Unificando conceitos de avaliação de desempenho,
engenharia de desempenho e teste de software para
a análise de sistemas computacionais

Rafael de Souza Stabile

SERVIÇO DE PÓS-GRADUAÇÃO DO ICMC-USP

Data de Depósito:

Assinatura: _____

Rafael de Souza Stabile

Unificando conceitos de avaliação de desempenho,
engenharia de desempenho e teste de software para a
análise de sistemas computacionais

Dissertação apresentada ao Instituto de Ciências
Matemáticas e de Computação – ICMC-USP,
como parte dos requisitos para obtenção do título
de Mestre em Ciências – Ciências de Computação e
Matemática Computacional. *VERSÃO REVISADA*

Área de Concentração: Ciências de Computação e
Matemática Computacional

Orientador: Prof. Dr. Marcos José Santana

USP – São Carlos
Agosto de 2016

Ficha catalográfica elaborada pela Biblioteca Prof. Achille Bassi
e Seção Técnica de Informática, ICMC/USP,
com os dados fornecidos pelo(a) autor(a)

S634u   Stabile, Rafael de Souza
Unificando conceitos de avaliação de desempenho,
engenharia de desempenho e teste de software para
a análise de sistemas computacionais / Rafael de
Souza Stabile; orientador Marcos José Santana. - São
Carlos - SP, 2016.
77 p.

Dissertação (Mestrado - Programa de Pós-Graduação
em Ciências de Computação e Matemática Computacional)
- Instituto de Ciências Matemáticas e de Computação,
Universidade de São Paulo, 2016.

1. Relacionamento entre áreas. 2. Ontologia.
3. Análise de Desempenho. I. Santana, Marcos José,
orient. II. Título.

Rafael de Souza Stabile

Unifying concepts of performance evaluation, performance
engineering and performance testing for computer systems
analysis

Master dissertation submitted to the Instituto de
Ciências Matemáticas e de Computação – ICMC-
USP, in partial fulfillment of the requirements for the
degree of the Master Program in Computer Science
and Computational Mathematics. *FINAL VERSION*

Concentration Area: Computer Science and
Computational Mathematics

Advisor: Prof. Dr. Marcos José Santana

USP – São Carlos
August 2016

*Este trabalho é dedicado à todos que conviveram comigo e
acreditaram em mim durante o mestrado,
e em especial à minha família.*

AGRADECIMENTOS

Agradeço primeiramente a Deus, pois sem Ele nada seria possível.

Agradeço também à minha família, à minha mãe Sandra, ao meu pai Luiz e ao meu irmão Gabriel, pela paciência e apoio dados durante todo o mestrado, mas principalmente pelos dias de minha ausência longe de casa.

Agradeço também aos meus orientadores, os professores Sarita Mazzini Bruschi, pela sabedoria passada e preocupação a cada reunião realizada e Marcos Santana, por tratar de questões burocráticas. Agradeço também ao prof. Paulo Sergio, pelo apoio, ajuda e conselhos dados durante todo o mestrado.

Agradeço também aos demais professores e funcionários do Laboratório de Sistemas Distribuídos e Programação Concorrente e do ICMC-USP de um modo geral, por todo apoio necessário dado, principalmente aos professores Simone e Júlio.

Agradeço aos amigos feitos durante o mestrado, que foram como uma família longe da minha casa, em um período de grande amadurecimento, mudança e crescimento pessoal. Nesses estão inclusos os amigos feitos no LaSDPC, aos quais convivi diariamente, os amigos feitos no programa de pós, os amigos que morei junto e aqueles de outros cursos da USP-São Carlos. E principalmente aqueles que de alguma forma me ajudaram em qualquer momento, nem com uma conversa e que compartilharam momentos comigo durante esse período. À todos eles, eu não seria capaz de nomear, pois esqueceria de alguém, mas ficam os meus mais profundos agradecimentos.

Por fim, agradeço à Capes e ao CNPQ pela ajuda financeira dada.

“Tenho em mim todos os sonhos do mundo.”
(Fernando Pessoa)

RESUMO

STABILE, R. S.. **Unificando conceitos de avaliação de desempenho, engenharia de desempenho e teste de software para a análise de sistemas computacionais**. 2016. 77 f. Dissertação (Mestrado em Ciências – Ciências de Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e de Computação (ICMC/USP), São Carlos – SP.

Este projeto de mestrado apresenta uma visão de como áreas da Ciência da Computação relacionadas a desempenho (Avaliação de Desempenho, Engenharia de Desempenho de Software e Teste de Desempenho de Software) podem ser relacionadas e também propõe uma metodologia de análise de desempenho que contém conceitos das áreas anteriormente apontadas, para que esta seja mais completa e possa ser compreendida por profissionais das três áreas. Para formalizar esse relacionamento, foi construída uma ontologia que mostra como ocorre a correlação entre as áreas, e a partir dessa metodologia proposta, foi possível analisar o desempenho da ferramenta ValiPar, em sua versão paralela e concluir que o principal gargalo com relação a escalabilidade dela é sua porção executada sequencialmente. Por fim, foi observado que a metodologia apresenta vantagens se comparada as demais, como uma formalização nas suas etapas de análise.

Palavras-chave: Relacionamento entre áreas, Ontologia, Análise de Desempenho.

ABSTRACT

STABILE, R. S.. **Unificando conceitos de avaliação de desempenho, engenharia de desempenho e teste de software para a análise de sistemas computacionais**. 2016. 77 f. Dissertação (Mestrado em Ciências – Ciências de Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e de Computação (ICMC/USP), São Carlos – SP.

This master project presents a view of how Computers Science areas related to performance (Performance Evaluation, Software Performance Engineering and Software Performance Testing) can be related and also proposes a performance analysis methodology that contains concepts from all areas previously identified, so that it is more complete and can be understood by professionals of these three areas. To formalize this relationship, an ontology that shows how the correlation occurs between areas was built. And from the proposed methodology, it was possible to analyze the performance of ValiPar tool, in its parallel version, and it was concluded that the main bottleneck with respect to its scalability is its portion executed sequentially. Finally, it was observed that the proposed methodology has advantages compared to others, as a formalization in its analysis steps.

Key-words: Relating between areas, Ontology, Performance Analysis.

LISTA DE ILUSTRAÇÕES

Figura 1 – Arquitetura da ValiPar (SOUZA; SOUZA; ZALUSKA, 2014)	9
Figura 2 – Arquitetura ValiPar (BATISTA, 2015)	10
Figura 3 – Ontologia de Teste de Desempenho (SILVA, 2013)	12
Figura 4 – Fases da EDS (SMITH, 1986)	22
Figura 5 – Etapas da EDS, adaptado de Bondi (2014)	23
Figura 6 – Mapa Conceitual para Avaliação de Desempenho	27
Figura 7 – Mapa Conceitual para Engenharia de Desempenho de Software	28
Figura 8 – Mapa Conceitual para Teste de Desempenho de Software	29
Figura 9 – Taxonomia relacionando as áreas de Avaliação de Desempenho, Engenharia de Desempenho de Software e Teste de Desempenho de Software.	36
Figura 10 – Mapa Conceitual unificando os conceitos das três áreas	39
Figura 11 – Diagrama do fluxo de execução da metodologia	45
Figura 12 – Diagrama da execução dos módulos da ValiPar (BATISTA, 2015)	50
Figura 13 – Diagrama da execução simplificada da Darch	52
Figura 14 – Tempos da Execução de Variantes para os benchmarks (a) Matrix Ambos paradigmas (b) RollerCoaster (c) Hilzers Barbershop (d) Matrix Memória Compartilhada	53
Figura 15 – Tempos de Sessão de Teste para os benchmarks (a) Matrix Ambos paradigmas (b) RollerCoaster (c) Hilzers Barbershop (d) Matrix Memória Compartilhada	54
Figura 16 – Speedup para (a) Tempo de Execução de Variantes e (b) Tempo de Sessão de Teste	55
Figura 17 – Eficiência para (a) Tempo de Execução de Variantes e (b) Tempo de Sessão de Teste	56
Figura 18 – Tamanho do arquivo de rastro por execução em MB	57
Figura 19 – Tamanho do arquivo de rastro total em MB	57

LISTA DE ALGORITMOS

Algoritmo 1 – Execução automática dos experimentos	51

LISTA DE TABELAS

Tabela 1 – Tabela com Resumo dos Trabalhos Relacionados	14
Tabela 2 – Exemplo de uma entrada na lista de conceitos	32
Tabela 3 – Tabela com os conceitos que têm a mesma definição	33
Tabela 4 – Especialistas do projeto	35
Tabela 5 – Métricas das Ontologias	38
Tabela 6 – Tabela com Fatores e Níveis	48
Tabela 7 – Configurações dos Benchmarks	48
Tabela 8 – Configurações das máquinas	49

SUMÁRIO

1	INTRODUÇÃO	1
1.1	Contextualização e Motivação	1
1.2	Descrição do Problema	2
1.3	Objetivos	3
1.4	Metodologia	3
1.5	Estrutura da Dissertação	4
2	REVISÃO TEÓRICA	5
2.1	Considerações Iniciais	5
2.2	Ontologias	5
2.3	ValiPar	7
2.3.1	<i>Teste de Programas Concorrentes</i>	8
2.3.2	<i>A Ferramenta ValiPar</i>	8
2.4	Trabalhos Relacionados	10
2.4.1	<i>Resumo dos trabalhos relacionados</i>	13
2.5	Considerações Finais	14
3	ÁREAS DE ESTUDO DE DESEMPENHO	15
3.1	Considerações Iniciais	15
3.2	Avaliação de Desempenho	15
3.2.1	<i>Planejamento de Experimentos</i>	16
3.2.2	<i>Técnicas de Avaliação</i>	17
3.3	Análise de Resultados	19
3.3.1	<i>Metodologia para Avaliação de Desempenho</i>	19
3.4	Engenharia de Desempenho de Software	20
3.4.1	<i>Definições</i>	20
3.4.2	<i>Métodos</i>	22
3.5	Teste de Desempenho de Software	24
3.5.1	<i>Metodologia para o Teste de Desempenho de Software</i>	25
3.6	Mapas Conceituais	26
3.7	Considerações Finais	29
4	RELACIONAMENTO ENTRE AS ÁREAS	31
4.1	Considerações Iniciais	31

4.2	Lista de Conceitos e Definições	31
4.3	Validação da Lista de Conceitos	34
4.4	Ontologia Reunindo as Três Diferentes Áreas	35
4.4.1	<i>Taxonomia</i>	36
4.4.2	<i>Propriedades da Ontologia</i>	37
4.4.3	<i>Axiomas</i>	37
4.4.4	<i>Comparação</i>	38
4.5	Mapa Conceitual Unificando as Áreas	39
4.6	Considerações Finais	39
5	METODOLOGIA DE ANÁLISE DE DESEMPENHO	41
5.1	Considerações Iniciais	41
5.2	Metodologia de Análise de Desempenho	41
5.3	Uso da Metodologia na ValiPar	44
5.3.1	<i>Detalhamento da Execução</i>	49
5.3.2	<i>Resultados</i>	51
5.4	Análise e Comparação com Outras Metodologias	58
5.5	Considerações Finais	60
6	CONCLUSÕES	61
6.1	Conclusões do Trabalho	61
6.2	Principais Contribuições	62
6.3	Dificuldades Encontradas	62
6.4	Publicações	63
6.5	Trabalhos Futuros	63
	REFERÊNCIAS	65
ANEXO A	LISTA DE CONCEITOS E DEFINIÇÕES	69

INTRODUÇÃO

1.1 Contextualização e Motivação

O desempenho tem um papel fundamental no ciclo de vida de um sistema computacional, seja para satisfazer as necessidades do usuário final bem como comparar com outros sistemas. O desempenho de um sistema computacional pode ser caracterizado através de seu comportamento e quantificado por meio de métricas. Ele deve ser considerado durante as fases de projeto e implementação, como quando o sistema já está pronto. Deve também ser considerado tanto nos sistemas computacionais desenvolvidos em *hardware* como em *software*, para atender no mínimo as expectativas para aquele sistema.

Diferentes tipos de estudos que analisam o desempenho de sistemas computacionais surgem com frequência, retratando a importância que esse tema tem. Ele pode ser tratado de diferentes formas, seja para projetar um *software* considerando-se o desempenho, ou verificar se o desempenho de um *software* é o mesmo que foi requerido para ele, ou ainda quantificar o desempenho de um sistema computacional e comparar diferentes configurações desse sistema.

A fim de abordar as formas que o desempenho pode ser estudado, diferentes áreas, dentro da Ciência da Computação, se desenvolveram com objetivos e nomes diferentes. Como exemplo, pode-se citar as seguintes áreas: Avaliação de Desempenho (AD) (JAIN, 1991), Engenharia de Desempenho de Software (EDS) (SMITH; WILLIAMS, 2001) e Teste de Desempenho de Software (TDS) (MOLYNEAUX, 2009). Os objetivos de cada uma dessas áreas são:

- **Avaliação de Desempenho:** A área de AD tem como objetivo avaliar um sistema (*software* e/ou *hardware*) a fim de quantificar seu desempenho, seja a partir do próprio sistema ou de um modelo representativo.
- **Engenharia de Desempenho de Software:** A área de EDS tem como objetivo projetar e desenvolver *softwares*, visando o desempenho, o qual é considerado durante todo o ciclo

de vida desse *software*, seja para definir os requisitos de desempenho necessários para o projeto ou durante a implementação.

- **Teste de Desempenho de Software:** A área de TDS tem como objetivo testar *softwares* depois que eles foram implementados, determinar suas características de desempenho e comparar com aquelas especificadas nos requisitos desse *software* para verificar se são compatíveis ou não.

Em todas as áreas apresentadas, o desempenho é o ponto central, seja para fazer uma avaliação de um sistema (AD), ou para projetar um *software* (EDS), ou mesmo para verificar se o desempenho de um *software* é o esperado para ele (TDS). Cada uma das áreas pode ser vista como um conjunto de conceitos (termos e métodos) que ela utiliza. Existem conceitos que são tratados pelas três áreas apresentadas, como por exemplo "carga de trabalho" ou "métricas de desempenho". Outros conceitos são definidos somente por uma, ou por duas áreas, devido às diferenças entre elas, como por exemplo "requisitos de desempenho", que não é considerado na área de AD.

1.2 Descrição do Problema

Considerando a existência de diferentes áreas de estudo de desempenho, foi possível perceber que elas apresentam vocabulários diferentes, embora todas elas tratem o desempenho como ponto central e em outras vezes tratam os mesmos conceitos, embora tenham objetivos diferentes.

Um outro problema a ser destacado é que uma metodologia desenvolvida considerando os conceitos de, por exemplo, Teste de Desempenho de Software, pode não ser totalmente compreendida por um profissional que trabalhe na área de Avaliação de Desempenho. Os profissionais dessas áreas podem estar realizando a mesma atividade (analisar o desempenho de sistemas), apenas com uma visão diferente entre si, e estariam replicando esforços.

As áreas de AD, EDS e TDS utilizam termos e métodos que são diferentes entre si, porém podem ser reunidos para mostrar que eles têm o mesmo significado, enquanto outros métodos e termos possuem exatamente o mesmo nome porém são tratados de formas diferentes pelas áreas. Surge, dessa forma, a necessidade de um estudo que ilustre que tais áreas podem ser relacionadas, mostrando quais os conceitos entre essas áreas têm o mesmo significado, e que mostre também uma formalização desse relacionamento.

Tal relacionamento não é trivial, já que correlacionar tais áreas depende de uma compreensão de cada uma delas. Além disso, a fim de evitar a replicação de esforços de diferentes especialistas, uma metodologia de análise de desempenho que possa agregar conceitos de cada uma das áreas também poderia existir.

1.3 Objetivos

Tendo em vista o problema descrito, os objetivos deste trabalho de mestrado são: mostrar que as áreas de Avaliação de Desempenho, Engenharia de Desempenho de Software e Teste de Desempenho de Software possuem relação uma com a outra; e propor uma metodologia de análise de desempenho, considerando o relacionamento existente, que contenha conceitos dessas áreas que são tratadas neste trabalho.

O relacionamento entre as áreas foi feito mostrando quais conceitos (termos e métodos) são comuns. Para isso, foram pesquisados os conceitos e suas definições a fim de observar quais deles têm o mesmo significado. É importante destacar que não serão apresentados todos os conceitos que as áreas tratadas nesse trabalho têm em comum, dado a grande abrangência das mesmas.

Dessa forma, o escopo desse trabalho de mestrado é mostrar que as áreas de AD, EDS e TDS de fato se relacionam, sendo que os conceitos a serem apresentados e relacionados cumprem esse objetivo. Para formalizar o relacionamento existente, foi construída uma ontologia. Também utilizando esses conceitos, foi possível propor uma metodologia de análise de desempenho considerando as três áreas tratadas.

Com a proposta de uma metodologia feita, um estudo de caso foi desenvolvido utilizando a ferramenta ValiPar (SOUZA; SOUZA; ZALUSKA, 2014), a fim de analisar o desempenho da mesma, principalmente com relação a sua escalabilidade. A ferramenta realiza testes em programas concorrentes, que usam tanto o paradigma de passagem de mensagem quanto de memória compartilhada.

1.4 Metodologia

Para poder cumprir com os objetivos estabelecidos anteriormente, este projeto de mestrado seguiu algumas etapas, as quais são detalhadas nos próximos capítulos dessa dissertação de mestrado.

Primeiramente, foi feita uma revisão da literatura em busca de trabalhos relacionados. Essa revisão se estendeu durante todo o projeto de mestrado.

Em seguida, foi feito um estudo de cada uma das áreas tratadas neste trabalho utilizando as principais referências de cada uma delas e, a partir desse estudo, foram também desenvolvidos mapas conceituais para cada uma das áreas.

Em seguida ao estudo das áreas, o relacionamento entre as mesmas foi proposto. Para isso, foi construída uma lista com os conceitos dessas áreas e suas definições, para que os significados pudessem ser comparados, e observar se as áreas têm conceitos em comum. A escolha dos conceitos que estariam presentes na lista foi feita a partir dos mapas conceituais,

porém não foram limitados a eles.

Então uma ontologia para formalizar os relacionamentos existentes foi desenvolvida. Para a construção dessa ontologia, cada entrada da lista tornava-se um elemento da mesma, sendo possível relacioná-los a partir das mesmas definições observadas na lista.

Por fim, foi feita uma proposta de metodologia de análise de desempenho de sistemas computacionais contendo conceitos das três áreas. Essa metodologia foi então aplicada na ferramenta ValiPar observando sua escalabilidade e fatores que afetam seu desempenho.

1.5 Estrutura da Dissertação

O restante dessa dissertação de mestrado está organizada da seguinte maneira:

- Capítulo 2: Neste capítulo são apresentados os conceitos teóricos utilizados no desenvolvimento do trabalho (ontologias e da ferramenta ValiPar). Os conceitos teóricos das áreas relacionadas ao desempenho descritas anteriormente, devido à importância dos mesmos, são apresentados em um outro capítulo. Também são apresentados os trabalhos relacionados.
- Capítulo 3: Neste capítulo são apresentadas as áreas de estudo de desempenho tratadas neste trabalho: Avaliação de Desempenho, Engenharia de Desempenho de Software e Teste de Desempenho de Software. Além disso, são apresentados os mapas conceituais desenvolvidos para cada uma dessas áreas.
- Capítulo 4: Neste capítulo são apresentados como as áreas tratadas neste projeto se relacionam, como foi feito esse relacionamento, e também a ontologia construída.
- Capítulo 5: Neste capítulo são apresentadas a metodologia proposta neste trabalho e sua aplicação na análise de desempenho da ferramenta ValiPar.
- Capítulo 6: Este capítulo apresenta a conclusão do trabalho realizado, mostrando as principais contribuições, as publicações realizadas e possíveis trabalhos futuros.

