocell: A cellular genetic algorithm for multiobjective optimization. *INTERNATIONAL JOURNAL OF INTELLIGENT SYSTEMS*, Wiley, v. 24, p. 726–746, 2009. Citado 10 vezes nas páginas 39, 42, 45, 46, 47, 54, 82, 83, 84 e 87.

60 KONAK, A.; COIT, D. W.; SMITH, A. E. Multi-objective optimization using genetic algorithms: A tutorial. *Reliability Engineering & System Safety*, Elsevier, v. 91, n. 9, p. 992–1007, 2006. Citado na página 39.

61 ZHANG, Y. Research on human resource allocation optimization based on genetic algorithm from the perspective of two-way choice model. In: IEEE. *Educational and Information Technology (ICEIT), 2010 International Conference on*. [S.l.], 2010. v. 1, p. V1–380. Citado 2 vezes nas páginas 40 e 88.

62 EIBEN, Á. E.; HINTERDING, R.; MICHALEWICZ, Z. Parameter control in evolutionary algorithms. *IEEE Transactions on evolutionary computation*, IEEE, v. 3, n. 2, p. 124–141, 1999. Citado 2 vezes nas páginas 40 e 41.

63 HESSER, J.; MÄNNER, R. Towards an optimal mutation probability for genetic algorithms. In: SPRINGER. *International Conference on Parallel Problem Solving from Nature*. [S.l.], 1990. p. 23–32. Citado na página 41.

64 GREFENSTETTE, J. J. Optimization of control parameters for genetic algorithms. *IEEE Transactions on systems, man, and cybernetics*, IEEE, v. 16, n. 1, p. 122–128, 1986. Citado 2 vezes nas páginas 41 e 42.

65 WANG, S. et al. A practical guide to select quality indicators for assessing pareto-based search algorithms in search-based software engineering. In: IEEE. *Software Engineering (ICSE), 2016 IEEE/ACM 38th International Conference on*. [S.l.], 2016. p. 631–642. Citado 4 vezes nas páginas 42, 43, 46 e 87.

66 DEB, K. et al. A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: Nsga-ii. In: SPRINGER. *International Conference on Parallel Problem Solving From Nature*. [S.l.], 2000. p. 849–858. Citado na página 42.

67 KALYANMOY, D. Multi-objective optimization using evolutionary algorithms: An introduction. *KanGAL Report*, n. 2011003, 2011. Citado na página 42.

68 YUSOFF, Y.; NGADIMAN, M. S.; ZAIN, A. M. Overview of nsga-ii for optimizing machining process parameters. *Procedia Engineering*, Elsevier, v. 15, p. 3978–3983, 2011. Citado na página 43.

69 SANTOS, P. d. A. dos et al. A hybrid approach to suggest software product line portfolios. *Applied Soft Computing*, Elsevier, v. 49, p. 1243–1255, 2016. Citado 3 vezes nas páginas 43, 44 e 53.

70 ZITZLER, E.; DEB, K.; THIELE, L. *Comparison of multiobjective evolutionary algorithms: Empirical results (revised version)*. [S.l.], 1999. Citado na página 44.

71 ZITZLER, E.; LAUMANNS, M.; THIELE, L. Spea2: Improving the strength pareto evolutionary algorithm. *TIK-report*, Eidgenössische Technische Hochschule Zürich (ETH), Institut für Technische ..., v. 103, 2001. Citado 2 vezes nas páginas 44 e 45.

72 ZITZLER, E.; DEB, K.; THIELE, L. Comparison of multiobjective evolutionary algorithms: Empirical results. *Evolutionary computation*, MIT Press, v. 8, n. 2, p. 173–195, 2000. Citado na página 44.

73 SHENG, W. et al. An improved strength pareto evolutionary algorithm 2 with application to the optimization of distributed generations. *Computers & Mathematics with Applications*, Elsevier, v. 64, n. 5, p. 944–955, 2012. Citado na página 44.

74 BRERETON, P. et al. Lessons from applying the systematic literature review process within the software engineering domain. *Journal of systems and software*, Elsevier, v. 80, n. 4, p. 571–583, 2007. Citado na página 48.