REVISÃO TEÓRICA

2.1 Considerações Iniciais

Neste capítulo são apresentados os conceitos utilizados no desenvolvimento do projeto, que não estão relacionados às áreas relacionadas a desempenho. Primeiramente são apresentados os conceitos teóricos de ontologias, ilustrando a construção de uma ontologia.

Neste capítulo é apresentada também a ferramenta ValiPar (SOUZA; SOUZA; ZALUSKA, 2014). Como já descrito no Capítulo 1, a ValiPar foi utilizada para validar a metodologia de análise de desempenho proposta. São realçados conceitos de teste de programas concorrentes e os módulos da ferramenta.

Por fim, são apresentados os trabalhos encontrados na literatura que são relacionados a este trabalho de mestrado.

2.2 Ontologias

Segundo Gruber (1995), uma ontologia pode ser definida como "especificação formal e explícita de uma abstração, ou uma visão simplificada de um domínio de conhecimento". Porém, em Noy e McGuinness (2001) é feita uma expansão desta mesma definição de ontologia, como sendo uma "descrição formal e explícita dos conceitos de um domínio de conhecimento, das suas propriedades (atributos e relacionamentos) e restrições".

As ontologias podem ser utilizadas para criar ou padronizar um vocabulário específico do domínio de conhecimento que a mesma aborda (NAKAMURA, 2010). A maneira pela qual a ontologia é criada depende do grau de formalidade que a mesma apresenta, podendo ser de quatro níveis diferentes (USCHOLD; GRUNINGER, 1996):

- Altamente Informal: vocabulário expresso unicamente em linguagem natural;

- Semi-Informal: vocabulário expresso em alguma linguagem natural restrita e estruturada, reduzindo a ambiguidade;
- Semi-Formal: vocabulário expresso em uma linguagem formalmente definida;
- Rigorosamente Formal: vocabulário expresso em conceitos formais, teoremas e provas;

Independentemente do nível em que a ontologia está implementada, para que seja útil, é necessário que a mesma seja compreendida por seus usuários.

O conjunto de conceitos tratados por uma ontologia é chamado de seu domínio, e embora cada ontologia trate domínios diferentes, existem alguns elementos que compõem qualquer uma delas ([MARTIMIANO, 2006](#)):

- **Objetos:** conceitos de representação do mundo real, que podem ser agrupados em classes. As classes podem formar uma hierarquia, e a esta é dada o nome de taxonomia;
- **Atributos:** valores variados que os objetos podem assumir, definidos através das propriedades de dados;
- **Relacionamentos:** relação entre objetos ou entre as classes, feita através de propriedades de objeto;

Com relação às metodologias para a criação de ontologias, pode-se citar a metodologia proposta por [Noy e McGuinness \(2001\)](#), a Methontology ([A; GOMEZ-PEREZ; JURISTO, 1997](#)) e a metodologia proposta por [Uschold e Gruninger \(1996\)](#). O desenvolvimento da ontologia deste trabalho seguiu as etapas da metodologia de [Noy e McGuinness \(2001\)](#) por se mostrar mais adequada a este projeto e facilidade de segui-la. Tal metodologia está descrita a seguir:

1. Determinar domínio e escopo da ontologia: deve-se definir o domínio da ontologia, sua finalidade e quem deve usá-la e mantê-la;
2. Considerar reuso das ontologias existentes: considerar outras ontologias criadas e já validadas é bastante válido, mais ainda se esta outra ontologia tem o mesmo escopo da que se deseja propor;
3. Enumerar conceitos importantes da ontologia: deve-se enumerar os conceitos que são relevantes para a ontologia a ser criada. Tal lista ainda não precisa se preocupar com relacionamentos entre conceitos, nem mesmo se há sobreposições entre eles ou se esses conceitos serão classes;
4. Definir classes e hierarquias: nessa etapa os conceitos são representados como classes e os relacionamentos entre essas classes são definidos. Esta etapa é considerada a mais importante no desenvolvimento da ontologia;

5. Definir atributos de cada classe: atributos definem a classe e podem ser adquiridos da lista de conceitos criada anteriormente. Os atributos devem ser vinculados a uma classe ou a um objeto;
6. Definir restrições de atributo: as restrições podem ser o tipo de valor armazenado ou mesmo valores máximos ou mínimos;
7. Definir instâncias: esta etapa envolve escolher uma classe, criar uma instância dela e dar valores aos seus atributos.

Quanto às linguagens para se implementar uma ontologia, a princípio pode-se usar qualquer linguagem de programação para formalizar uma ontologia. A escolha dessa linguagem deve estar de acordo com os propósitos da ontologia a ser criada. Porém, na prática, poucas linguagens são usadas para este fim. Existem linguagens específicas para ontologias, entre elas pode-se citar a *Web Ontology Language* (OWL) (GARCÍA-PEÑALVO; GARCÍA; THERÓN, 2011).

A OWL (MCGUINNESS; HARMELEN *et al.*, 2004) é uma linguagem de marcação semântica para a publicação e compartilhamento de recursos. Ela pode ser aplicada para entender o significado de conceitos. Além disso, com essa linguagem é possível representar explicitamente os vocabulários dos conceitos e os relacionamentos do domínio de conhecimento da ontologia a ser construída. A OWL estende o vocabulário da *eXtensible Markup Language* (XML), sendo mais apropriada para implementar classes e o relacionamento entre essas classes.

O desenvolvimento de uma ontologia não é uma tarefa simples, especialmente porque se deve seguir uma série de passos e durante esse desenvolvimento é gerada uma série de documentos. Desse modo, existem diversas ferramentas para modelar, projetar e avaliar ontologias. Dentre essas ferramentas pode-se citar o Protegé, uma ferramenta bastante utilizada.

A ferramenta Protegé (GENNARI *et al.*, 2003) vem sendo desenvolvida desde 1997 pela universidade de Stanford, e tem como principal objetivo auxiliar no desenvolvimento de ontologias (GENNARI *et al.*, 2003).

A ferramenta é implementada em linguagem Java com código aberto, e permite incorporar *plugins* e APIs (*Application Programming Interface*), que podem ser usados para estender ou mesmo modificar o comportamento da ferramenta, tornando-a flexível (MARTIMIANO, 2006). Ela é composta por estruturas que permitem a criação, manipulação e visualização de ontologias em diferentes formatos.

2.3 ValiPar

A ferramenta ValiPar (SOUZA; SOUZA; ZALUSKA, 2014) faz parte do projeto TestPar (SOUZA *et al.*, 2013) que tem como objetivo prover técnicas para a realização de testes em

programas concorrentes. Já o objetivo da ferramenta é realizar testes em programas concorrentes auxiliando assim o testador.

A ferramenta será apresentada, pois ela foi utilizada como estudo de caso na aplicação da metodologia de análise de desempenho proposta. A fim de explicar melhor a ferramenta, são apresentados alguns conceitos de teste de programas concorrentes.

2.3.1 *Teste de Programas Concorrentes*

Os programas concorrentes têm a característica de serem não-determinísticos, o que torna a atividade de teste mais complicada. Ou seja, um mesmo valor de entrada pode ter saídas diferentes. Esse comportamento ocorre devido à comunicação, sincronização e ao paralelismo existente entre os diferentes processos. Desse modo, é preciso, primeiro, entender quais os tipos de defeitos a serem encontrados em programas concorrentes.

Os tipos de defeitos encontrados em programas concorrentes são os de observabilidade, travamento e o erro computacional. O defeito de observabilidade está relacionado ao ambiente de teste, e ocorre quando o testador não pode controlar o comportamento do programa que está executando em paralelo (SOUZA; SOUZA; VERGILIO,).

Outro tipo de defeito é o de travamento, que pode ocorrer, por exemplo, quando um processo fica travado esperando um recurso ser liberado, ou uma mensagem que nunca chegará, formando assim um *deadlock* (SOUZA; SOUZA; VERGILIO,). Além desses defeitos, existe também o erro computacional, que ocorre quando a sequência lógica do programa está correta, porém existe alguma operação computacional que está implementada de maneira errada.

Para testar os programas concorrentes, pode-se usar os testes estruturais (SARMANHO *et al.*, 2008), que devem começar com a construção do grafo de fluxo de controle com algumas modificações. Em Souza *et al.* (2008) é definido o grafo de fluxo de controle paralelo. Neste grafo, além dos nós e arestas presentes no grafo de fluxo de controle comum, são adicionadas as arestas que fazem a comunicação entre processos, as quais são chamadas de arestas de sincronização.

Aos elementos que compõem o grafo que o teste deve cobrir, é dado o nome de elementos requeridos do teste. Além dos nós, arestas e arestas de sincronização, outros elementos requeridos são as definições de variáveis, a utilização de variáveis, entre outros. A partir dos elementos requeridos, pode-se realizar o cálculo de quanto um determinado teste realizado cobriu de seus elementos e então retornar uma porcentagem de cobertura ao testador.

2.3.2 *A Ferramenta ValiPar*

A ValiPar é uma ferramenta que realiza testes em programas concorrentes, a partir de critérios. A arquitetura da ferramenta é mostrada na Figura 1. A ValiPar tem quatro módulos principais: ValiInst, que realiza toda a análise estática do programa concorrente; ValiElem, que

gera a lista de elementos requeridos; ValiEval, que realiza avaliação de casos de teste; e ValiExec, que faz a execução do teste (SOUZA *et al.*, 2013).

Figura 1 – Arquitetura da ValiPar (SOUZA; SOUZA; ZALUSKA, 2014)

- **ValiInst**

O módulo ValiInst é responsável por extrair informações do fluxo do programa paralelo e o instrumenta com comandos que registram os caminhos reais de execução.

- **ValiElem**

A ValiElem gera os elementos requeridos das coberturas de casos de teste. Tais elementos são gerados a partir do grafo de fluxo de controle paralelo (GFCP) e do fluxo de dados proveniente da ValiInst.

- **ValiEval**

A ValiEval avalia a cobertura obtida por um caso de teste definido por um determinado critério. A ValiEval usa os descritores, os elementos requeridos gerados pela ValiElem e os caminhos executados pelos casos de teste para verificar quais os elementos requeridos de um determinado critério de teste foram executados.

- **ValiExec**

A ValiExec executa o programa instrumentado com os dados de teste fornecidos pelo usuário. Um *script* é usado para inicializar o ambiente de passagem de mensagens antes da execução do programa paralelo. A ValiExec armazena o caso de teste, os parâmetros de execução e de seu respectivo rastro.

A ferramenta trabalha com o conceito de sessões de teste, que pode ser configurado para testar um determinado programa paralelo, o que permite parar a atividade de teste e retomá-la em

uma outra oportunidade (SOUZA *et al.*, 2008). O que a ferramenta faz é fornecer funcionalidades para criar sessões de teste, salvar e executar os dados de teste e, por fim, avaliar a cobertura do teste a partir de um dado critério.

A ferramenta testa as possibilidades de casamento entre *sends* e *receives*, formando assim várias possíveis execuções do programa. Essa versão da ferramenta é estritamente sequencial, ou seja, cada uma das execuções do programa concorrente é feita uma seguida da outra. Em Batista (2015) é feita uma outra versão da ferramenta que executa essas diferentes versões de execução do programa de forma paralela. Na Figura 2 é mostrada essa arquitetura agora com um novo módulo, a ValiSync. Além desse módulo, os módulos ValiExec e ValiEval também trabalham de forma um pouco diferente da versão anterior.

Figura 2 – Arquitetura ValiPar (BATISTA, 2015)

Nessa arquitetura, a ValiExec é distribuída entre diferentes máquinas que processam paralelamente as versões de execução do programa concorrente. A ValiEval faz o cálculo de cobertura a partir dos arquivos de rastro que chegam dessas diferentes instâncias da ValiExec, e a ValiSync faz a geração de diferentes possibilidades de execução do programa e distribui para as instâncias da ValiExec.

2.4 Trabalhos Relacionados

Primeiramente, esta seção descreve as ontologias já desenvolvidas e encontradas na literatura, as quais relacionam-se com o desempenho de sistemas computacionais, principalmente com a análise de desempenho. São apresentados trabalhos que especificam ontologias das áreas de desempenho de maneira separada, pois não foi encontrado um trabalho que contivesse as três

áreas juntas. Dessa forma, os quatro primeiros trabalhos apresentados são referentes a ontologias para o desempenho de sistemas.

- *An Approach to Ontology-aided Performance Engineering through NFR Framework* - 2007 ([SANCHO et al., 2007](#))

Neste trabalho é proposto um conjunto de regras semânticas a fim de formalizar o conhecimento sobre a Engenharia de Desempenho de Software. O que se propõe é uma base de dados ontológica que contenha todo o conteúdo desta área.

Este trabalho foi desenvolvido especialmente para formalizar os conceitos que a área de Engenharia de Desempenho de Software tem em comum com a Engenharia de Software, e evitar ambiguidades entre esses conceitos.

Dessa forma, permite-se que os termos da Engenharia de Desempenho possam ser relacionados ou mesmo que estes se relacionem com outros requisitos não-funcionais, além do desempenho. Tais termos são métricas, como o tempo de resposta, além de técnicas utilizadas para atingir um bom desempenho.

- *Ontology-based Modeling of Production Systems for Design and Performance Evaluation* - 2014 ([TERKAJ; URGO, 2014](#))

Neste trabalho é desenvolvido um modelo de dados baseado em ontologias para o projeto e avaliação de desempenho de sistemas de produção, utilizados pela indústria. O objetivo deste trabalho é interligar a modelagem da representação espacial de objetos físicos da indústria em questão com a caracterização de seus estados e comportamento.

O trabalho utiliza técnicas de Web semântica, em especial a OWL, para solucionar problemas de interoperabilidade entre sistemas que podem ser desenvolvidos com diferentes estruturas de dados e que empregam diferentes tecnologias. Seu foco é fornecer um modelo de dados mais completo de sistemas de produção e dos seus recursos, ligando a caracterização dos recursos de produção com os resultados das atividades de avaliação de desempenho em termos de um histórico de desempenho.

- *An Ontology for Guiding Performance Testing* - 2014 ([FREITAS; VIEIRA, 2014](#))

Neste trabalho foi desenvolvida uma ontologia para representar o conhecimento sobre Teste de Desempenho, guiando assim o testador em como proceder para a realização de tais testes. Além disso, os autores apresentam uma revisão sistemática da literatura a respeito de ontologia de teste de software e desempenho de software, e chegam a conclusão que não há nenhuma outra ontologia que cobre especificamente o domínio de teste de desempenho.

Segundo os autores, para reunir todo o conhecimento a respeito de teste de desempenho de software, o uso de ontologias é ideal pois podem prover uma escolha de termos precisa e podem ser usadas para a comunicação entre testadores, desenvolvedores e usuários.

Dessa forma, o trabalho tem como objetivo ajudar os testadores a definir o que se deve levar em consideração na execução de testes de desempenho. A ontologia deve indicar, por exemplo, ferramentas de teste de desempenho, atividades, métricas, objetivos e artefatos, ajudando assim na decisão do testador sugerindo uma metodologia para o desenvolvimento de teste e a geração do plano de teste de *software*. De modo geral, a ontologia deve responder porque e como fazer testes de desempenho e quais resultados obter.

A ontologia construída engloba nove termos gerais, sendo eles: a atividade de teste, os artefatos, a especificação do sistema sob teste, os tipos de teste de desempenho, as características do testador, as métricas de desempenho, os conceitos que não fazem parte do domínio mas mesmo assim são utilizados, os objetivos do teste e ferramentas para a execução de testes de desempenho. É possível observar tais termos da ontologia na Figura 3.

Figura 3 – Ontologia de Teste de Desempenho (SILVA, 2013)

- *How far are we from the definition of a common software performance ontology* - 2005 (CORTELLESA, 2005)

Este é um artigo onde o autor discute se é possível a existência de uma ontologia que contenha os conceitos e relações a respeito de desempenho de *software*, embora o próprio autor revele que diferentes vocabulários para diferentes dados de desempenho são usados em diferentes abordagens.

O foco desta discussão é sobre os modelos e métricas de desempenho de *software*. Conceitos como medição, monitoramento e gerenciamento de desempenho e geração de carga de trabalho não são muito detalhados, apesar disto, afirma-se que se tal ontologia existisse, tais termos deveriam ser considerados.

O artigo não propõe uma ontologia que unifique conceitos de desempenho de software e, ao invés disso, discute se é possível a existência de tal ontologia.

Nessa discussão são apontados termos que, se tal ontologia existisse, deveria tratar, como a capacidade de representar os resultados de uma análise de desempenho, que devem ser representados de uma forma única, identificando assim, gargalos e possíveis alternativas de projeto do sistema. Além disso, deve tratar como o *software* se comportará sobre uma certa carga de trabalho.

Esses trabalhos abordam ontologias com as três áreas de estudo de desempenho envolvidas neste projeto, embora tratem elas de maneira separadas com exceção do trabalho de Cortellessa (2005), porém não deixa claro que a unificação proposta trata diferentes áreas da Ciência da Computação. O próximo trabalho apresenta uma metodologia que integra teste e avaliação de desempenho especificamente para sistemas de tempo real.

- *A methodology for design, test, and evaluation of real-time systems* - 2002 (SHERE; CARLSON, 2002)

A metodologia apresentada neste trabalho é adequada para ser usada como parte de uma abordagem de prototipagem de sistemas de tempo real. Esta metodologia pode ser usada na modelagem e na simulação, bem como em testes para desenvolvimento ao longo do ciclo de vida do software.

A metodologia foi projetada com o objetivo de reduzir os custos do projeto e encurtar o tempo gasto com a avaliação de desempenho e os testes de integração, já que identifica os problemas mais facilmente e com um custo menor, facilitando a correção deles.

Embora esta metodologia use conceitos dessas duas áreas, ela não tem como objetivo utilizar os conceitos das duas áreas ao mesmo tempo, e sim separadamente.

2.4.1 *Resumo dos trabalhos relacionados*

Considerando os trabalhos relacionados encontrados e este trabalho de mestrado, foram destacadas algumas características principais, com o objetivo de comparar tais trabalhos. Tais características são:

1. Faz o uso de Ontologias;
2. Foca na área de Engenharia de Desempenho;

3. Foca na área de Avaliação de Desempenho;
4. Foca na área de Teste de Desempenho de Software;
5. Propõe uma metodologia.
6. Tem como objetivo guiar a análise de desempenho de um software;

Já na Tabela 1 são mostradas quais características referem-se a qual trabalho relacionado, fazendo, dessa forma, uma comparação entre eles, com este trabalho.

Tabela 1 – Tabela com Resumo dos Trabalhos Relacionados

Trabalhos	1	2	3	4	5	6
(SANCHO <i>et al.</i>, 2007)	X	X				
(TERKAJ; URGU, 2014)	X		X			
(FREITAS; VIEIRA, 2014)	X			X		X
(CORTELLESA, 2005)	X					X
(SHERE; CARLSON, 2002)			X	X	X	X
Trabalho Proposto	X	X	X	X	X	X

2.5 Considerações Finais

Neste capítulo foram apresentados dois dos conceitos teóricos utilizados no desenvolvimento deste trabalho de mestrado.

Primeiramente foram mostrados os conceitos sobre ontologias, suas definições encontradas, seus componentes e a metodologia para a construção de uma ontologia. Esses conceitos foram apresentados pois este trabalho propõe uma ontologia que reúne conceitos de áreas de estudo relacionadas a desempenho.

Em seguida foi apresentada a ferramenta ValiPar, seus módulos e as versões existentes da ferramenta. Além disso, foram mostrados alguns conceitos sobre teste de programas concorrentes. A ferramenta foi apresentada, pois ela foi utilizada como estudo de caso da metodologia de análise de desempenho proposta neste trabalho.

Por fim, foram apresentados os trabalhos relacionados a este trabalho de mestrado, e também uma comparação entre eles.

No próximo Capítulo são apresentadas as áreas de estudo de desempenho estudadas.