75 SAYYAD, A. S.; AMMAR, H. Pareto-optimal search-based software engineering (posbse): A literature survey. In: IEEE. *Realizing Artificial Intelligence Synergies in Software Engineering (RAISE), 2013 2nd International Workshop on*. [S.l.], 2013. p. 21–27. Citado na página 51.

76 GUIZZO, G. et al. A hyper-heuristic for the multi-objective integration and test order problem. In: ACM. *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*. [S.l.], 2015. p. 1343–1350. Citado na página 53.

77 COLANZI, T. E. et al. A search-based approach for software product line design. In: ACM. *Proceedings of the 18th International Software Product Line Conference-Volume 1*. [S.l.], 2014. p. 237–241. Citado na página 53.

78 SIMONS, C. L.; PARMEE, I. C. An empirical investigation of search-based computational support for conceptual software engineering design. In: IEEE. *Systems, Man and Cybernetics, 2009. SMC 2009. IEEE International Conference on*. [S.l.], 2009. p. 2503–2508. Citado na página 53.

79 HASHEMINEJAD, S. M. H.; JALILI, S. An evolutionary approach to identify logical components. *Journal of Systems and Software*, Elsevier, v. 96, p. 24–50, 2014. Citado na página 53.

80 LIMA, J. A. P.; VERGILIO, S. R. et al. Automatic generation of search-based algorithms applied to the feature testing of software product lines. In: ACM. *Proceedings of the 31st Brazilian Symposium on Software Engineering*. [S.l.], 2017. p. 114–123. Citado na página 53.

81 CHEN, J.; NAIR, V.; MENZIES, T. Beyond evolutionary algorithms for search-based software engineering. *Information and Software Technology*, Elsevier, v. 95, p. 281–294, 2018. Citado na página 53.

82 MAGDALENO, A. M. et al. Collaboration optimization in software process composition. *Journal of Systems and Software*, Elsevier, v. 103, p. 452–466, 2015. Citado na página 53.

83 HARMAN, M. et al. Exact scalable sensitivity analysis for the next release problem. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, ACM, v. 23, n. 2, p. 19, 2014. Citado na página 53.

84 KHALIL, E.; ASSAF, M.; SAYYAD, A. S. Human resource optimization for bug fixing: balancing short-term and long-term objectives. In: SPRINGER. *International Symposium on Search Based Software Engineering*. [S.l.], 2017. p. 124–129. Citado 3 vezes nas páginas 53, 70 e 89.

85 FERREIRA, T. do N. et al. Incorporating user preferences in ant colony optimization for the next release problem. *Applied Soft Computing*, Elsevier, v. 49, p. 1283–1296, 2016. Citado na página 53.

86 SAGRADO, J. del; AGUILA, I. M. del; ORELLANA, F. J. Multi-objective ant colony optimization for requirements selection. *Empirical Software Engineering*, Springer, v. 20, n. 3, p. 577–610, 2015. Citado 2 vezes nas páginas 53 e 70.

87 GHANNEM, A.; BOUSSAIDI, G. E.; KESSENTINI, M. Model refactoring using examples: a search-based approach. *Journal of Software: Evolution and Process*, Wiley Online Library, v. 26, n. 7, p. 692–713, 2014. Citado na página 53.

88 MANSOOR, U. et al. Multi-objective code-smells detection using good and bad design examples. *Software Quality Journal*, Springer, v. 25, n. 2, p. 529–552, 2017. Citado na página 53.

89 AL-ZUBAIDI, W. H. A. et al. Multi-objective search-based approach to estimate issue resolution time. In: ACM. *Proceedings of the 13th International Conference on Predictive Models and Data Analytics in Software Engineering*. [S.l.], 2017. p. 53–62. Citado na página 53.

90 FERRUCCI, F. et al. Not going to take this anymore: multi-objective overtime planning for software engineering projects. In: IEEE PRESS. *Proceedings of the 2013 International Conference on Software Engineering*. [S.l.], 2013. p. 462–471. Citado na página 53.