ÁREAS DE ESTUDO DE DESEMPENHO

3.1 Considerações Iniciais

Este capítulo tem o objetivo de apresentar as áreas de desempenho envolvidas neste projeto: Avaliação de Desempenho, Engenharia de Desempenho de Software e Teste de Desempenho de Software. É apresentado um resumo do estudo feito para cada uma das áreas, detalhando termos e métodos.

Os mapas conceituais desenvolvidos para cada uma dessas áreas também são detalhados, de modo a destacar os termos e métodos, e promover um melhor entendimento de seus conceitos.

3.2 Avaliação de Desempenho

É possível verificar a importância da avaliação de desempenho de sistemas computacionais, sendo crucial, por exemplo, na escolha de qual o melhor ambiente computacional para um determinado cenário. A avaliação de desempenho pode apoiar a escolha do sistema operacional, da rede local e de outros elementos que compõem um determinado sistema. Se não escolhidos de maneira correta, é possível deteriorar o desempenho de todo o sistema computacional, gerando gastos desnecessários (FORTIER; MICHEL, 2003).

Quando se avalia o desempenho, buscam-se métricas que indiquem qualitativamente ou quantitativamente a eficiência de um serviço prestado. Também pode ser utilizada para determinar se um sistema computacional atinge as expectativas e necessidades daqueles que projetam ou usam tal sistema.

Três etapas são necessárias para se realizar uma Avaliação de Desempenho, sendo elas: o Planejamento de Experimentos, que é uma técnica utilizada para definir quais dados, em que quantidade e em que condições devem ser coletados durante um determinado experimento; as Técnicas de Avaliação, que são técnicas de como coletar os dados do sistema a ser avaliado; e,

por fim, a Análise dos Resultados, que é a caracterização dos resultados coletados.

3.2.1 Planejamento de Experimentos

O planejamento de experimentos designa toda uma área de estudos da Estatística que desenvolve técnicas de planejamento e análise de experimentos.

Para que se possa planejar experimentos de forma controlada, é feita a manipulação de certas variáveis independentes (fatores), definindo-se os valores mais prováveis que essas variáveis podem assumir (níveis) para verificar o efeito que esta manipulação provoca nas variáveis dependentes (variáveis de resposta) (JAIN, 1991). Para uma melhor confiabilidade nos resultados, fazem-se necessárias replicações dos experimentos, ou seja, há uma repetição controlada.

Dessa forma, as técnicas ajudam a planejar os experimentos, cobrindo a totalidade das possibilidades: diversos fatores, diferentes quantidades de níveis e tratamento de replicações. Em outras palavras, tais técnicas definem em quais condições os experimentos devem ocorrer.

Para a Avaliação de Desempenho, é a partir dos experimentos que se obtém as informações do sistema que se deseja avaliar, as quais podem ser coletadas tanto a partir do próprio sistema computacional como a partir de um modelo representativo do mesmo.

Os objetivos do planejamento de experimentos são obter o máximo de informação com um número mínimo de experimentos, permitindo separar os efeitos de vários fatores no resultado observado, e determinar o quão significativo é o efeito dos fatores nos resultados observados (JAIN, 1991). Busca-se assim satisfazer dois grandes objetivos: a maior precisão estatística possível nos resultados, e um menor custo.

Outro conceito importante é o da carga de trabalho que será imposta ao sistema, que segundo Menascé *et al.* (2003) é o conjunto de todas as informações de entrada que um sistema recebe durante um período de tempo determinado. É preciso definir antecipadamente qual carga será imposta ao sistema, que pode ser diferente dependendo do objetivo da avaliação de desempenho a ser feita. Ela pode representar o sistema em condições normais ou em condições não usuais.

Existem três formas para se realizar o planejamento de experimento: Simples, Fatorial Completo e Fatorial Parcial (JAIN, 1991).

O planejamento fatorial simples parte de uma configuração inicial, com todos os fatores fixados em um nível. Para gerar as combinações, apenas um deles é variado por vez. Este modelo tem como principal objetivo verificar qual fator afeta mais o desempenho. É fácil de ser realizado, porém não permite verificar relação entre os fatores.

Já o planejamento fatorial completo utiliza todas as combinações possíveis geradas a partir das variações de todos os fatores e todos os níveis. Sua principal vantagem é que todos

os fatores são avaliados, e assim pode-se determinar o efeito de qualquer fator no resultado. Porém, dependendo da quantidade de fatores e níveis, é necessária uma grande quantidade de experimentos.

Já o planejamento fatorial parcial tem como objetivo minimizar os custos em se executar os experimentos. A partir do planejamento fatorial completo, é feita uma redução no número de experimentos, sendo excluídos aqueles em que a interação entre os fatores e o resultado não existe ou é insignificante.

3.2.2 Técnicas de Avaliação

Existem diversas técnicas para se avaliar o desempenho de um sistema computacional. As técnicas em questão se diferem em como são coletadas as informações referentes aos parâmetros requeridos para a avaliação (FORTIER; MICHEL, 2003).

Basicamente, as técnicas podem ser divididas em dois grupos, sendo eles: Técnicas baseadas na Aferição, na qual as informações são colhidas do próprio sistema; ou as Técnicas baseadas na Modelagem, na qual se colhem informações através de modelos que representam o sistema (JAIN, 1991).

As técnicas de aferição realizam a análise de desempenho a partir de coletas de dados feitas diretamente do próprio sistema a ser avaliado, utilizando para isso um *software* ou um *hardware* para obter informações do seu comportamento (JAIN, 1991). Esse grupo pode ser subdividido ainda em três métodos que envolvem sua execução, sendo eles: Coleta de Dados, *Benchmarking* e Prototipação.

A coleta de dados deve ocorrer em sistemas reais, sob cargas de trabalho que possivelmente revelam o comportamento do sistema. A coleta pode ser feita através de monitores, que podem ser programados em *software*, como desenvolvidos em *hardware* específico. Os monitores de *software*, por serem programados, são mais gerais e flexíveis, porém produzem interferência no sistema que se está coletando os dados. Já os monitores de *hardware* são menos invasivos porém possuem um custo maior para o desenvolvimento.

Os *benchmarks* são programas específicos, escritos em linguagem de alto nível, que avaliam o desempenho de uma parte específica do sistema ou do sistema como um todo, dependendo do objetivo do *benchmark* e das operações por ele executadas. Os *benchmarks* são utilizados quando se deseja comparar o desempenho de diferentes sistemas computacionais. Para isso, é necessário que todos os sistemas sejam submetidos às mesmas cargas de trabalho e sob as mesmas condições.

As técnicas com protótipos envolvem a construção simplificada do sistema real. É utilizada quando o sistema ainda não está disponível (BRUSCHI, 1997). A partir de então pode-se usar *benchmarks* ou monitores para se obter informações sobre o desempenho do sistema. Tal técnica costuma fornecer bons resultados, porém existe um custo na construção destes protótipos,

podendo se tornar inviável dependendo da dimensão do sistema real.

A outra técnica de avaliação de desempenho é a modelagem. Nela é necessário que se desenvolva um modelo representativo do sistema a ser analisado, com suas principais características. Tal modelo pode, então, ser resolvido através de simulações (KELTON; LAW, 2000) ou ainda analiticamente (matematicamente).

A modelagem analítica permite que se desenvolva uma relação entre as métricas de desempenho do sistema escolhidas e seus parâmetros (FORTIER; MICHEL, 2003). Tal relação é feita através de equações que podem ser resolvidas analiticamente. Se feita de maneira correta, esta técnica produz resultados precisos, porém, com uma complexidade alta, que pode aumentar conforme o sistema é ampliado. O custo de se implementar esta técnica é baixo e o tempo para a sua solução está diretamente relacionado com a complexidade do sistema avaliado.

Já a simulação exige que se tenha um modelo que represente fielmente o sistema, para que se possa fazer então um programa de tal modelo (BRUSCHI, 1997). Esta técnica tenta prever possíveis consequências no desempenho quando se altera alguma configuração do sistema. Esta técnica pode ser aplicada tanto em sistemas existentes quanto não existentes, tornando-se flexível.

Para o uso da técnica de modelagem, existem diversas maneiras que permitem a representação do sistema computacional, podendo-se destacar as Redes de Filas e as Redes de Petri.

As Redes de Filas são ideais para representar sistemas onde existe a formação de filas de solicitantes de serviços (JERRY, 1984), o que pode ocorrer caso o servidor que forneça o serviço já esteja atendendo outro solicitante. Tal modelo permite a representação de um ou mais servidores e uma ou mais filas por servidor. Porém, este tipo de modelo apresenta certas restrições, como em sua representação gráfica, que oferece apenas os elementos fila e servidor e não permite posse simultânea de recursos pelo mesmo solicitante, o que leva a uma perda da realidade quando estão representando sistemas reais. Este tipo de modelo é indicado para representar sistemas onde o objetivo é realizar uma avaliação quantitativa, obtendo informações de tempo de serviço, tempo total de atendimento do solicitante, tamanho da fila, entre outras.

Já as Redes de Petri (MURATA, 1989) são usadas para a descrição e a análise de concorrência e sincronização em sistemas paralelos, demonstrando ações cooperativas de diferentes entidades. Os elementos básicos de uma Rede de Petri são os lugares, que representam variáveis de estado, e as transições, que representam as ações realizadas pelo sistema. A utilização dessa técnica é apropriada quando se deseja validar o comportamento de um sistema.

A utilização de cada uma das técnicas já citadas depende do acesso ao sistema, ou em que estágio se encontra o trabalho. Por exemplo, caso se deseje avaliar o desempenho ou validar um projeto de sistema, pode-se usar a técnica de simulação para tentar prever possíveis comportamentos do sistema. Porém, caso o sistema já esteja pronto, ou caso deseje-se escolher

um sistema, as técnicas de aferição são mais indicadas.

3.3 Análise de Resultados

A etapa de análise de resultados é necessária para se interpretar os dados que foram obtidos com a utilização das técnicas de avaliação. É a partir dessa etapa que se pode obter conclusões a respeito do desempenho do sistema.

A partir de diversos resultados obtidos, pode-se calcular dois tipos de medidas: as de posição e as de dispersão. As medidas de posição referem-se ao valor que uma determinada métrica assume, como o cálculo da média ou da mediana.

Já as medidas de dispersão servem para dar mais confiança ao valor obtido pelas medidas de posição. Elas calculam quanto um valor variou nas vezes em que o experimento foi executado. Como exemplos dessas medidas, pode-se citar o intervalo de confiança e o *box-plot*.

Além do cálculo das medidas de posição e dispersão, pode ser calculado também a influência da variação dos fatores nas variáveis de resposta, porém esse cálculo só pode ser feito quando se escolheu o planejamento fatorial completo.

Para apresentar os resultados dos cálculos feitos, tabelas e gráficos são ideais para se comparar os diferentes cenários e então se obter conclusões sobre o desempenho do sistema que foi avaliado.

3.3.1 Metodologia para Avaliação de Desempenho

A metodologia utilizada para seguir com uma avaliação de desempenho, englobando o planejamento de experimentos, técnicas de avaliação e análise de resultados é apresentada a seguir ([JAIN, 1991](#)):

1. **Definir objetivos e definir o sistema a ser avaliado:** A primeira etapa é definir quais são os objetivos em se avaliar o sistema e, além disso, definir quais são os componentes desse sistema.
2. **Listar os serviços oferecidos pelo sistema:** Deve-se levar em consideração os serviços oferecidos.
3. **Selecionar métricas:** Deve-se escolher as métricas para comparação do desempenho. Em geral, as métricas são relacionadas com a velocidade, a precisão e a disponibilidade de serviços.
4. **Listar parâmetros:** O próximo passo é fazer uma lista de todos os parâmetros que podem afetar o desempenho. Esses parâmetros podem ser parâmetros do sistema, relacionados

ao *hardware* e *software* e parâmetros de carga de trabalho, como características das requisições de usuários.

5. **Selecionar fatores a serem estudados:** Deve-se definir as variáveis de entrada do experimento (fatores) e seus possíveis valores (níveis).
6. **Selecionar a técnica de avaliação:** Deve-se então escolher uma das técnicas de avaliação apresentadas na seção 3.2.2.
7. **Selecionar a carga de trabalho:** Deve-se escolher uma carga de trabalho representativa para o sistema, ou seja aquela que o sistema receberia normalmente em sua execução.
8. **Planejar os experimentos:** Então, deve-se escolher como realizar o planejamento de experimentos, como mostrado na seção 3.2.1, para então executar os experimentos.
9. **Analisar e interpretar os dados:** Com os dados obtidos a partir das execuções, deve-se analisá-los, comparando médias, medianas e levar em consideração parâmetros de variabilidade como intervalos de confiança.
10. **Apresentar os resultados:** Os resultados podem ser apresentados a partir de tabelas e gráficos.

3.4 Engenharia de Desempenho de Software

A Engenharia de Desempenho do Software (EDS) é uma coletânea de métodos para desenvolver softwares que têm como objetivo oferecer a execução do software com o desempenho requerido pelo usuário (SMITH, 1986). Esta área é, na verdade, uma subárea da Engenharia de Software, com foco no desempenho. Seu objetivo é analisar o desempenho do software durante seu ciclo de vida.

Seus métodos envolvem desde a utilização de processos de criação, advindos da Engenharia de Software, até análises de desempenho durante a criação do software, como construção e análises de modelos, utilizando, por exemplo, simulação. Segundo Woodside, Franks e Petriu (2007), a EDS pode ser definida como a coleção de atividades vindas da Engenharia de Software direcionadas para os requisitos de desempenho.

As próximas subseções apresentam, primeiramente, as definições dos termos usados na Engenharia de Desempenho de Software e seus métodos de desenvolvimento, segundo diferentes autores.

3.4.1 Definições

Muitas das definições apresentadas nesta seção também são usadas em outras áreas, como a Avaliação de Desempenho e Engenharia de Software.

Os termos relacionados abaixo foram extraídos da bibliografia buscada sobre Engenharia de Desempenho de Software (SMITH, 1997), porém não são os únicos da área e estão descritos por serem relevantes. São eles:

- **Requisitos:** definem objetivos, propriedades, restrições que o software deve possuir para satisfazer as especificações do usuário;
- **Custos:** são esforços gastos com a construção do software;
- **Estrutura do Sistema:** engloba as características e a arquitetura do sistema computacional;
- **Operações do Sistema:** são as funções do sistema computacional.

Como pode ser observado, estes termos são conhecidos, principalmente da Engenharia de Software. Ou seja, para considerar o desempenho, é necessário avaliar os requisitos do software, ponderando os custos de cada um deles. Deve-se considerar ainda as estruturas e operações do sistema computacional onde este software executará.

Além desses termos, na EDS também são determinadas e quantificadas métricas de desempenho usadas, as quais podem ser divididas em internas ou externas. As métricas de desempenho internas podem ser: o número de operações por unidade de tempo, como por exemplo, a taxa de transferência de um arquivo; ou o número de bytes transferidos por segundo; ou ainda a utilização de recursos, como processador ou memória. Já as métricas externas referem-se ao comportamento do sistema computacional como um todo, externo ao software, quando se executam algumas funções específicas deste software.

Em Smith (1997) é reunido todo um conjunto de informações necessárias para utilização das técnicas de Engenharia de Desempenho de Software. Tais termos são relacionados especificamente com o desempenho de sistemas. São eles:

- **Carga de Trabalho:** especifica o uso esperado do sistema;
- **Objetivos:** especifica os critérios quantitativos para a avaliação do sistema;
- **Modelo de Software:** descreve as fases de processamento do software;
- **Ambiente de execução:** descreve a plataforma e rede de comunicação necessária para a execução do software;
- **Recursos requeridos:** provê os recursos computacionais necessários, como acessos à base de dados e chamadas de sistema para comunicação;

3.4.2 Métodos

Assim como as técnicas de Avaliação de Desempenho, os métodos da Engenharia de Desempenho de Software, segundo [Woodside, Franks e Petriu \(2007\)](#), também podem ser divididos naqueles que são baseados na medição ou baseados em modelos.

Especificamente para a EDS, os baseados em medição não são muito utilizados, já que é necessário que o software esteja pronto. Segundo [Smith \(1986\)](#), tais técnicas são utilizadas na fase de teste de integração do sistema, e resulta em maiores custos para o desenvolvimento e manutenção.

Desse modo, as metodologias baseadas em modelos têm como objetivo construir primeiramente um modelo do sistema, testar seu desempenho, e então realizar a implementação. As metodologias apresentadas são divididas segundo os autores que as propõem.

Segundo [Rautenstrauch e Scholz \(2001\)](#), o processo para a construção de um software deve seguir pelo menos quatro fases: a Fase de Análise do Sistema, onde são buscados os requisitos do software a ser construído; Fase de *Design* do Sistema, que envolve a modelagem de funções e dados, e também se projetam os custos envolvidos e o desempenho que pode ser obtido; Fase de Implementação, onde se pode realizar otimizações em funções do código; e por fim a Fase de Teste, onde se verificam as características gerais do software.

Em [Schmietendorf, Scholz e Rautenstrauch \(2000\)](#), é apresentada outra metodologia que se difere um pouco das demais. O processo de construção é dividido em apenas três fases. São elas: Antes do Desenvolvimento, quando se definem e quantificam métricas de desempenho;

Figura 4 – Fases da EDS ([SMITH, 1986](#))

Fase de *Design* e Implementação, em que tais métricas são refinadas, o software implementado, e então certas funções podem ser avaliadas e alguns módulos podem ser testados; e por fim a Fase de Teste de Sistema, em que se faz um teste mais completo do software produzido.

Segundo [Smith \(1986\)](#), as fases de construção de um software, segundo a EDS, devem ser cíclicas, de modo que, após o projeto de cada módulo, deve-se construir um modelo representativo desse módulo, fazer análise de seu desempenho e mudar o projeto caso necessário, e se estiver tudo certo ele pode ser implementado. Na Figura 4 são apresentadas as fases da implementação de uma parte do ciclo de desenvolvimento do software, considerando os requisitos de desempenho.

[Bondi \(2014\)](#) também propõe etapas, as quais são apresentadas na Figura 5. Observa-se que é a partir dos requisitos funcionais, em conjunto com modelos de desempenho (como por exemplo redes de filas) que se define os requisitos de desempenho do software, para então construir uma arquitetura e implementá-la. Depois de implementada, são realizados testes funcionais e testes de desempenho para testar programa.

Figura 5 – Etapas da EDS, adaptado de [Bondi \(2014\)](#)

Assim como na Avaliação de Desempenho, os modelos de desempenho podem ser definidos utilizando Redes de Filas e Redes de Petri, e resolvidos analiticamente ou por simulação.

A Engenharia de Desempenho de Software, independentemente do tipo de método, usa o Teste de Desempenho de Software para testar o desempenho de programas construídos. A próxima seção apresenta os principais conceitos relacionados com o Teste de Desempenho de Software.

3.5 Teste de Desempenho de Software

O Teste de Desempenho de Software (TDS) é um tipo de teste de software (MEIER *et al.*, 2007). Nele são realizadas análises de desempenho para descobrir pontos no programa que contenham problemas, como por exemplo gargalos no software, ou para extrair algumas métricas de desempenho, como a responsividade, o tempo de resposta ou mesmo a vazão entre outros.

Apesar deste ser o principal objetivo do TDS, outros objetivos também podem ser alcançados ao se realizar um teste de desempenho, como por exemplo prever ou estimar as características de desempenho de uma aplicação, avaliar a capacidade do sistema ou ainda determinar as características de desempenho desejadas da aplicação antes e depois de alguma alteração (SUFFIAN; FAHRURAZI, 2012).

Para se obter resultados com o teste de desempenho, deve-se especificar anteriormente qual o desempenho esperado para o software, escolhendo uma métrica (MOLYNEAUX, 2009), e então comparar o resultado do teste realizado com o que foi determinado anteriormente. Neste caso, o testador deve ter acesso aos requisitos formulados pelos analistas.

Um teste de desempenho atinge seus objetivos se o resultado obtido não for compatível com o que foi especificado na construção do sistema, ou seja, se foi detectada uma especificação de desempenho do sistema que não foi cumprida. Ele pode ser usado para determinar ou validar a velocidade, escalabilidade e/ou características de estabilidade do produto em teste (SILVA, 2013).

Existem diversos testes de desempenho que podem ser implementados para se descobrir defeitos de desempenho de um sistema, sendo: (MEIER *et al.*, 2007):

- **Teste de Carga:** É usado para verificar o comportamento da aplicação sobre uma carga normal ou pouco acima dessa normalidade. Seu objetivo é verificar se o desempenho da aplicação é compatível com o requerido. Também pode determinar erros de funcionalidade do sistema quando submetido a essa carga.
- **Teste de Estresse:** É utilizado para determinar ou validar o comportamento de uma aplicação quando é submetida a ela uma carga de trabalho além do normal ou mesmo de pico, ou seja, uma carga que o sistema não está preparado para receber. O objetivo desse teste é revelar defeitos da aplicação que surgem somente sob condições de carga elevada. Estes defeitos podem ser, por exemplo, erros de sincronização.
- **Teste de Capacidade:** É usado para determinar quantos usuários ou quantas operações um determinado sistema irá atender cumprindo os requisitos de desempenho estabelecidos, ou seja, determinar o máximo que um sistema suporta. Também pode ser usado para determinar a escalabilidade de um sistema.

- **Teste de *Smoke*:** Esse não é um tipo de teste, é usado apenas para observar se o sistema irá responder o teste a ser feito. Esse teste é na verdade uma execução inicial de um teste de desempenho, para observar se a aplicação pode executar suas operações.
- **Teste de *Spike*:** É usado para determinar ou validar as características de desempenho do sistema em teste quando submetido a ele uma carga de trabalho que repetidamente aumenta além do previsto, em períodos de tempos curtos. Esse é na verdade um subconjunto do teste de estresse.
- **Teste de Validação:** Tem como objetivo comparar a velocidade, escalabilidade ou características de estabilidade do sistema em teste com as expectativas e requisitos de desempenho que foram definidas a ele.
- **Teste de Resistência:** É usado para determinar ou validar as características de desempenho do produto em teste, submetendo a ele uma carga de trabalho durante um período prolongado de tempo.

Já as métricas mais utilizadas no teste de desempenho são ([MOLYNEAUX, 2009](#)):

- **Disponibilidade do sistema:** tempo em que o sistema está disponível para o usuário final;
- **Tempo de resposta:** tempo que a aplicação leva para responder uma requisição do usuário;
- **Vazão:** taxa na qual uma aplicação é processada por unidade de tempo; e
- **Utilização de recursos:** porcentagem de capacidade teórica que um recurso pode ser usado.

O teste de desempenho é uma atividade ampla e complexa que pode assumir muitas formas, e muitos riscos caso os resultados não sejam analisados de forma correta ([LI; SHI; LI, 2013](#)). No entanto, podem fornecer uma variada gama de informações importantes para o desempenho no projeto de sistemas. É importante compreender os diferentes tipos de teste de desempenho a fim de reduzir os riscos, minimizar os custos, e saber quando aplicar o teste adequado para cada sistema ([MEIER et al., 2007](#)).

3.5.1 Metodologia para o Teste de Desempenho de Software

A metodologia utilizada para desenvolver um teste de desempenho de software é apresentada a seguir: ([MEIER et al., 2007](#))

1. **Identificar o ambiente de teste:** Identificar o ambiente físico de teste em que os testes ocorrerão. Além disso, deve-se identificar também ferramentas e recursos necessários para realizar o teste.

2. **Identificar critérios de aceitação de desempenho:** Identificar qual o tempo de resposta esperado, assim como vazão, ou utilização de recursos. Deve-se destacar o que se deseja observar com os testes.
3. **Planejar os testes:** Identificar os principais cenários (conjunto de elementos que compõem o ambiente de teste), determinar variabilidade entre os usuários e como simular essa variabilidade, definir dados de teste (possíveis entradas do programa), estabelecer métricas a serem coletadas, e definir quais testes serão executados de fato.
4. **Configurar o ambiente de teste:** Deve-se preparar o ambiente de teste, ferramentas e recursos necessários para a execução.
5. **Implementar o planejamento de testes:** Deve-se desenvolver os testes de acordo com o planejamento realizado.
6. **Executar os testes:** Deve-se executar os testes e coletar os resultados.
7. **Analisar e apresentar resultados:** Os dados devem ser analisados, para então serem apresentados.

3.6 Mapas Conceituais

A fim de destacar os termos e métodos de cada uma das áreas envolvidas neste projeto, e então compreendê-las melhor, foram construídos neste trabalho mapas conceituais para cada uma dessas áreas.

Um Mapa Conceitual é uma representação gráfica de um conjunto de conceitos construídos de tal forma que as relações entre eles sejam evidentes, mostrando como cada um dos conceitos relaciona-se com outro (TAVARES, 2007). Essa relação entre conceitos é feita geralmente através de verbos. Para um melhor entendimento, a leitura dos mapas conceituais deve começar pelo conceito central (que no caso dos mapas propostos neste trabalho, são os nomes das áreas de desempenho), seguindo as flechas e formando assim linhas de pensamento entre os conceitos do mapa e o verbo que liga eles.

Para mostrar quais conceitos pertencem a mais de uma área, os mapas são mostrados coloridos, e cada cor representa uma determinada área. A cor vermelha representa a área de Avaliação de Desempenho, a cor verde representa a área de Engenharia de Desempenho de Software e a cor azul representa a área de Teste de Desempenho de Software. Desse modo, a combinação dessas cores mostra que o conceito, no mapa, pertence também a outra área. A cor amarela mostra que o conceito pertence ao mesmo tempo a área de AD e EDS, já a cor magenta mostra que o conceito pertence a área de AD e TDS, e a cor verde água mostra que o determinado conceito pertence a área de EDS e TDS. E, por fim, a cor preta mostra que um conceito pertence as três áreas ao mesmo tempo.

Figura 6 – Mapa Conceitual para Avaliação de Desempenho

Na Figura 6 pode ser observado o mapa conceitual construído para a área de Avaliação de Desempenho. No mapa é possível observar, a partir do conceito do nome da área (neste caso, avaliação de desempenho), que o mapa se divide em três conceitos que são as três etapas principais para se realizar uma avaliação de desempenho: Planejamento de Experimentos, Técnicas de Avaliação e Análise de Resultados.

No planejamento de experimentos são definidos os fatores, replicação e o experimento em si, além de determinar a carga de trabalho. As técnicas de avaliação são divididas em modelagem, que envolve desenvolver modelos (utilizando Redes de Petri ou Redes de Filas) e solucioná-los através de resolução analítica ou por simulação, e também as técnicas de aferição, podendo ser por coleta de dados, *benchmarks* ou protótipos. Por fim, a análise dos resultados considera as medidas de desempenho, como as medidas de posição (como a média) e as medidas de dispersão (como o desvio padrão). Além disso, a Análise de Resultados também analisa a variável de resposta, a partir de métricas.

Na Figura 7 é apresentado o mapa conceitual para a área de Engenharia de Desempenho de Software. Partindo do conceito do nome da área, Engenharia de Desempenho de Software, é possível observar que a EDS considera requisitos, custos, operações e estrutura do sistema. Também determina quais métricas de desempenho devem ser levadas em consideração. Além disso, ela se utiliza de ferramentas como monitores e geradores de carga para se fazer uma análise de desempenho durante a criação do software.

Na Figura 7 é possível observar que os processos da EDS são: definir requisitos, identificar gargalos, prever desempenho, realizar teste de desempenho, fazer a manutenção do sistema e também uma análise completa do sistema, quando este estiver pronto. Também que a EDS se utiliza de carga de trabalho, do modelo de software, do ambiente de execução desse futuro

Figura 7 – Mapa Conceitual para Engenharia de Desempenho de Software

software, e dos recursos necessários para a criação de sistemas.

Por fim, são especificados no mapa da EDS os métodos que a área utiliza, os quais são divididos entre baseados em medição ou aferição e baseados em modelos. Os métodos baseados em modelos envolvem construir um modelo, testar o desempenho desse modelo e depois implementar um software a partir desse modelo.

A Figura 8 apresenta o mapa conceitual para a área de Teste de Desempenho de Software. Partindo do conceito que contém o nome da área, Teste de Desempenho de Software, é possível perceber que o TDS pode determinar o tempo de resposta, vazão, estabilidade ou mesmo a escalabilidade, entre outros, impondo aos sistemas testados uma carga de trabalho. Além disso, o TDS tem como objetivos encontrar problemas de desempenho em software, e assim otimizar tais sistemas, ou descobrir suas características de desempenho.

Para que o TDS cumpra com seus objetivos é preciso identificar o ambiente em que o teste executará, as ferramentas e os recursos necessários. No TDS são determinadas as métricas que devem ser definidas através do projeto dos testes, que também determina a variabilidade dos usuários, cenários de teste e os dados de teste. Após isso, deve-se analisar os resultados obtidos.

Por fim, o mapa mostra os tipos de teste que podem ser realizados, como o teste de carga, teste de estresse, teste de capacidade, teste de validação, teste de *spike* e teste de *smoke*.

RELACIONAMENTO ENTRE AS ÁREAS

4.1 Considerações Iniciais

Este capítulo apresenta como as áreas anteriormente descritas se relacionam. É importante ressaltar novamente que o objetivo deste trabalho não é mostrar cada um dos relacionamentos existentes, ou seja, todos os termos e métodos que são comuns a essas áreas; mas mostrar que elas, de fato, se relacionam. Os relacionamentos mostrados neste capítulo são suficientes para então propor uma metodologia que reúna conceitos de cada uma das áreas.

Para isso, uma lista foi construída contendo conceitos e definições que aparecem na bibliografia dessas áreas. Tal lista foi posteriormente validada com ajuda de professores que trabalham com desempenho ou teste de software. A partir dessa lista, foi construída uma ontologia para formalizar os relacionamentos.

4.2 Lista de Conceitos e Definições

Para construir a lista de conceitos e definições, a bibliografia de cada uma das áreas foi pesquisada, visando encontrar as definições de seus termos e métodos. O objetivo de construir tal lista foi observar e comparar cada uma das definições para descobrir se conceitos de uma área têm o mesmo sentido em outras áreas, i.e., se existem conceitos iguais com nomes diferentes ou conceitos diferentes com o mesmo nome. Os conceitos escolhidos para compor a lista estão presentes nos mapas conceituais construídos em cada área.

Cada linha da lista contém quatro campos: o nome do conceito, sua definição na área de Avaliação de Desempenho, caso exista, sua definição na área de Engenharia de Desempenho de Software, caso exista, e por fim sua definição na área de Teste de Desempenho de Software, caso exista. A fim de exemplificar a estrutura da lista, na tabela 2 pode ser observada a entrada para o conceito "Carga de Trabalho".

Caso o conceito não exista para uma determinada área ou não foi encontrado seu significado, o campo de definição é deixado em branco. A lista completa pode ser encontrada no Anexo A. As referências especificadas em cada entrada da tabela referem-se à bibliografia onde o conceito foi obtido.

Tabela 2 – Exemplo de uma entrada na lista de conceitos

Conceito	Definição		
	Avaliação de Desempenho	Engenharia de Desempenho de Software	Teste de Desempenho de Software
Carga de Trabalho	Requisições feitas por usuários do sistema.	Especifica a utilização prevista para o sistema.	Estímulo aplicado ao sistema para simular um padrão de uso

Alguns conceitos que foram encontrados, mas que não pertencem exclusivamente a estas áreas de desempenho, propositalmente não foram considerados. Esses são conceitos muito amplos e que, além das áreas anteriormente citadas, outras também os usam, como por exemplo o conceito de "Medida".

Com tais definições é possível observar quais são as mesmas para cada conceito ou que são diferentes. Também foram observados conceitos que, embora sejam diferentes, têm algum tipo de relação, o que será apresentado posteriormente.

Na Tabela 3 são listados os conceitos que têm o mesmo significado em mais de uma área e quais áreas são essas.

É importante observar que algumas vezes as definições dos conceitos usam as mesmas palavras para descrevê-los. Porém, outras vezes não, e nestes casos foi observado o significado daquela definição para decidir se o conceito é o mesmo ou não.

A seguir são listados os conceitos que não têm o mesmo significado para as áreas em questão:

- Cenários
- Métodos baseados na solução por aferição
- Métodos baseados na solução por modelagem
- Protótipos
- Sistema
- Implementação do software
- Projeto

As diferenças nos significados desses conceitos podem ser explicadas pela diferença das áreas. Por exemplo, quando se trata dos conceitos "Métodos baseados na solução por aferição",

Tabela 3 – Tabela com os conceitos que têm a mesma definição

Concept	Areas		
	ADS	EDS	TDS
Ambiente de Execução	X	X	X
Benchmarks	X	X	X
Carga de Trabalho	X	X	X
Capacidade do Sistema	X	X	X
Confiabilidade	X	X	X
Disponibilidade	X	X	X
Escalabilidade	X	X	X
Estabilidade		X	X
Ferramentas de Desempenho	X	X	X
Gargalos	X		X
Geradores de Carga	X	X	X
Métricas de Desempenho	X	X	X
Modelos Analíticos	X	X	
Monitores	X	X	X
Recursos Disponíveis	X	X	X
Redes de Filas	X	X	
Responsividade	X	X	X
Simulação	X	X	
Tempo de Resposta	X	X	X
Utilização de Recursos	X	X	X
Vazão	X	X	X
Análise de Resultados	X		X
Desenvolvimento de Modelos de Desempenho	X	X	
Planejamento e Gerenciamento de Capacidade	X	X	
Solução do Modelo	X	X	

"Métodos baseados na solução por modelagem", "Implementação do software" e "Projeto", para a área de Avaliação de Desempenho, referem-se a processos de se avaliar um sistema, sendo mais especificamente a implementação se referindo ao desenvolvimento de programas necessários para avaliação e o projeto ao planejamento de experimentos para a avaliação. Já esses mesmos conceitos, em Engenharia de Desempenho de Software, se referem à construção (projeto e implementação) de softwares.

Por fim, uma última análise que pode ser feita a respeito da lista é em relação aos conceitos que embora não sejam os mesmos, têm alguma relação em seu significado. São eles:

- Experimentos (Avaliação de Desempenho) e Sessão de Teste (Teste de Desempenho) - Um teste pode ser encarado como um experimento em um software com um propósito específico.
- Métodos baseados na solução por aferição (Avaliação de Desempenho e Engenharia de Desempenho de Software) - Embora este conceito tenha definições diferentes em AD e

EDS, a relação ocorre nas definições dessas duas áreas por estas serem muito parecidas. Na verdade, independentemente do objetivo de tais métodos, eles necessitam do sistema pronto para se analisar o desempenho, seja para uma avaliação final ou ainda no seu processo de construção.

- Métodos baseados na solução por modelagem (Avaliação de Desempenho e Engenharia de Desempenho de Software) - Mesmo caso do conceito anterior, as definições diferentes referem-se a ter que construir um modelo para analisar seu desempenho, com a finalidade de obter uma avaliação final ou como parte do processo de construção.
- Requisitos de Desempenho (Engenharia de Desempenho de Software) e Requisitos de Teste (Teste de Desempenho) - Embora tenham definições diferentes, esses dois conceitos tratam de requisitos, que para a EDS refere-se à implementação do software e para TDS um elemento a ser coberto pelo teste.
- Planejamento de Experimentos (Avaliação de Desempenho) e Planejamento de Teste (Teste de Desempenho de Software) - Esses dois conceitos se referem a etapas de planejamento, ou seja de decidir como o experimento ou teste será executado.

Com a lista completa foi possível observar que as áreas de desempenho estudadas de fato têm intersecções umas com as outras, tratando muitas vezes dos mesmos conceitos. Porém, uma ameaça à validade desta conclusão seria caso a lista tivesse erros em suas definições. A fim de diminuir o impacto deste problema, com o intuito de evitá-lo, a próxima sessão apresenta como foi feita a validação desta etapa do projeto.

4.3 Validação da Lista de Conceitos

A validação da lista de conceitos ocorreu de duas formas: a primeira foi observar se a bibliografia seguida, para a definição dos conceitos, representa, de fato, a área em questão para poder extrair as definições necessárias; a segunda forma, e mais decisiva, foi passar por especialistas (professores) que trabalham nas áreas em questão, ou têm conhecimento necessário para validar as definições propostas.

A bibliografia utilizada para extrair as definições foi composta de livros e artigos publicados em congressos e periódicos, como pode ser observado nas referências da lista completa apresentada no Anexo A. A fim de exemplificar, esta seção apresenta a descrição de três livros, um de cada área, como referências.

O livro de Jain (1991) serve como base para a área de Avaliação de Desempenho. Ele descreve as etapas a serem seguidas, como proceder com o planejamento de experimentos, tipos de avaliação, análise de resultados e cálculos estatísticos que podem ser feitos com os dados obtidos.

Já o livro de Bondi (2014) apresenta os conceitos da Engenharia de Desempenho de Software. Nele são citados métodos para a construção de sistemas contendo métricas de desempenho, requisitos de desempenho, análises de desempenho e caracterização da carga de trabalho.

Por fim, para a área de TDS, usou-se como base o livro de Meier *et al.* (2007). Apesar dele tratar teste de desempenho para aplicações web, nele há conceitos gerais sobre a área. Tais conceitos abrangem desde tipos de teste que existem, até etapas para a realização de um teste de desempenho.

A segunda etapa foi a validação da lista por especialistas. Por questões de tempo e disponibilidade de colaboradores, foram consultados apenas 4 especialistas em desempenho e teste, porém para este projeto mostrou-se o suficiente. Nessa etapa, foram consultados professores que observaram cada definição dos conceitos propostos, apontaram erros, melhorias e sugestões. Na tabela 4 são apresentados os especialistas que ajudaram na construção desta lista, onde trabalham e em qual área trabalham.

Tabela 4 – Especialistas do projeto

Nome	Instituição	Área
Prof ^ª . Dr ^ª . Sarita Mazzini Buschi	ICMC - USP	Avaliação de Desempenho
Prof. Dr. Paulo Sérgio Lopes de Souza	ICMC - USP	Avaliação de Desempenho / Teste de Software
Prof ^ª . Dr ^ª . Simone do Rocio Senger de Souza	ICMC - USP	Engenharia de Software / Teste de Software
Dr. Maicon Bernadino da Silveira	PUC/RS	Engenharia de Desempenho / Teste de Desempenho de Software

4.4 Ontologia Reunindo as Três Diferentes Áreas

A ontologia apresentada nesta seção visa à formalização das relações existentes entre as três áreas pesquisadas, ou seja, tem como foco os conceitos que são comuns a elas. Esta ontologia também é usada posteriormente como base para se propor a nova metodologia de análise de desempenho.

A ontologia foi construída com a ajuda da ferramenta Protégé. Seu código, em OWL (*Web Ontology Language*) pode ser obtido em um repositório *online*¹. Ela pode ser lida pela ferramenta Protegé ou por qualquer outra capaz de interpretar códigos em OWL. É importante ressaltar que, como a ontologia é baseada na lista, seu conteúdo não retrata todos os relacionamentos existentes entre as áreas estudadas. A metodologia seguida para a construção de ontologia foi a metodologia proposta em Noy e McGuinness (2001).

Esta ontologia, assim como a lista, poderia ser aprofundada dependendo do nível requerido. Para o objetivo deste projeto, os conceitos apresentados já mostram que as áreas têm relações umas com as outras. Portanto, tendo como base os objetivos deste trabalho, os conhecimentos representados nesta ontologia são considerados suficientes.