91 RAMÍREZ, A.; ROMERO, J. R.; VENTURA, S. On the performance of multiple objective evolutionary algorithms for software architecture discovery. In: ACM. *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation*. [S.l.], 2014. p. 1287–1294. Citado na página 53.

92 FÉDERLE, É. L. et al. Opla-tool: a support tool for search-based product line architecture design. In: ACM. *Proceedings of the 19th International Conference on Software Product Line*. [S.l.], 2015. p. 370–373. Citado na página 53.

93 O'KEEFFE, M.; CINNÉIDE, M. Ó. Search-based refactoring: an empirical study. *Journal of Software Maintenance and Evolution: Research and Practice*, Wiley Online Library, v. 20, n. 5, p. 345–364, 2008. Citado na página 53.

94 CHAVES-GONZÁLEZ, J. M.; PEREZ-TOLEDANO, M. A.; NAVASA, A. Software requirement optimization using a multiobjective swarm intelligence evolutionary algorithm. *Knowledge-Based Systems*, Elsevier, v. 83, p. 105–115, 2015. Citado na página 53.

95 CHAVES-GONZÁLEZ, J. M.; PÉREZ-TOLEDANO, M. A.; NAVASA, A. Teaching learning based optimization with pareto tournament for the multiobjective software requirements selection. *Engineering Applications of Artificial Intelligence*, Elsevier, v. 43, p. 89–101, 2015. Citado na página 53.

96 HARMAN, M. et al. The gismoe challenge: Constructing the pareto program surface using genetic programming to find better programs (keynote paper). In: ACM. *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering*. [S.l.], 2012. p. 1–14. Citado na página 53.

97 ZHANG, Y. et al. Today/future importance analysis. In: ACM. *Proceedings of the 12th annual conference on Genetic and evolutionary computation*. [S.l.], 2010. p. 1357–1364. Citado na página 53.

98 CERVANTES, J.; STEPHENS, C. R. Optimal mutation rates for genetic search. In: ACM. *Proceedings of the 8th annual conference on Genetic and evolutionary computation*. [S.l.], 2006. p. 1313–1320. Citado na página 70.

99 OCHOA, G. Setting the mutation rate: Scope and limitations of the 1/l heuristic. In: MORGAN KAUFMANN PUBLISHERS INC. *Proceedings of the 4th Annual Conference on Genetic and Evolutionary Computation*. [S.l.], 2002. p. 495–502. Citado na página 70.

100 WEN, F.; LIN, C.-M. Multistage human resource allocation for software development by multiobjective genetic algorithm. *The Open Applied Mathematics Journal*, v. 2, p. 95–103, 2008. Disponível em: <http://www.bentham.org/open/toamj/articles/V002/95TOAMJ.pdf>. Citado 2 vezes nas páginas 70 e 88.

101 ZHANG, Y.; HARMAN, M.; MANSOURI, A. *The SBSE Repository: A repository and analysis of authors and research articles on Search Based Software Engineering.* [S.l.]: CREST Centre, UCL, 2017. <crestweb.cs.ucl.ac.uk/resources/sbse_repository/>. Citado na página 87.

102 SAYYAD, A. S.; AMMAR, H. Pareto-optimal search-based software engineering (posbse): A literature survey. In: IEEE. *Realizing Artificial Intelligence Synergies in Software Engineering (RAISE), 2013 2nd International Workshop on.* [S.l.], 2013. p. 21–27. Citado na página 87.

103 CONNOR, A. M.; SHAH, A. Resource allocation using metaheuristic search. In: *Proceedings of the 4th International Conference on Computer Science and Information Technology (CCSIT '14).* Sydney, Australia: [s.n.], 2014. Citado 2 vezes nas páginas 87 e 88.

104 PEIXOTO, D. C.; MATEUS, G. R.; RESENDE, R. F. The issues of solving staffing and scheduling problems in software development projects. In: IEEE. *Computer Software and Applications Conference (COMPSAC), 2014 IEEE 38th Annual.* [S.l.], 2014. p. 1–10. Citado na página 88.