¹ Repositório disponível em www.lasdpc.icmc.usp.br/downloads - Performance Ontology

A ontologia desenvolvida é classificada como Semi-Informal, pois ela utiliza linguagem natural com algumas restrições para expressar seu domínio. Primeiramente, será apresentada a taxonomia, que contém as classes e a hierarquia formada por elas. Em seguida serão mostradas as propriedades de objetos e de dados da ontologia.

4.4.1 Taxonomia

A taxonomia parte da ontologia proposta para a formalização dos relacionamentos que existem entre as áreas. Ela focou em dividir os conceitos em classes que representam a intersecção entre as áreas de desempenho tratadas neste trabalho. A Figura 9 representa a taxonomia.

Figura 9 – Taxonomia relacionando as áreas de Avaliação de Desempenho, Engenharia de Desempenho de Software e Teste de Desempenho de Software.

Como é possível observar na Figura 9, a taxonomia contém 7 classes. Seguindo a hierarquia de baixo para cima, a primeira classe "*Common*" representa os conceitos que são comuns a todas as áreas tratadas. As classes acima dessa retratam as intersecções de conceitos de duas em duas áreas, sendo: "*Testing-Eng*", representando os conceitos em comum ao TDS e EDS; "*Testing-Evaluation*", representando os conceitos de TDS e AD; e por fim a classe "*Eng-Evaluation*", representando os conceitos em comum à EDS e AD.

Já as classes na hierarquia acima, representam os demais conceitos da lista que não têm relação com nenhuma outra área, ou seja, são conceitos que pertencem apenas à AD ou TDS ou EDS, respectivamente representados pelas classes *PE*, *SPT* e *SPE*.

Os objetos da ontologia são os conceitos, especificados anteriormente, pertencentes a cada uma dessas classes, podendo ser a só uma das áreas ou a mais de uma delas.

4.4.2 Propriedades da Ontologia

As propriedades de objeto desenvolvidas para esta ontologia são apresentadas a seguir:

- **Uses:** une a classe de SPE a SPT, indicando que a área de EDS usa o TDS como parte do ciclo de desenvolvimento do software.
- **Relates:** relaciona os objetos que têm definições diferentes, porém têm alguma relação.
- **Differs:** especifica os objetos que embora tenham o mesmo nome, têm definições diferentes em cada área.

E já as propriedades de dados desenvolvidas para ontologia foram:

- **hasconcept:** Uma propriedade das classes usada para armazenar os conceitos, sendo do tipo "string".
- **hasvalue:** Uma propriedade para objetos genérica para todos os objetos da ontologia para que possam ser valores. Por ser uma propriedade genérica, é do tipo "string".

4.4.3 Axiomas

Tendo definido a taxonomia e as propriedades da ontologia, é possível definir axiomas. Axiomas são sentenças verdadeiras geradas a partir das propriedades e objetos da ontologia e, no caso da ferramenta Protégé, são gerados automaticamente. Já os axiomas lógicos são parte dos axiomas usados especificamente para descrever os conceitos tratados pela ontologia.

Como exemplo de um axioma gerado, o qual se refere à hierarquia de classes, pode-se citar:

'#Common'SubClassOf'#Testing – Eng'

que especifica que a classe *'Common'* é uma subclasse de *'Testing – Eng'*. Outro exemplo que pode-se citar:

'#PE'DisjointClasses'#SPE'

que especifica que os elementos que pertencem a classe *'PE'* não pertencem a classe *'SPE'*. Essas relações são pré estabelecidas para qualquer ontologia e são formadas a partir da definição das classes da ontologia.

Como exemplo de axioma gerado que contém as propriedades definidas, pode-se citar:

'#Experiments'Relates'#Testsection'

que especifica que o objeto *'Experiments'* tem uma relação em sua definição com o objeto *'Testsection'*.

4.4.4 Comparação

Com o objetivo de se observar como a ontologia construída está estruturada em relação à outro trabalho, ela foi comparada com um dos trabalhos relacionados descritos anteriormente.

Como foi descrito anteriormente, não foram encontrados trabalhos que fizessem esse tipo de análise entre as áreas de AD, EDS e TDS, de modo a provar que elas têm intersecções, ou mesmo trabalhos que proponham ontologias dessas áreas juntas. Porém, o trabalho de [Freitas e Vieira \(2014\)](#) propõe uma ontologia para Teste de Desempenho de Software, que é uma das áreas tratadas neste trabalho e que disponibilizou o acesso à mesma.

Portanto, tal ontologia é a única que poderia ser comparada com a desenvolvida neste trabalho. Porém, é importante ressaltar que a ontologia de [Freitas e Vieira \(2014\)](#) tem como objetivo ser completa para a área de TDS e a deste trabalho não têm como objetivo ser completa em nenhuma área em que ela abrange, apenas mostrar que as áreas têm conceitos em comum.

Como comparação quantitativa das duas ontologias, a tabela 5 apresenta métricas extraídas do Protégé.

Tabela 5 – Métricas das Ontologias

Métrica	Ontologia proposta neste trabalho	Ontologia proposta por Freitas e Vieira (2014)
Axiomas	103	3520
Número de Axiomas Lógicos	43	3170
Classes	7	100
Propriedades de Objetos	3	44
Propriedades de Dados	2	9
Número de Indivíduos	63	171

De modo geral, considerando todas as métricas utilizadas para comparar as ontologias, percebe-se que os números são maiores na ontologia proposta por [Freitas e Vieira \(2014\)](#). Isso é porque, como já descrito anteriormente, ela tem como objetivo ser completa para a área de TDS, e detalhar cada classe sua. Já a ontologia apresentada neste trabalho trata de uma pequena parte de TDS, sendo apenas aqueles conceitos que se relacionam com outras áreas. A ontologia construída nesse trabalho apenas agrupa e diferencia os conceitos das áreas de AD, EDS e TDS. O mesmo poderia acontecer caso a comparação ocorresse com possíveis ontologias para EDS e AD, caso estas existam e não tenham sido encontradas no levantamento bibliográfico realizado neste trabalho.

É importante destacar que muitos dos conhecimentos, considerados neste trabalho, a respeito da área de TDS foram extraídos a partir do trabalho de [Freitas e Vieira \(2014\)](#), portanto alguns dos conceitos foram considerados por ambos os trabalhos.

4.5 Mapa Conceitual Unificando as Áreas

Para mostrar de forma gráfica quais conceitos as três áreas têm em comum, um outro mapa conceitual foi construído. Na Figura 10 é apresentado tal mapa que mostra quais conceitos pertencem a todas elas ou duas a duas (AD/EDS, EDS/TDS e EDS/TDS).

Além disso, a fim de obter uma melhor identificação de quais conceitos pertencem a mais de uma área, o mapa também segue o mesmo esquema de cores dos mapas anteriores. Então, a cor preta mostra que o conceito pertence às três áreas, a cor amarela mostra que o conceito pertence ao mesmo tempo à área de AD e EDS, já a cor magenta mostra que o conceito pertence à área de AD e TDS, e a cor verde água mostra que o determinado conceito pertence à área de EDS e TDS.

Figura 10 – Mapa Conceitual unificando os conceitos das três áreas

4.6 Considerações Finais

Este capítulo apresentou, primeiramente, a lista de conceitos e suas definições para as áreas de AD, EDS e TDS. A partir dessa lista foi possível obter conclusões a respeito de conceitos que pertencem a mais de uma área. Foram identificados também conceitos que embora tenham nomes iguais nas áreas, têm significados diferentes e ainda aqueles que embora sejam diferentes, suas definições são relacionadas. Também foi mostrada a ontologia que formaliza esse conhecimento do relacionamento existente entre as áreas, apresentando a taxonomia e componentes da ontologia.

Por fim, o mapa conceitual unificando as três áreas também foi apresentado.

O próximo capítulo apresenta uma proposta de metodologia de análise de desempenho a ser seguida, tendo como base a análise feita sobre os conceitos e definições de AD, EDS e TDS.

METODOLOGIA DE ANÁLISE DE DESEMPENHO

5.1 Considerações Iniciais

Neste capítulo é discutida primeiramente a proposta da metodologia de análise de desempenho e como ela foi elaborada. Em seguida é apresentada a aplicação da metodologia proposta na análise de desempenho da ferramenta ValiPar, em sua versão paralela, com o objetivo de identificar possíveis problemas com relação à sua escalabilidade. Para isso, cada etapa da metodologia foi instanciada para a análise da ferramenta.

A execução da ferramenta é detalhada de modo a justificar as métricas de desempenho extraídas dela, e por fim, são apresentados os resultados obtidos com a aplicação da metodologia na ValiPar.

5.2 Metodologia de Análise de Desempenho

A metodologia foi construída considerando o relacionamento existente entre as três áreas (AD, EDS e TDS), de modo que profissionais delas possam atingir seus objetivos. Dessa forma, foram agrupados métodos e termos das três áreas tratadas neste trabalho, que estão detalhadas nas seções 3.2, 3.4 e 3.5 desta dissertação, em uma única metodologia de análise de desempenho. O termo análise de desempenho foi escolhido para representar uma atividade mais abrangente que engloba as atividades de se avaliar e testar.

Entretanto, não foram considerados somente os conceitos que pertencem a todas as áreas de estudo. Foram usados também termos que pertenciam a duas áreas, ou somente uma, com o objetivo de deixar a metodologia mais completa. Quando esses termos aparecem, eles são explicados no detalhamento de cada passo da metodologia. Um exemplo de um termo que não

pertence a área de AD é "Requisitos", porém aparece na metodologia a fim de torná-la mais abrangente.

A metodologia para a análise de desempenho de sistemas computacionais proposta neste trabalho é composta dos seguintes passos:

1. Identificar o sistema e definir os objetivos

Esta etapa consiste em saber qual o sistema (*software* ou *hardware*) que será analisado e, dessa forma, listar os serviços oferecidos por ele. Após isso, deve-se definir quais os objetivos de se analisar o sistema, como por exemplo a identificação de gargalos ou mesmo comparação dos resultados de desempenho de diferentes experimentos realizados.

Resultado da etapa: Sistema a ser analisado; Objetivos da análise.

2. Enumerar os requisitos de desempenho

Deve-se enumerar os requisitos de desempenho que a análise deve cobrir, ou seja, o que os experimentos devem satisfazer para que os objetivos da etapa 1 sejam cumpridos. Como exemplo, pode-se julgar pertinente observar o tempo de resposta, comparar esse tempo com o de outros sistemas, ou mesmo observar a quantidade de dados processados em uma quantidade de tempo (vazão).

Resultado da etapa: Requisitos de desempenho da análise.

3. Determinar os parâmetros que afetam o desempenho do sistema

Faz-se uma lista com os parâmetros do sistema que afetam seu desempenho, tanto de *hardware* como de *software* já conhecidos. Alguns exemplos de tipos de parâmetros que podem afetar são aqueles relacionados à carga de trabalho, como a variabilidade de requisições feitas por usuários, ou mesmo problemas de *hardware*.

Resultado da etapa: Lista de parâmetros que afetam o desempenho.

4. Selecionar métricas de desempenho

A partir dos requisitos identificados na etapa anterior, juntamente com os parâmetros conhecidos que afetam o desempenho, as métricas do sistema que serão analisadas devem ser escolhidas. Tais Métricas devem ser comparáveis com os requisitos identificados. Por exemplo, se o requisito é comparar tempos de sistemas, a métrica escolhida deve ser o tempo de resposta.

Tais métricas podem ser: capacidade do sistema, confiabilidade, disponibilidade, eficiência, escalabilidade, estabilidade, responsividade, tempo de resposta, vazão, utilização de recursos, entre outras.

Resultado da etapa: Métricas de desempenho.

5. Determinar a técnica de avaliação apropriada

Nesta etapa se escolhe a técnica correta de avaliação, podendo ser aquelas baseadas na aferição ou na modelagem. Se for aferição, deve-se apontar que tipo de ferramentas serão utilizadas, como monitores ou *benchmarks*. Se for simulação, que tipo de simulador será utilizado ou se será por modelagem analítica.

Resultado da etapa: Técnica escolhida, apontando as respectivas ferramentas necessárias.

6. Determinar as variáveis de entrada e as replicações do experimento

Deve-se escolher as variáveis de entrada (fatores) dos experimentos, as quais devem representar o que se deseja analisar no sistema e com elas descobrir potenciais pontos de problemas de desempenho (gargalos), caso existam. Além disso, deve-se definir qual o valor dessas variáveis (níveis), e determinar a quantidade de replicação dos experimentos.

Resultado da etapa: Fatores, níveis e número de replicações.

7. Determinar as ferramentas de desempenho

Esta etapa consiste em identificar as ferramentas necessárias, apontadas na etapa 5 para a análise a ser feita, dependendo da escolha feita por essas ferramentas, caso não estejam disponíveis, é necessário desenvolvê-las. Para as técnicas baseadas em aferição, deve-se desenvolver monitores ou *benchmarks*. Já para as técnicas baseadas na simulação, deve-se desenvolver simuladores e modelos.

Resultado da etapa: Ferramentas e modelos prontos.

8. Determinar a carga de trabalho conforme o tipo de teste de desempenho apropriado

Devem ser escolhidos diversos tipos de cargas de trabalho que seguem os tipos de teste existentes, como a carga de trabalho característica (Teste de carga), ou a carga máxima suportada pelo sistema (Teste de capacidade), ou uma carga acima da suportada (Teste de estresse), ou uma carga que aumenta repentinamente (Teste de Spike).

Resultado da etapa: Especificação da Carga de Trabalho.

9. Identificar e configurar o ambiente físico

Deve-se identificar o ambiente físico do sistema que será analisado e deve-se também configurá-lo. Além disso, deve-se deixar todas as ferramentas e recursos preparados para a execução dos experimentos.

Resultado da etapa: Especificação do ambiente e sua configuração.

10. Implementar e executar os experimentos

Deve-se executar os experimentos planejados, usando as ferramentas anteriormente definidas, e então coletar os resultados das métricas especificadas.

Resultado da etapa: Conjuntos de dados de respostas dos experimentos.

11. Analisar os resultados e apresentá-los

A partir dos resultados obtidos, deve-se obter a média, e intervalo de confiança (caso necessário), para obter precisão estatística. Tais resultados devem ser interpretados e comparados com os requisitos especificados anteriormente. Caso algum resultado não tenha atingido os requisitos especificados, provavelmente há um problema de desempenho no sistema. Além disso, deve-se comparar os resultados dos diferentes cenários de cada experimento para descobrir, por exemplo, qual configuração do sistema teve o melhor desempenho. Caso algum objetivo da análise não tenha sido atingido, deve-se retornar à etapa 6 e planejar novamente os experimentos.

Resultado da etapa: Tabelas ou gráficos que mostram os resultados da análise feita.

Na Figura 11 é apresentado o diagrama que mostra a sequência de etapas da execução da metodologia proposta. Nela os quadros representam as etapas da metodologia e os circulares os resultados dessas etapas. Como é possível observar, as etapas 2 e 3 podem ser executadas paralelamente, já que os resultados dessas etapas não têm dependência um com o outro. O diagrama apresenta as dependências existentes com os resultados de etapas anteriores, mostra também que a metodologia é cíclica, ou seja, se os resultados atingidos não satisfizerem os requisitos de desempenho, deve-se voltar a uma das etapas iniciais de definições, dependendo da análise dos resultados.

5.3 Uso da Metodologia na ValiPar

Com o objetivo de usar a metodologia proposta neste trabalho e observar os resultados com ela obtidos, a mesma foi instanciada para a análise da ferramenta ValiPar, descrita na seção 2.3.

Como já descrito, a ferramenta foi desenvolvida inicialmente para ser executada de maneira sequencial. Porém, notou-se que o tempo para a execução dos pares de sincronização entre "sends" e "receives" gerados era muito grande, o que motivou a paralelização desta execução (BATISTA, 2015). A cada execução desses pares de sincronização é dado o nome de variante.

Figura 11 – Diagrama do fluxo de execução da metodologia

A partir da execução paralela das variantes de sincronização surgiu a necessidade de se analisar como a ferramenta se comportava com relação à sua escalabilidade. O objetivo de

tal análise é verificar o desempenho da ValiPar quando se aumenta o número de elementos de processamento para a execução das variantes, aumentando também a carga de trabalho.

Segundo Grama *et al.* (2003), a escalabilidade pode ser medida através da capacidade do software paralelo de variar proporcionalmente o seu *speedup* em relação à variação do número de elementos de processamento, ou manter a eficiência constante em uma mesma proporção, quando há aumento de carga de trabalho e de capacidade de processamento.

A escalabilidade também pode ser afetada por parâmetros que impedem a execução paralela do sistema. Por exemplo, no caso da ValiPar, sabe-se que o tamanho do arquivo de rastro gerado aumenta muito conforme aumenta a quantidade de variantes executadas e pelo tamanho do programa gerado. Tal aumento pode consumir todo o espaço em disco disponível, impossibilitando a execução.

Considerando isso, a metodologia foi instanciada com o objetivo de se analisar a escalabilidade da ferramenta ValiPar em sua versão paralela. Os resultados de cada etapa da metodologia apresentada neste trabalho são:

1. **Identificar o sistema e definir os objetivos:** O sistema a ser analisado é a versão paralela da ferramenta ValiPar. O objetivo da análise é observar o comportamento do desempenho da ferramenta em relação à sua escalabilidade.
2. **Enumerar os requisitos de desempenho:** Os requisitos para a análise são: observar o quanto a ferramenta é mais veloz ao se aumentar o número de elementos de processamento e a carga de trabalho imposta a ela; verificar quais parâmetros da própria ferramenta afetam sua escalabilidade, ou seja, se ela é, de fato, escalável.
3. **Determinar os parâmetros que afetam o desempenho do sistema:** O parâmetro escolhido que afeta o desempenho, em relação à escalabilidade foi o tamanho do arquivo de rastro gerado.
4. **Selecionar métricas de desempenho:** As métricas de desempenho escolhidas para a análise são: tempo de resposta, *Speedup*, eficiência e tamanho do arquivo de rastro. A métrica tamanho do arquivo de rastro foi escolhida considerando o parâmetro que afeta o desempenho tratado anteriormente. Com relação ao tempo de resposta, para uma melhor análise, foram observados dois tempos: um somente da execução de variantes, e o outro com o tempo total que a ferramenta demora para realizar o teste. Tais tempos são melhor descritos na seção 5.3.1.

Já o *Speedup* mede quantas vezes um algoritmo paralelo é mais rápido que sua versão sequencial. O *Speedup* é definido pelo tempo de execução sequencial dividido pelo tempo de execução do algoritmo em paralelo ($Sp = T1/Tn$).

A Eficiência é definida como sendo o tempo de execução em um processador dividido pelo tempo de execução em n processadores multiplicado por este número de processa-

dores ($E = T1/(n * Tn)$) ou simplesmente o valor do *speedup* dividido pelo número de processadores ($E = Sp/n$).

5. **Determinar a técnica de avaliação apropriada:** A técnica de avaliação apropriada para esta análise é a Aferição, já que existe acesso ao sistema, coletando os tempos de execução e tamanho do arquivo de rastro. Portanto, a técnica utilizada é a de Coleta de Dados. Porém, representar a carga imposta ao sistema foram utilizados *benchmarks*, a fim de mostrar as características de desempenho da ferramenta.

Os *benchmarks* usados na análise de desempenho foram escolhidos a partir do trabalho de [Dourado \(2015\)](#), que desenvolveu uma série de *benchmarks* para permitir a análise e comparação de diferentes ferramentas de teste de programas concorrentes. Os *benchmarks* considerados para essa análise foram escolhidos por apresentarem diferentes paradigmas de programação concorrente (passagem de mensagem, memória compartilhada e ambos paradigmas). Uma breve descrição dos *benchmarks* encontra-se a seguir:

- **Matrix - Ambos Paradigmas:** O *benchmark* implementa uma multiplicação de duas matrizes de ordem 2x2. O programa conta com um processo mestre e quatro processos escravos, e esses criam mais duas *threads* cada um.
- **RollerCoaster:** Este *benchmark* simula uma montanha-russa que possui um carro com N lugares e possui X passageiros a espera para entrar no carrinho.
- **Hilzers Barbershop:** Este *benchmark* é um programa que implementa uma barbearia com duas cadeiras, dois barbeiros, uma sala de espera e um sofá. Quando um barbeiro está livre, um cliente que esteja no sofá pode se sentar na cadeira do barbeiro liberando um assento no sofá. Um cliente que esteja na sala de espera pode se sentar no sofá no lugar que ficou vago. Quando o corte de cabelo for realizado, o cliente efetua o pagamento, porém o pagamento é aceito de apenas um cliente por vez.
- **Matrix - Memória Compartilhada:** Neste *benchmark* uma multiplicação de matrizes de ordem N é implementada. A quantidade de *threads* é definida por $N^3 + N^2 + 1$. Como o dado de teste usado foi N=2, obteve-se como saída o valor 13 *threads*.

6. **Determinar as variáveis de entrada e as replicações do experimento:** Na Tabela 6 são apresentados os fatores e os níveis escolhidos para a análise.

Os fatores foram escolhidos de modo a observar a escalabilidade, ou seja, aumentar do número de elementos de processamento aumentando também a carga de trabalho, representada pelos *benchmarks*.

Os níveis do fator "Número de elementos de processamento" foram escolhidos de modo a representar uma quantidade crescente de elementos para a análise de escalabilidade. O número máximo (64) foi limitado pela quantidade de máquinas virtuais disponíveis para os experimentos.

Tabela 6 – Tabela com Fatores e Níveis

Fatores	Níveis
Número de Elementos de Processamento	1
	2
	4
	8
	16
	32
	64
Benchmarks	Matrix - Memória Compartilhada RollerCoaster Hilzers Barbershop Matrix - Ambos Paradigmas

Já os *benchmarks* foram escolhidos de forma a representar diferentes tamanhos de cargas de trabalho. Suas configurações são detalhadas na tabela 7.

Tabela 7 – Configurações dos Benchmarks

Benchmarks	Nº de Processos	Nº de Threads	Nº de Arestas de Sincronização	Nº de Variantes	Paradigma do Benchmark
Matrix - Ambos Paradigmas	5	13	68	160	Ambos
RollerCoaster	6	6	132	356	Passagem de Mensagem
Hilzers Barbershop	1	7	208	1227	Memória Compartilhada
Matrix - Memória Compartilhada	1	13	192	2611	Memória Compartilhada

É importante destacar que a quantidade de variantes geradas em um programa depende não só do número de arestas de sincronização que o programa possui, mas também de sua característica. Dependendo de como o programa está implementado, ele pode gerar mais variantes quando é testado, como por exemplo, um programa que implementa comunicação em anel não deverá gerar muitas variantes já que a comunicação, neste caso, é limitada.

E, por fim, foi definida uma quantidade de 10 replicações para os experimentos, sendo esta a quantidade suficiente para ter uma precisão estatística dos resultados. Cada experimento foi composto pela combinação de todos os níveis de todos os fatores, totalizando 28 experimentos.

7. **Desenvolvimento de ferramentas de desempenho (se necessário):** Não foi necessário criar de nenhum *benchmark* a mais. Como apontado anteriormente, eles foram usados a partir do trabalho de [Dourado \(2015\)](#).
8. **Determinar a carga de trabalho conforme o tipo de teste de desempenho apropriado:** Foi definida uma carga de trabalho crescente de acordo com os *benchmarks* escolhidos

para o desenvolvimento. Nesse caso, foram escolhidos como base, o teste de carga e capacidade, pois o teste de carga é usado para verificar o comportamento da ferramenta e o teste de capacidade foi usado para verificar a escalabilidade.

9. **Identificar e configurar o ambiente físico:** O ambiente físico em que os experimentos foram executados foi o *cluster* "Halley" do Laboratório de Sistemas Distribuídos e Programação Concorrente do ICMC-USP. Esse *cluster* conta com 13 nós, sendo 1 nó mestre e 12 escravos. Cada nó conta ainda com 8 máquinas virtuais já instanciadas em que se pode realizar experimentos. As configurações dos nós físicos e das máquinas virtuais é descrita na Tabela 8.

Tabela 8 – Configurações das máquinas

Tipo de máquina	Especificações
Máquina Física	Processador: Intel Core I7 Memória: 32 GB RAM Disco: 2TB Video Nvidia GTX 650 Sistema Operacional: Linux Ubuntu 14.04 Rede: Gigabit Ethernet
Máquina Virtual	Virtualizador : KVM Processador: 4 núcleos Memória: 8 GM RAM Sistema Operacional: Linux Ubuntu 14.04

Para a preparação do ambiente, as máquinas virtuais necessárias para os experimentos foram deixadas ligadas. Além disso, para a execução foi criado um *script*, já que a execução completa da ValiPar é feita em etapas. Tal *script* será detalhado na próxima seção, no detalhamento da execução dos experimentos.

Os passos 10 e 11 são, respectivamente, executar de fato o experimento, coletar os dados e mostrar os resultados. Para uma melhor apresentação, os resultados e conclusões sobre o desempenho com relação à escalabilidade da ferramenta são mostrados na seção 5.3.2.

5.3.1 Detalhamento da Execução

Darch (*Distributed Architecture*) é o nome dado ao ambiente distribuído responsável pela execução paralela de variantes na ferramenta ValiPar (BATISTA, 2015). A Darch instancia e controla os módulos da ferramenta em cada máquina virtual. Os módulos da ferramenta são a ValiEval, ValiSync e diversas instâncias da ValiExec, além de uma instância de um orquestrador que coordena essas execuções. Na Figura 12 pode ser observado esse funcionamento.

De modo geral, a primeira execução do caso de teste é feita de maneira não determinística. A partir dela, é gerado o rastro da execução, e avaliada a cobertura que essa execução obteve.

Figura 12 – Diagrama da execução dos módulos da ValiPar (BATISTA, 2015)

Então, se foi atingida a cobertura requerida pelo testador, o teste pode ser encerrado. Caso contrário, são geradas variantes e essas são distribuídas para as instâncias da ValiExec, para serem executadas, retornando os rastros gerados e a cobertura é novamente calculada. Essas etapas são feitas para cada dado de teste que se deseja executar.

O paralelismo implementado por Batista (2015) é um paralelismo de dados, ou seja, cada execução é feita de forma independente em uma parte dos dados (variantes).

Para que a execução das variantes possa ocorrer, isso contando a ValiExec, ValiSync e ValiEval, ou seja a execução do teste no programa concorrente propriamente dito, é necessário que outras etapas da ValiPar executem também (ValiInst, ValiElem e criação de dados de teste). Estes porém não são contabilizados na execução do teste propriamente dito, por serem atividades que precisam ser feitas no início da sessão de teste apenas uma vez.

Como era necessária a execução de diversas replicações e diferentes quantidades de máquinas virtuais para os experimentos, foi criado um script para automatizá-los. No algoritmo 1 é apresentado como essa automatização foi realizada.

Como é possível perceber na linha 03 do algoritmo 1, foram criados diretórios para que uma execução de testes não influenciasse no resultado de outra execução. Já na execução da Darch, a parte paralela da ValiPar, alvo desta análise de desempenho, é importante destacar que independentemente da execução, sempre serão instanciadas apenas uma ValiSync e uma ValiEval. Já a quantidade de instâncias da ValiExec depende do grau de paralelismo que se deseja. Para os experimentos realizados, foram instanciados sempre o valor de $i + 2$ máquinas virtuais (i ValiExecs + 1 ValiSync + 1 ValiEval). Nos resultados estes são apresentados com o número de instâncias com o valor de i .

Além disso, existem dois outros parâmetros importantes que devem ser destacados. O

Algoritmo 1: Execução automática dos experimentos

```

1  inicio
2      para i em 1, 2, 4, 8, 16, 32 e 64 faça
3          Criar_diretório i
4          Entrar_no_diretório i
5          para j de 1 a 10 faça
6              Criar_diretório j
7              Entrar_no_diretório j
8              ValiInst Código_programa;
9              ValiElem;
10             ValiTestCase Dado_de_teste; Darch Instancia_Sync 1 Instancia_Eval 1
11             Instancia_Exec i Cobertura Profundidade
12         fim
13 fim

```

---

primeiro é a cobertura requerida de arestas de sincronização, ou seja, o testador pode escolher o quanto de cobertura das arestas ele deseja cobrir para que o teste pare. Para os experimentos da análise de desempenho feita, foi escolhido um total de 100% de cobertura. O outro parâmetro é o de profundidade, que se refere à profundidade da árvore de geração de variantes. A árvore é construída a partir do algoritmo de geração proposto em Batista (2015), que faz uma combinação entre variantes formando cada nível da árvore. Para os experimentos da análise feita neste trabalho a profundidade foi de 2.

Já com relação às variáveis de resposta, foram colhidos os tempos de execução de variantes e o tempo de sessão de teste. Esse último engloba o tempo de execução de variantes, o tempo para gerar variantes e o tempo para calcular a cobertura que o teste atingiu. Na Figura 13 são mostradas de forma simplificada as etapas de execução. É possível observar que é formado um ciclo entre o cálculo de cobertura, a geração e execução de variantes.

5.3.2 Resultados

Serão mostrados primeiramente os tempos de execução de variantes e sessão de teste, para então serem apresentados os *speedups* e eficiências obtidas a partir desses tempos.

Na Figura 14 são mostrados os tempos de execução de variantes para cada um dos *benchmarks* executados. Os intervalos de confiança foram calculados com uma confiança de 95%.

É possível observar que em todos os gráficos da Figura 14 ocorre uma diminuição no tempo de execução de variantes conforme se aumenta o número de máquinas virtuais, o que pode ser observado pelas médias e intervalos de confiança que não se sobrepõem em nenhum dos resultados. Essa diminuição chega a 97,8%, 97,6%, 91,4% e 98,7% para 64 MVs, respectivamente para os *benchmarks* Matrix - Ambos paradigmas, RollerCoaster, Hizers Barbershop e Matrix -

Figura 13 – Diagrama da execução simplificada da Darch

Memória Compartilhada.

Uma melhor análise a respeito do desempenho será apresentada mais à frente, considerando os tempos de sessão de teste, os *speedups* e eficiências.

Já na Figura 15 são apresentados os tempos de sessão de teste para cada um dos *benchmarks* executados. Mais uma vez, os intervalos de confiança foram calculados com uma confiança de 95%.

É possível perceber que em todos os gráficos da Figura 15 também ocorre uma diminuição nos tempos mostrados conforme se aumenta o número de elementos de processamento. Porém, esses tempos, se comparados com os de execução de variantes, são maiores, pois contabilizam também os tempos de geração de variantes e análise de cobertura. Essa diminuição chegou a 97%, 97,3%, 93% e 87,3% para 64 MVs, respectivamente para os *benchmarks* *Matrix* - Ambos paradigmas, *RollerCoaster*, *Hizers Barbershop* e *Matrix* - Memória Compartilhada. Mais uma vez, os intervalos de confiança não se sobrepõem em nenhum dos resultados.

Além disso, é possível perceber nos tempos de sessão de teste que, embora esses tempos diminuam com o aumento do número de máquinas virtuais para a execução de variantes, esse tempo não é proporcional a esse aumento. Isso será comprovado com os cálculos de *speedup* e eficiência.

Os gráficos da Figura 16 mostram os resultados calculados de *speedup* tanto dos tempos

Figura 14 – Tempos da Execução de Variantes para os benchmarks (a) Matrix Ambos paradigmas (b) RollerCoaster (c) Hilzers Barbershop (d) Matrix Memória Compartilhada

de execução de variantes quanto para o de sessão de teste. O *speedup* foi calculado dividindo-se o tempo de execução em uma única máquina pelos demais tempos. Ou seja, o tempo de execução na Darch com uma única máquina virtual para execução de variantes foi definido como equivalente a versão sequencial da ferramenta. Os intervalos mostrados nos gráficos da Figura 16 foram calculados, com a fórmula do *speedup* ($Sp = T(1)/T(n)$) com os limites inferior e superior da média dos tempos.

Os gráficos apresentam também o *speedup* teórico e ideal a ser obtido com a paralelização. Tanto para o tempo de sessão de teste como para o tempo de execução de variantes, os melhores *speedups* foram com os *benchmarks* que tiveram menos variantes. Isso se deve ao comportamento dos *benchmarks* analisados, principalmente com o *benchmark Matrix - Ambos paradigmas* de comunicação (passagem de mensagem e memória compartilhada) e com o *benchmark RollerCoaster*. Estes *benchmarks* apresentam poucos elementos requeridos de arestas de sincronização, o que faz com que o desempenho não seja degradado com a avaliação da cobertura. Como o *benchmark RollerCoaster* contém um número de arestas de sincronização um pouco maior, o seu desempenho também foi um pouco menor.

Já o *benchmark Hilzers Barbershop* e *Matrix - Memória Compartilhada* apresentam muitas variantes a serem executadas, tendo um expressivo ganho de desempenho na execução

Figura 15 – Tempos de Sessão de Teste para os benchmarks (a) Matrix Ambos paradigmas (b) RollerCoaster (c) Hilzers Barbershop (d) Matrix Memória Compartilhada

de variantes, embora não tendo os melhores *speedups* entre os *benchmarks*. A razão pela qual o desempenho desses *benchmarks* teve uma degradação foi a grande quantidade de arestas de sincronização que precisavam ter calculadas as suas coberturas.

Especificamente para os *speedups* do tempo de execução de variantes, embora o *benchmark Matrix - Ambos paradigmas* foi melhor que *Matrix - Memória compartilhada*, essa diferença foi muito pouca, apenas de 18,9%.

Percebe-se ainda que os *speedups* do tempo de execução de variantes são maiores que o de sessão de teste. Isso porque o tempo de sessão de teste considera execuções que são sequenciais, diminuindo assim o *speedup*. Isso será comprovado com os resultados das eficiências mostradas nos gráficos da Figura 17.

As eficiências confirmam as análises feitas a partir dos *speedups*. Porém, alguns pontos merecem destaque. As linhas da eficiência dos *benchmarks Matrix - Ambos paradigmas* e *RollerCoaster* se cruzam, tanto para as eficiências do tempo de execução de variantes quando do tempo de sessão de teste, isso se deve ao fato das quantidades de variantes entre eles serem muito próximas, diferentes dos demais.

Existe também uma grande diferença entre as eficiências dos *benchmarks Hilzers Bar-*

(a)

(b)

Figura 16 – Speedup para (a) Tempo de Execução de Variantes e (b) Tempo de Sessão de Teste

bershop e *Matrix* - Memória Compartilhada para os tempos de sessão de teste, o que evidencia o quanto a quantidade de arestas a serem cobertas influencia no desempenho.

De modo geral, as eficiências dos tempos de execução de variantes e de sessão de teste decaem conforme aumenta-se o número de máquinas virtuais. Porém, essa diminuição nos valores da eficiência para execução de variantes ocorre de maneira mais suave, tanto que no pior caso nenhuma eficiência ficou abaixo de 0.6. A eficiência para execução de variantes tende a se manter com um maior número de elementos de processamento. O mesmo não ocorre com as eficiências dos tempos de sessão de teste, essas têm um decréscimo muito acentuado, por exemplo, o *benchmark Hilzers Barbershop* começa sua eficiência com 0,7 para 2 máquinas virtuais e termina com 0,2 para 64 máquinas virtuais.

Dessa forma, os experimentos realizados confirmam a Lei de Amdahl, que diz que o desempenho de um programa paralelo é limitado por sua porção sequencial. Fica evidenciado que o cálculo de cobertura e geração de variantes, partes sequenciais da ferramenta, determinam seu desempenho final. Com relação à escalabilidade, é possível afirmar, que a parte executada em paralelo, ou seja, a execução de variantes pode ser considerada escalável, porém o restante da ferramenta não.

(a)

(b)

Figura 17 – Eficiência para (a) Tempo de Execução de Variantes e (b) Tempo de Sessão de Teste

Com relação ao tamanho do arquivo de rastro, os gráficos com os resultados podem ser visualizados na Figura 18, com a média e o intervalo de confiança do tamanho dos arquivos considerando as replicações. É possível observar que conforme se tem mais variantes, maior é o valor do arquivo de rastro gerado. Os *benchmarks* *Matrix - Ambos* Paradigmas, *RollerCoaster* e *Hilzers Barbershop* não apresentam variação no tamanho de arquivo de rastro conforme a quantidade de máquinas virtuais que estão executando aumenta.

Já no *Benchmark Matrix - Memória* compartilhada, que apresenta uma maior quantidade de variantes comparado aos demais *benchmarks*, é possível observar um pequeno aumento no tamanho do arquivo, embora os resultados comparados de forma pareada sejam estatisticamente equivalentes. Esse pequeno aumento se deve provavelmente ao fato do rastro também armazenar onde a execução é feita, porém isso só fica evidente com uma quantidade grande de variantes.

Na Figura 19 é possível observar o gráfico que contém o tamanho total gerado nos experimentos realizados em cada *benchmark*. Como é possível observar, mesmo o tamanho total não é suficiente para ocupar todo o disco, que nos experimentos tem 2TB, porém se a ferramenta for utilizada por muitos usuários, possivelmente em algum momento o disco ficará totalmente ocupado e não será possível a execução da ferramenta, mostrando assim uma limitação com

Figura 18 – Tamanho do arquivo de rastro por execução em MB

Figura 19 – Tamanho do arquivo de rastro total em MB

relação à escalabilidade da mesma.

Por fim, um outro ponto que deve ser mencionado, o qual ocorreu durante os experimentos realizados, foi que durante a execução da ferramenta com 32 ou 64 máquinas virtuais, houve algumas vezes um travamento na execução, evidenciando assim um possível problema de implementação, fato que afeta a escalabilidade do programa, já que aumentando-se o número de máquinas virtuais, poderia ocorrer um travamento. Para solucionar isso e obter os resultados para esta análise, os experimentos foram replicados mais vezes para esses níveis do experimento.

Ao se comparar os requisitos de desempenho definidos com os resultados obtidos foi possível observar que o desempenho da ValiPar melhora quando se aumenta o número de elementos de processamento. Além disso, foi possível evidenciar pontos em que se reduz a escalabilidade da ferramenta, como os pontos que a ferramenta não é paralelizada, o tamanho do arquivo de rastro gerado e os travamentos causados quando se aumenta o número de máquinas

virtuais.

5.4 Análise e Comparação com Outras Metodologias

A fim de comparar a metodologia proposta neste trabalho e mostrar sua vantagem em relação as demais, ou seja, a importância de uma análise de desempenho com uma formalização em seus conceitos, foi instanciada a análise da ferramenta ValiPar também para as metodologias de Avaliação de Desempenho e Teste de Desempenho de Software, mostradas nas seções 3.3.1 e 3.5.1, respectivamente.

Seguindo a metodologia de Avaliação de Desempenho, tem-se os seguintes resultados em cada uma de suas etapas:

1. O sistema a ser analisado é a versão paralela da ferramenta ValiPar. O objetivo da análise é observar o comportamento do desempenho da ferramenta em relação à sua escalabilidade.
2. O serviço oferecido pelo sistema é o teste de programas concorrentes, onde se pode especificar a quantidade de instâncias para a execução de variantes.
3. As métricas de desempenho escolhidas para a análise são: Tempo de resposta, *Speedup* e Eficiência.
4. Os parâmetros conhecidos que afetam o desempenho são: tamanho do arquivo de rastro gerado; Possíveis defeitos no hardware, como problemas de energia.
5. Os fatores são: número de elementos de processamento e *benchmarks* utilizados. Os níveis para número de elementos de processamento são 1, 2, 4, 8, 16, 32 e 64 e para *benchmarks* são *Matrix* - Ambos Paradigmas, *RollerCoaster*, *Hilzers Barbershop* e *Matrix* - Memória Compartilhada, escolhidos a partir do trabalho de [Dourado \(2015\)](#).
6. A técnica de avaliação escolhida é a aferição por meio de *benchmarks*, já que se tem acesso ao sistema.
7. A carga de trabalho é considerada um fator, representada pelos *benchmarks*. Não sendo escolhida, portanto uma carga que representa as características usuais do sistema.
8. O Planejamento de experimentos escolhido é o Fatorial Completo.

As etapas 9 e 10 representam a análise dos dados obtidos com os experimentos e sua apresentação, que não serão mostrados já que correspondem à seção 5.3.2 desta dissertação.

Seguindo a metodologia de Teste de Desempenho de Software, tem-se os seguintes resultados em cada uma de suas etapas:

1. O ambiente em que se executa a ferramenta é o *cluster "Halley"* do Laboratório de Sistemas Distribuídos e Programação Concorrente do ICMC-USP. As configurações dos nós físicos e das máquinas virtuais do *cluster* são descritas na Tabela 8.
2. Os requisitos para o teste são: verificar o quanto tempo a ferramenta é mais rápida aumentando o número de elementos de processamento e a carga de trabalho imposta a ela; Verificar quais parâmetros da própria ferramenta que afetam sua escalabilidade, ou seja, se ela é de fato escalável.
3. O planejamento de teste constitui em escolher *benchmarks*, representando a carga de trabalho imposta ao sistema, de modo a implementar o teste de carga e capacidade. Além disso, o número de elementos de processamento deve aumentar para se obter conclusões a respeito da escalabilidade. O número de elementos de processamento são 1, 2, 4, 8, 16, 32 e 64 e os *benchmarks* são *Matrix* - Ambos Paradigmas, *RollerCoaster*, *Hilzers Barbershop* e *Matrix* - Memória.
4. A preparação do ambiente, consiste em deixar as máquinas virtuais necessárias para os experimentos ligadas. Além disso, para a execução, foi criado um *script*, já que a execução completa da ValiPar é feita em etapas.
5. Os testes implementados são Teste de Carga e Teste de Capacidade.

As etapas 6 e 7 constituem em implementar e executar os testes e analisar os resultados. Os resultados já foram mostrados, com a análise proposta neste trabalho, por isso não serão apresentados novamente.

Como é possível observar, os resultados da aplicação isolada das metodologias de Avaliação de Desempenho e Teste de Desempenho de Software poderiam ser os mesmos com os obtidos com a metodologia proposta neste trabalho. Porém, as etapas que constituem essas metodologias muitas vezes se tornam confusas, ou faltam formalismos, o que poderia afetar os resultados finais.

Comparando especificamente a metodologia de Avaliação de Desempenho com a proposta neste trabalho, três pontos principais podem ser destacados: o primeiro consiste em destacar formalismos necessários para uma análise de desempenho, destacando requisitos possíveis de serem comparados com o resultado final obtido com os experimentos; o segundo constitui em formalizar a escolha da carga de trabalho, de modo que não se escolha somente uma carga de trabalho normal ao sistema, mas diversas para representar as possibilidades de carga recebida; por fim o terceiro ponto é a escolha das métricas, que deveria considerar os parâmetros que afetam o desempenho e requisitos, no caso de Avaliação de Desempenho, o parâmetros são listados depois que se escolhe as métricas.

Já a metodologia de Teste de Desempenho de software pode-se destacar que falta detalhar a etapas do planejamento de testes, facilitando para o testador. Além disso, falta também tratar

parâmetros que afetam o desempenho e uma escolha mais detalhada de métricas a serem analisadas.

Pode-se destacar também que a metodologia proposta é cíclica, ou seja, caso os objetivos da análise não sejam atingidos, deve-se retornar ao planejamento. Desse modo é possível destacar as vantagens de se utilizar a metodologia proposta, para avaliadores de desempenho, testadores de desempenho e engenheiros de desempenho de software.

5.5 Considerações Finais

Esse capítulo apresentou a metodologia de análise de desempenho proposta, com cada uma das suas etapas, e o que deve ser produzido em cada uma delas. Essa metodologia foi produzida a partir de termos e metodologias de áreas que possuem o desempenho como ponto central.

A partir dessa metodologia, foi analisada a ferramenta ValiPar em sua versão paralela, de modo a observar principalmente sua escalabilidade e descobrir quais parâmetros da própria ferramenta que afetam seu desempenho, produzindo assim diversos resultados.

Os resultados mostraram que o principal gargalo da ferramenta com relação à sua escalabilidade são suas porções sequenciais, que fazem o desempenho degradar, sendo possível observar através do gráfico que mostra a eficiência obtida nos experimentos. Além disso, foi comprovado que os arquivos de rastro gerados de fato podem consumir todo o espaço em disco e foi possível evidenciar problemas na implementação, o que causou travamento da ferramenta com 32 e 64 elementos de processamento.

Por fim, foi apresentada uma comparação e análise com as metodologias utilizadas nas áreas de Avaliação de Desempenho e Teste de Desempenho de Software, apresentando as vantagens de se utilizar a metodologia proposta e que esta pode ser utilizada por profissionais das áreas de desempenho tratadas neste trabalho.

CONCLUSÕES

6.1 Conclusões do Trabalho

O desempenho é de fundamental importância para sistemas computacionais. Isso pode ser comprovado, na área de Ciência da Computação, com o desenvolvimento de diversas áreas de estudo de desempenho. Algumas dessas áreas são: Avaliação de Desempenho (AD), Engenharia de Desempenho de Software (EDS) e Teste de Desempenho de Software (TDS).

Embora tais áreas tratem o desempenho como ponto central, elas têm diferentes objetivos e com isso vocabulários diferentes. Porém, muitos de seus conceitos são exatamente os mesmos, sendo apenas tratados de formas diferentes. Outro ponto que surge nessa diferença é que as metodologias de uma determinada área podem não ser completamente compreendidas por um profissional de uma outra área.

Dessa forma, nesse trabalho foi mostrado que as áreas de AD, EDS e TDS se relacionam, e dessa forma apresentam conceitos que são comuns a todas as áreas ou pelo menos duas áreas. Foi também proposta uma metodologia de análise de desempenho que contivesse conceitos dessas três áreas.

Para mostrar como as áreas se relacionam, primeiramente foi feito um estudo de cada uma das áreas e construídos mapas conceituais para cada uma delas. A partir de então uma lista com conceitos e suas definições também foi criada e então pode-se observar, a partir do significado dos conceitos, quais deles pertenciam a mais de uma área e quais deles têm nomes iguais, porém com definições diferentes em áreas diferentes.

Para formalizar os relacionamentos existentes entre as áreas, uma ontologia que contém esses conceitos e que mostra quais deles pertencem a mais de uma área, foi criada. A partir de então foi mostrado que as áreas de estudo de desempenho da Ciência da Computação se correlacionam.

Em seguida, uma metodologia de análise de desempenho foi proposta. Ela foi usada para analisar a ferramenta ValiPar e observar seu desempenho, principalmente com relação a sua escalabilidade. Os resultados mostraram que a porção sequencial da ferramenta é determinante para que ela não seja escalável, mais ainda que existe mais um fator dela que limita sua execução que é o tamanho do arquivo de rastro gerado, que pode preencher todo o disco. Outro defeito descoberto é que, com muitos elementos de processamento, a ferramenta pode travar, revelando um defeito de implementação, possivelmente de sincronismo.

Comparando a metodologia proposta com as de Avaliação de Desempenho e de Teste de Desempenho de Software, foram percebidas algumas vantagens em se utilizar a metodologia proposta neste trabalho, tais como uma abrangência maior e etapas mais detalhadas.

Concluimos, assim, que as áreas que estudam desempenho tratam de muitos conceitos que são os mesmos. É possível concluir também que a utilização da metodologia proposta pode ser abrangente o suficiente para cumprir os passos das metodologias das outras áreas.

6.2 Principais Contribuições

As principais contribuições científicas que este trabalho apresenta são primeiramente mostrar que as áreas de AD, EDS e TDS se relacionam, mostrando inclusive como esse relacionamento ocorre.

A segunda contribuição é a proposta de uma metodologia de análise de desempenho que contém conceitos das três áreas tratadas neste trabalho, de modo que se possa analisar o desempenho, independente do objetivo que se queria, seja na quantificação de desempenho, ou verificar se o desempenho é o mesmo que foi requerido.

Como contribuições de ordem técnica, este trabalho implementou uma ontologia que formaliza o relacionamento existente entre as áreas de estudo de desempenho. Além disso também foi realizada uma análise de desempenho da ferramenta ValiPar para se obter algumas conclusões sobre sua escalabilidade e possíveis defeitos da ferramenta.

6.3 Dificuldades Encontradas

As principais dificuldades encontradas durante a condução do projeto são relacionadas a bibliografia, validação da proposta e execução dos experimentos. Em relação à bibliografia, houve dificuldade em definir o que seria uma bibliografia suficiente que representasse o conhecimento das áreas.

Depois de escolhida a bibliografia e definidos os mapas conceituais, houve dificuldade em encontrar e contar com a colaboração de especialistas das áreas tratadas no trabalho, de modo que fosse possível fazer uma melhor validação da proposta.

E por fim, uma última dificuldade foi a execução dos experimentos na ferramenta ValiPar, que demandou um esforço para que pudesse ser executada.

6.4 Publicações

Como resultados desse trabalho de mestrado foram publicados os seguintes artigos:

- STABILE, R. S.; BRUSCHI, S. M. ; SOUZA, P. S. L. . Relacionando Conceitos de Áreas de Estudo de Desempenho da Ciência da Computação. XIV *Workshop em Desempenho de Sistemas Computacionais e de Comunicação*, Recife-PE, 2015.
- STABILE, R. S.; BRUSCHI, S. M. ; SOUZA, P. S. L. . Unificando Conceitos de Desempenho de Diferentes Áreas da Ciência da Computação para a Análise de Sistemas Computacionais. VI Escola Regional de Alto Desempenho de São Paulo, São José do Rio Preto - SP, 2015.

Além desses dois trabalhos, será submetido mais um artigo para a conferência:

- *10th International Conference on Performance Evaluation Methodologies and Tools - ValueTools 2016*.

Essa conferência concentra-se em metodologias e modelagens de avaliação de desempenho e otimização de sistemas computacionais.

6.5 Trabalhos Futuros

Como trabalhos futuros a esse mestrado, pode-se citar os seguintes aprofundamentos de estudo:

Primeiramente, como este trabalho mostrou que as áreas correlacionam-se entre si, e foi mostrado uma série de conceitos que pertencem a mais de uma área, um outro estudo poderia aprofundar o que foi feito e mostrar todos os conceitos existentes que pertencem a mais de uma área, a partir de uma revisão sistemática de outros artigos e livros.

Uma segunda maneira de continuar este trabalho seria especificar a metodologia proposta para sistemas específicos, como por exemplo sistemas embarcados, já que a metodologia proposta é para qualquer sistema, ou seja ela é genérica. Nesse sentido, um estudo de métricas específicas para o sistema alvo deveria ser realizado, e também quais tipos de teste são comuns a esses sistemas.

Como o presente trabalho analisou exclusivamente a ferramenta ValiPar, como trabalho futuro propõe-se a utilização da metodologia para analisar outro tipo de sistema com a metodologia apresentada, a fim de mostrar também pontos em que a mesma poderia ser melhorada. Além disso, de maneira a facilitar a utilização da metodologia por outros usuários, pode-se criar um formulário que contenha os passos a serem seguidos, guiando os usuários na utilização.

E, uma última sugestão seria implementar um módulo que auxiliasse a análise de desempenho, para que ela seja feita de forma automática.

REFERÊNCIAS

- A, F. M.; GOMEZ-PEREZ, A.; JURISTO, N. Methodology: from ontological art to ontological engineering. In: *AAAI Spring Symposium Series*. [S.l.: s.n.], 1997. p. 33–40. Citado na página 6.
- BATISTA, R. N. **Otimizando o teste estrutural de programas concorrentes: uma abordagem determinística e paralela**. Dissertação (Mestrado) — Universidade de São Paulo, 2015. Citado 6 vezes nas páginas 15, 10, 44, 49, 50 e 51.
- BONDI, A. B. **Foundations of Software and System Performance Engineering**. [S.l.]: Addison Wesley, 2014. Citado 10 vezes nas páginas 15, 23, 35, 70, 71, 72, 73, 75, 76 e 77.
- BRUSCHI, S. M. **Extensão do ASiA para simulação de arquiteturas de computadores**. 1997. Citado 2 vezes nas páginas 17 e 18.
- CORTELLESSA, V. How far are we from the definition of a common software performance ontology? In: *ACM. Proceedings of the 5th international workshop on Software and performance*. [S.l.], 2005. p. 195–204. Citado 3 vezes nas páginas 12, 13 e 14.
- DOURADO, G. G. M. **Contribuindo para a avaliação do teste de programas concorrentes: uma abordagem usando benchmarks**. Dissertação (Mestrado) — Universidade de São Paulo, 2015. Citado 3 vezes nas páginas 47, 48 e 58.
- FORTIER, P.; MICHEL, H. **Computer systems performance evaluation and prediction**. [S.l.]: Digital Press, 2003. Citado 8 vezes nas páginas 15, 17, 18, 70, 73, 75, 76 e 77.
- FREITAS, A.; VIEIRA, R. An ontology for guiding performance testing. In: *IEEE COMPUTER SOCIETY. Proceedings of the 2014 IEEE/WIC/ACM International Joint Conferences on Web Intelligence (WI) and Intelligent Agent Technologies (IAT)-Volume 01*. [S.l.], 2014. p. 400–407. Citado 3 vezes nas páginas 11, 14 e 38.
- GARCÍA-PEÑALVO, F.; GARCÍA, J.; THERÓN, R. Analysis of the owl ontologies: A survey. **Scientific Research and Essays**, v. 6, n. 20, p. 4318–4329, 2011. Citado na página 7.
- GENNARI, J. H.; MUSEN, M. A.; FERGERSON, R. W.; GROSSO, W. E.; CRUBÉZY, M.; ERIKSSON, H.; NOY, N. F.; TU, S. W. The evolution of protégé: an environment for knowledge-based systems development. **International Journal of Human-computer studies**, Elsevier, v. 58, n. 1, p. 89–123, 2003. Citado na página 7.
- GRAMA, A.; KUMAR, V.; GUPTA, A.; KARYPIS, G. **Introduction to parallel computing**. [S.l.]: Pearson Education, 2003. Citado 2 vezes nas páginas 46 e 71.
- GRUBER, T. R. Toward principles for the design of ontologies used for knowledge sharing. **International journal of human-computer studies**, Elsevier, v. 43, n. 5, p. 907–928, 1995. Citado na página 5.
- HALANG, W. A.; STOYENKO, A. D. **Real Time Computing**. [S.l.]: Springer-Verlag New York, Inc., 1994. Citado na página 75.

JAIN, R. **The art of computer systems performance analysis**. [S.l.]: John Wiley & Sons, 1991. Citado 13 vezes nas páginas 1, 16, 17, 19, 34, 70, 71, 72, 73, 74, 75, 76 e 77.

JERRY, B. **Discrete-event system simulation**. [S.l.]: Pearson Education India, 1984. Citado na página 18.

KELTON, W. D.; LAW, A. M. **Simulation modeling and analysis**. [S.l.]: McGraw Hill Boston, 2000. Citado na página 18.

LI, P.; SHI, D.; LI, J. Performance test and bottle analysis based on scientific research management platform. In: IEEE. **Wavelet Active Media Technology and Information Processing (ICCWAMTIP), 2013 10th International Computer Conference on**. [S.l.], 2013. p. 218–221. Citado na página 25.

MACDOUGALL, M. H. **Simulating computer systems: techniques and tools**. [S.l.]: MIT press, 1989. Citado na página 76.

MARTIMIANO, L. A. F. **Sobre a estruturação de informação em sistemas de segurança computacional: o uso de ontologias**. Tese (Doutorado) — Universidade de São Paulo, 2006. Citado 2 vezes nas páginas 6 e 7.

MCGUINNESS, D. L.; HARMELEN, F. V. *et al.* Owl web ontology language overview. **W3C recommendation**, v. 10, n. 10, 2004. Citado na página 7.

MEIER, J.; FARRE, C.; BANSODE, P.; BARBER, S.; REA, D. **Performance testing guidance for web applications: patterns & practices**. [S.l.]: Microsoft press, 2007. Citado 11 vezes nas páginas 24, 25, 35, 70, 71, 72, 73, 74, 75, 76 e 77.

MENASCÉ, D. A.; ALMEIDA, V. A.; RIEDI, R.; RIBEIRO, F.; FONSECA, R.; JR, W. M. A hierarchical and multiscale approach to analyze e-business workloads. **Performance Evaluation**, Elsevier, v. 54, n. 1, p. 33–57, 2003. Citado na página 16.

MOLYNEAUX, I. **The Art of Application Performance Testing: Help for Programmers and Quality Assurance**. [S.l.]: "O'Reilly Media, Inc.", 2009. Citado 3 vezes nas páginas 1, 24 e 25.

MURATA, T. Petri nets: Properties, analysis and applications. **Proceedings of the IEEE**, IEEE, v. 77, n. 4, p. 541–580, 1989. Citado na página 18.

NAKAMURA, L. H. V. **Utilização de web semântica para seleção de informações de web services no registro UDDI uma abordagem com qualidade de serviço**. Dissertação (Mestrado) — Universidade de São Paulo, 2010. Citado na página 5.

NOY, N. F.; MCGUINNESS, D. L. **Ontology development 101: A guide to creating your first ontology**. [S.l.]: Stanford knowledge systems laboratory technical report KSL-01-05 and Stanford medical informatics technical report SMI-2001-0880, Stanford, CA, 2001. Citado 3 vezes nas páginas 5, 6 e 35.

RAUTENSTRAUCH, C.; SCHOLZ, A. Performance engineering on the basis of performance service levels. In: **Performance Engineering**. [S.l.]: Springer, 2001. p. 68–77. Citado na página 22.

- SANCHO, P. P.; JUIZ, C.; PUIGJANER, R.; CHUNG, L.; SUBRAMANIAN, N. An approach to ontology-aided performance engineering through nfr framework. In: ACM. **Proceedings of the 6th international workshop on Software and performance**. [S.l.], 2007. p. 125–128. Citado 2 vezes nas páginas 11 e 14.
- SARMANHO, F. S.; SOUZA, P. S.; SOUZA, S. R.; SIMÃO, A. S. Structural testing for semaphore-based multithread programs. In: **Computational Science–ICCS 2008**. [S.l.]: Springer, 2008. p. 337–346. Citado na página 8.
- SCHMIETENDORF, A.; SCHOLZ, A.; RAUTENSTRAUCH, C. Evaluating the performance engineering process. In: ACM. **Proceedings of the 2nd international workshop on Software and performance**. [S.l.], 2000. p. 89–95. Citado na página 22.
- SHERE, K. D.; CARLSON, R. A. A methodology for design, test, and evaluation of real-time systems. **Computer**, IEEE, v. 27, n. 2, p. 35–48, 2002. Citado 2 vezes nas páginas 13 e 14.
- SILVA, A. L. **Ontologia para teste de desempenho de software**. Dissertação (Mestrado) — Pontifícia Universidade Católica do Rio Grande do Sul, 2013. Citado 3 vezes nas páginas 15, 12 e 24.
- SMITH, C. Performance engineering for software architectures. In: IEEE. **Computer Software and Applications Conference, 1997. COMPSAC’97. Proceedings., The Twenty-First Annual International**. [S.l.], 1997. p. 166–167. Citado na página 21.
- SMITH, C. U. The evolution of software performance engineering: a survey. In: IEEE COMPUTER SOCIETY PRESS. **Proceedings of 1986 ACM Fall joint computer conference**. [S.l.], 1986. p. 778–783. Citado 4 vezes nas páginas 15, 20, 22 e 23.
- SMITH, C. U.; WILLIAMS, L. G. **Performance Solutions. A Practical Guide to Creating Responsive, Scalable Software**. [S.l.]: Addison-Wesley Professional, 2001. Citado 7 vezes nas páginas 1, 70, 71, 72, 73, 75 e 76.
- SOMMERVILLE, I.; MELNIKOFF, S. S. S.; ARAKAKI, R.; BARBOSA, E. de A. **Engenharia de software**. [S.l.]: Addison Wesley São Paulo, 2003. v. 6. Citado 5 vezes nas páginas 71, 72, 73, 74 e 76.
- SOUZA, P.; SOUZA, S.; VERGILIO, S. Teste de programas concorrentes. In: DELAMARO JOSÉ C. MALDONADO, M. J. M. E. (Ed.). **Introdução ao teste de software**. [S.l.]: Campus. Citado 3 vezes nas páginas 8, 71 e 75.
- SOUZA, P. S.; SOUZA, S. R.; ZALUSKA, E. Structural testing for message-passing concurrent programs: an extended test model. **Concurrency and Computation: Practice and Experience**, Wiley Online Library, v. 26, n. 1, p. 21–50, 2014. Citado 5 vezes nas páginas 15, 3, 5, 7 e 9.
- SOUZA, P. S.; SOUZA, S. S.; ROCHA, M. G.; PRADO, R. R.; BATISTA, R. N. Data flow testing in concurrent programs with message passing and shared memory paradigms. **Procedia Computer Science**, Elsevier, v. 18, p. 149–158, 2013. Citado 2 vezes nas páginas 7 e 9.
- SOUZA, S.; VERGILIO, S. R.; SOUZA, P.; SIMAO, A.; HAUSEN, A. C. Structural testing criteria for message-passing parallel programs. **Concurrency and Computation: Practice and Experience**, Wiley Online Library, v. 20, n. 16, p. 1893–1916, 2008. Citado 2 vezes nas páginas 8 e 10.

SUFFIAN, M. D. M.; FAHRURAZI, F. R. Performance testing: Analyzing differences of response time between performance testing tools. In: IEEE. **Computer & Information Science (ICCIS), 2012 International Conference on**. [S.l.], 2012. v. 2, p. 919–923. Citado na página 24.

TAVARES, R. Construindo mapas conceituais. **Ciências & Cognição**, Instituto de Ciências Cognitivas, v. 12, p. 72–85, 2007. Citado na página 26.

TERKAJ, W.; URGO, M. Ontology-based modeling of production systems for design and performance evaluation. In: IEEE. **Industrial Informatics (INDIN), 2014 12th IEEE International Conference on**. [S.l.], 2014. p. 748–753. Citado 2 vezes nas páginas 11 e 14.

USCHOLD, M.; GRUNINGER, M. Ontologies: Principles, methods and applications. **The knowledge engineering review**, Cambridge Univ Press, v. 11, n. 02, p. 93–136, 1996. Citado 2 vezes nas páginas 5 e 6.

WOODSIDE, M.; FRANKS, G.; PETRIU, D. C. The future of software performance engineering. In: IEEE. **Future of Software Engineering, 2007**. [S.l.], 2007. p. 171–187. Citado 2 vezes nas páginas 20 e 22.

LISTA DE CONCEITOS E DEFINIÇÕES

A lista a seguir mostra cada conceito encontrado para cada área, e a definição existente para aquela área. Os conceitos são mostrados em ordem alfabética.

Lista de Conceitos e Definições

Rafael de Souza Stabile

5 de agosto de 2016

Conceito	Definição		
	Avaliação de Desempenho	Engenharia de Desempenho de Software	Teste de Desempenho de Software
Análise de Resultados	Etapa da Avaliação de Desempenho. Interpretação dos resultados, comparação entre eles. (JAIN, 1991)		Ou Investigação, é um etapa de teste com base na coleta de informações relacionadas às características de velocidade, escalabilidade e / ou a estabilidade do produto em teste que podem ter valor na determinação ou melhorar a qualidade do produto (MEIER <i>et al.</i> , 2007).
Ambiente de execução	Especificações dos componentes computacionais onde o sistema avaliado executa (FORTIER; MICHEL, 2003).	Especificações dos componentes computacionais onde o software a ser construído executará (SMITH; WILLIAMS, 2001)	Representa a arquitetura do sistema e sua implementação física, em que se fará os testes (MEIER <i>et al.</i> , 2007).
Benchmarks	Programa usado para extrair métricas de desempenho e utilização de recursos, executando certas funções do sistema. (JAIN, 1991)	Programa usado para extrair métricas de desempenho e utilização de recursos, executando certas funções do sistema. (BONDI, 2014)	Programa usado para extrair métricas de desempenho e utilização de recursos, executando certas funções do sistema. (MEIER <i>et al.</i> , 2007)
Capacidade do sistema	Carga de trabalho processada pelo sistema sem que perca desempenho. (FORTIER; MICHEL, 2003)	Carga de trabalho máxima que o sistema pode lidar (BONDI, 2014)	Carga de trabalho máxima que o sistema pode lidar (MEIER <i>et al.</i> , 2007).

Conceito	Definição		
	Avaliação de Desempenho	Engenharia de Desempenho de Software	Teste de Desempenho de Software
Carga de Trabalho	Requisições feitas por usuários do sistema(JAIN, 1991).	Especifica a utilização prevista para o sistema (SMITH; WILLIAMS, 2001).	Estímulo aplicado ao sistema para simular um padrão de uso(MEIER <i>et al.</i> , 2007).
Casos de Teste			Conjunto de Dados de Teste (Elementos de entrada do software) mais a saída esperada(SOUZA; SOUZA; VERGILIO,).
Cenários	Representam as configurações diferentes para cada experimento realizado, variando-se os níveis dos fatores envolvidos. (JAIN, 1991)	Descrições do sistema, representado através de casos de uso que são diagramas que mostram as possíveis interações com o sistema (SOMMERVILLE <i>et al.</i> , 2003).	É uma sequência de etapas da aplicação. Um cenário pode representar um caso de uso ou uma função de negócios (MEIER <i>et al.</i> , 2007).
Confiabilidade	Tempo médio entre as falhas ocorridas no sistema. (JAIN, 1991)	Requisito de desempenho que especifica o funcionamento correto do sistema (sem falhas) por um período de tempo (BONDI, 2014).	Representa o correto funcionamento das funções do software por um longo período de tempo. Medido através da taxa de falhas ocorridas no sistema(MEIER <i>et al.</i> , 2007) .
Desenvolvimento de modelos de desempenho	Técnica que consiste na construção de modelos de modo que se possa extrair métricas de desempenho analiticamente ou por simulação (JAIN, 1991)	Processo de construção de modelos de modo que se possa extrair métricas de desempenho analiticamente ou por simulação (SMITH; WILLIAMS, 2001)	
Disponibilidade	Fração de tempo que o sistema está disponível para requisições dos usuários. (JAIN, 1991)	Requisito de desempenho esperado. Representa o período de tempo em que o sistema responde à requisições. (BONDI, 2014)	Período de tempo que o sistema responde à requisições. (MEIER <i>et al.</i> , 2007)
Eficácia			Atributo do teste em atingir os objetivos a ele determinados, ou seja, ser capaz de revelar os defeitos do software.(SOUZA; SOUZA; VERGILIO,)
Eficiência	Medida que reflete o quão eficiente é um determinado recurso (JAIN, 1991) (Ou especificamente para programas concorrentes é tempo de execução em um processador dividido pelo tempo de execução em n processadores multiplicado por este número de processadores. (GRAMA <i>et al.</i> , 2003))		

Conceito	Definição		
	Avaliação de Desempenho	Engenharia de Desempenho de Software	Teste de Desempenho de Software
Escalabilidade	Capacidade do sistema de estar preparado para crescer ou diminuir a fim de manipular uma carga crescente ou decrescente de trabalho de forma uniforme. (JAIN, 1991)	Requisito de desempenho de sistemas de crescer ou diminuir a fim de manipular uma carga crescente ou decrescente de trabalho de forma uniforme. (BONDI, 2014)	Habilidade da aplicação de tratar carga de trabalho adicional, sem afetar o desempenho (MEIER <i>et al.</i> , 2007).
Estabilidade		Requisito de desempenho do sistema que é o conjunto de confiabilidade, robustez, disponibilidade e a consistência da resposta (BONDI, 2014)	Conjunto de confiabilidade, robustez, disponibilidade e a consistência da resposta do sistema (MEIER <i>et al.</i> , 2007).
Experimentos	Execuções controladas do sistema avaliado sob determinadas cargas de trabalho.		
Fatores	Variável que afeta as variáveis de resposta e que podem assumir diversas alternativas (JAIN, 1991).		
Ferramentas de desempenho	Softwares utilizados para auxiliar a avaliação de desempenho (JAIN, 1991)	Softwares utilizados para auxiliar na análise de desempenho do sistema (BONDI, 2014)	Softwares utilizados para auxiliar na realização de testes (MEIER <i>et al.</i> , 2007)
Gargalos	Parte que limita o desempenho ou a capacidade de todo um sistema (JAIN, 1991)		Parte do sistema que faz o desempenho do sistema degradar (SMITH; WILLIAMS, 2001)
Geradores de carga	Ferramenta utilizada para produzir cargas de trabalho (JAIN, 1991).	Ferramenta usada para simular requisições de usuários no sistema	Ferramenta que produz requisições de usuários ao sistema, simulando cargas de trabalho (MEIER <i>et al.</i> , 2007)
Hits por segundo			Métrica de desempenho que visa medir a quantidade de requisições recebidas por um servidor. (MEIER <i>et al.</i> , 2007)
Implementação do Software	Construção de Benchmarks, protótipos ou programas de simulação. (JAIN, 1991)	Construção/Desenvolvimento de softwares com desempenho. (SOMMERVILLE <i>et al.</i> , 2003)	
Latência			Medida da capacidade de resposta que representa o tempo que leva a completar a execução de uma requisição. (MEIER <i>et al.</i> , 2007)

Conceito	Definição		
	Avaliação de Desempenho	Engenharia de Desempenho de Software	Teste de Desempenho de Software
Manutenção		Processo de melhoria e otimização de um software já desenvolvido, como também reparo de defeitos revelados. (SOMMERVILLE <i>et al.</i> , 2003)	
Métodos baseados na solução por aferição	Método de avaliação no qual as métricas são extraídas diretamente do sistema avaliado(JAIN, 1991)	Método para a construção de sistemas no qual o desempenho é testado depois de o software estar concluído, e feito modificações nele caso necessário. (SMITH; WILLIAMS, 2001)	
Métodos baseados na solução por modelagem	Método de avaliação no qual as métricas de desempenho são extraídas a partir de um modelo representativo do sistema avaliado(JAIN, 1991).	Método para a construção de sistemas com desempenho no qual, antes de implementação é feito um modelo do software e simulado para verificar o desempenho do mesmo. (SMITH; WILLIAMS, 2001)	
Métrica de desempenho	Medida usada para avaliar o desempenho de um sistema(JAIN, 1991).	Descrição quantitativa de desempenho. Usadas para descrever o os requisitos de desempenho.(BONDI, 2014)	Medições obtidas através de execuções de teste de desempenho, expressadas normalmente através de uma escala conhecida (MEIER <i>et al.</i> , 2007).
Modelos Analíticos	Modelos feitos através de fórmulas matemáticas que mostram como se comporta o sistema analisado (JAIN, 1991)	Modelos feitos através de fórmulas matemáticas que mostram como se comporta o sistema analisado (SMITH; WILLIAMS, 2001)	
Monitores	Ferramenta usada para coletar dados enquanto o sistema está em execução(JAIN, 1991).	Ferramenta usada para coletar dados enquanto o sistema está em execução. (SMITH; WILLIAMS, 2001)	Ferramenta usada para coletar dados enquanto o sistema está em execução. (MEIER <i>et al.</i> , 2007)
Níveis	Os valores que um determinado fator pode assumir(JAIN, 1991).		
Perfil de usuário			Scripts usados para simular o comportamento do usuário no sistema ou uma carga de trabalho. (MEIER <i>et al.</i> , 2007)
Planejamento e gerenciamento de Capacidade	Planejar como os recursos serão utilizados e como a carga de trabalho imposta é tratada. (FORTIER; MICHEL, 2003)	Projetar como a carga de trabalho a ser recebida pelo sistema será processada (BONDI, 2014)	

Conceito	Definição		
	Avaliação de Desempenho	Engenharia de Desempenho de Software	Teste de Desempenho de Software
Planejamento de Experimentos	Etapas da avaliação de desempenho. Determina quais dados e a quantidade que devem ser coletados. Determina a quantidade de experimentos a ser realizado.(JAIN, 1991)		
Planejamento de teste			Identifica cenários principais, determina a variabilidade entre os usuários representativos e como simular esta variabilidade, defini os dados do teste e estabelece métricas para coletar. (MEIER <i>et al.</i> , 2007)
Projeto	Refere-se ao projeto de experimentos, parte do Planejamento de experimentos. Determina o número de experimentos a serem considerados, incluindo o número de fatores e níveis, a combinação entre os níveis e o número de replicações. São três tipos: Projeto Fatorial Completo, que utiliza todas as possibilidades de combinação entre os níveis de todos os fatores; Projeto Fatorial Parcial, que realiza apenas partes dos experimentos o projeto fatorial completo; E Projeto Simples, que se inicia com uma configuração e varia um fator por vez, para ver como cada fator influencia no desempenho (JAIN, 1991).	Refere-se ao projeto do software. Determina como serão implementados seus módulos, a comunicação entre eles e a arquitetura do software considerando seus requisitos. (SOMMERVILLE <i>et al.</i> , 2003)	
Protótipos	Versão inicial e simples do sistema a ser avaliado(JAIN, 1991).	Um software sem funcionalidades inteligentes (acesso à base de dados, por exemplo), podendo conter apenas funcionalidades gráficas (SOMMERVILLE <i>et al.</i> , 2003).	
Recursos disponíveis	Físicos como processador, memória, disco. E também lógicos como base de dados ou recursos do sistema operacional.	Físicos como processador, memória, disco. E também lógicos como base de dados ou recursos do sistema operacional.	Físicos como processador, memória, disco. E também lógicos como base de dados ou recursos do sistema operacional.

Conceito	Definição			Teste de Desempenho de Software
	Avaliação de Desempenho	Engenharia de Desempenho de Software		
Redes de Filas	Tipo de modelagem que envolve um ramo da probabilidade que estuda o fenômeno da formação de filas de solicitantes de serviços, que são providos por um determinado recurso(FORTIER; MICHEL, 2003).	Tipo de modelagem que envolve um ramo da probabilidade que estuda o fenômeno da formação de filas de solicitantes de serviços, que são providos por um determinado recurso. (SMITH; WILLIAMS, 2001)		
Redes de Petri		Técnica de modelagem que permite a representação do comportamento de sistemas através de estados e transições. (SMITH; WILLIAMS, 2001)		
Replicação	Repetição de parte ou todo o experimento (JAIN, 1991)			
Requisitos de Desempenho		São objetivos, propriedades e restrições que o sistema deve possuir para satisfazer contratos, padrões ou especificações de acordo com o(s) usuário(s). (BONDI, 2014)	São objetivos, propriedades que devem ser cobertos pelo teste. (MEIER <i>et al.</i> , 2007)	
Requisitos de Teste			Um elemento específico de um artefato de software que um caso de teste deve satisfazer ou cobrir. (SOUZA; SOUZA; VERGILIO,)	
Responsividade	Medida que especifica se o sistema responde a uma requisição e quanto tempo leva para obter essa resposta(HALANG; STOYENKO, 1994).	Requisito de desempenho que especifica se o sistema responde a uma requisição e quanto tempo leva para obter essa resposta. (SMITH; WILLIAMS, 2001)	Medida que especifica se o sistema responde a uma requisição e quanto tempo leva para obter essa resposta. (MEIER <i>et al.</i> , 2007)	
Sessão de Teste			Agrupar informações de uma atividade de teste conduzida, como programas testados, elementos avaliados, casos de teste e critérios de teste selecionados. Tem a característica de armazenar essas informações e permitir que a atividade possa ser interrompida e continuada posteriormente a partir do ponto em que foi interrompida, preservando portanto, as informações necessárias para que a atividade de teste seja continuada. (SOUZA; SOUZA; VERGILIO,)	

Conceito	Definição		
	Avaliação de Desempenho	Engenharia de Desempenho de Software	Teste de Desempenho de Software
Simulação	Imitar o comportamento de sistemas reais através de execuções de modelos (MACDOUGALL, 1989).	Execução/Solução de modelos (SMITH; WILLIAMS, 2001)	
Sistema	Elemento computacional que pode ser avaliado como software, SO, componente de hardware... (FORTIER; MICHEL, 2003)	Conjunto de Software, ambiente de execução e rede. (SOMMERVILLE <i>et al.</i> , 2003)	
Solução do modelo	Extração de métricas de desempenho do modelo construído.	Extração de métricas de desempenho do modelo construído.	
Tempo de Resposta	Tempo para responder uma requisição (JAIN, 1991)	Tempo para responder uma requisição. (BONDI, 2014)	Tempo para responder uma requisição (MEIER <i>et al.</i> , 2007).
Técnicas de Avaliação	Técnicas utilizadas para quantificar uma métrica de desempenho. São elas: Aferição (Benchmarks, Coleta de Dados e Protótipos), e Modelagem (Analítica ou por simulação) (JAIN, 1991).		
Teste de Capacidade			Determina o ponto de falha do sistema, ou seja, qual a carga máxima suportada (MEIER <i>et al.</i> , 2007).
Teste de Carga			Determina ou valida as características do sistema teste quando submetido a cargas de trabalho e volumes de carga previstas durante suas operações (MEIER <i>et al.</i> , 2007).
Teste de Desempenho		Etapa da Engenharia de Desempenho que visa testar o desempenho dos softwares produzidos. (BONDI, 2014)	
Teste de Estresse			Destinado a avaliar o comportamento de uma aplicação quando ela é submetida em condições de carga de pico. O objetivo do teste de estresse é revelar erros da aplicação que surgem somente sob condições de alta carga (MEIER <i>et al.</i> , 2007).
Teste de Resistência			É usado para determinar ou validar as características de desempenho do produto em teste, submetido à ele uma carga de trabalho durante um período prolongado de tempo. (MEIER <i>et al.</i> , 2007)

Conceito	Definição		
	Avaliação de Desempenho	Engenharia de Desempenho de Software	Teste de Desempenho de Software
Teste de Smoke			Execução inicial de um teste de desempenho para ver se o sistema pode executar suas operações sob uma carga esperada(MEIER <i>et al.</i> , 2007).
Teste de Spike			Determina ou valida as características de desempenho do produto em teste quando sujeito à volumes de carga de trabalho que aumentam repentinamente(MEIER <i>et al.</i> , 2007).
Teste de Validação			Compara as características de velocidade, escalabilidade e estabilidade do sistema em teste com as expectativas que foram criadas ou presumidas para ele(MEIER <i>et al.</i> , 2007).
Transação			Período de tempo que se refere a uma ação do usuário no sistema (MEIER <i>et al.</i> , 2007)
Usuário Virtual			Simula o comportamento do usuário real no sistema em teste (MEIER <i>et al.</i> , 2007)
Utilização de Recurso	Fração de tempo em que um recurso está ocupado respondendo requisições (FORTIER; MICHEL, 2003)	Fração de tempo em que um recurso está ocupado respondendo ou processando requisições (BONDI, 2014).	Fração de tempo em que um recurso está ocupado respondendo ou processando requisições (MEIER <i>et al.</i> , 2007)
Variável de Resposta	Resultado de um experimento, que geralmente é a medida de desempenho de um sistema(JAIN, 1991).		
Vazão (Throughput)	Número de atividades executadas por unidade de tempo (JAIN, 1991)	Número de atividades executadas por unidade de tempo (BONDI, 2014)	Número de atividades executadas por unidade de tempo(MEIER <i>et al.</i> , 2007).