Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Identificação automatizada de processos elementares a partir da análise de requisições HTTP

Diego Emanuel Ferreira da Rocha

Dissertação apresentada como requisito parcial para qualificação do
Mestrado Profissional em Computação Aplicada

Orientador
Prof. Dr. André Luiz Peron Martins Lanna

Brasília
2025

Ficha catalográfica elaborada automaticamente,
com os dados fornecidos pelo(a) autor(a)

ER672i

Emanuel Ferreira da Rocha, Diego
Identificação automatizada de processos elementares a partir da análise de requisições HTTP / Diego Emanuel Ferreira da Rocha; orientador Andre Luiz Peron Martins Lanna. Brasília, 2025.
62 p.

Dissertação(Mestrado Profissional em Computação Aplicada) Universidade de Brasília, 2025.

1. Medição de software. 2. Simple Function Points. 3. Longest Common Subsequence. 4. Automatização. 5. Hypertext Transfer Protocol. I. Peron Martins Lanna, Andre Luiz, orient. II. Título.

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Identificação automatizada de processos elementares a partir da análise de requisições HTTP

Diego Emanuel Ferreira da Rocha

Dissertação apresentada como requisito parcial para qualificação do
Mestrado Profissional em Computação Aplicada

Prof. Dr. André Luiz Peron Martins Lanna (Orientador)
FGA/UnB

Prof.a Dr.a Monalessa Perini Barcellos
PPGI - UFES

Dr. Sérgio Antônio Andrade de Freitas
FGA - UnB

Prof. Dr. Edna Dias Canedo
Coordenador do Programa de Pós-graduação em Computação Aplicada

Brasília, 23 de Outubro de 2025

Resumo

A medição funcional é um importante recurso para predição de custos, estimativa de duração e aferição de indicadores de projetos ao longo do ciclo de vida de desenvolvimento do software. Dentre os métodos existentes, destaca-se o *Simple Function Points* (SFP), uma versão simplificada da Análise de Pontos de Função (APF), por ter ampla aceitação de mercado, suporte da comunidade (*International Function Point Users Group - IFPUG*) e vasto *benchmarking*. No entanto, a adoção do SFP em projetos de desenvolvimento traz desafios pois o processo de medição é manual e propenso a falhas, requer documentação do sistema medido e especialistas que conheçam o método. Superar a dependência da execução de atividades manuais, em especial aquelas relacionadas à medição propriamente dita, pode levar a uma maior eficiência na utilização de métricas de tamanho funcional em projetos de desenvolvimento. Este trabalho apresenta um método que automatiza o processo de identificação de processos elementares com base em requisições HTTP, de modo independente das tecnologias utilizadas na construção do software e aderente às diretrizes definidas pelo SFP. Por meio da captura e análise de requisições HTTP de aplicações WEB o método filtra as requisições relacionadas à funcionalidades do software, identifica as jornadas de usuário pelo agrupamento de requisições que ocorrem em conjunto e, por fim, classifica cada agrupamento como processos elementares seguindo as regras do SFP. Os resultados obtidos mostram à capacidade do método de analisar padrões de requisições de forma detalhada e obter valores de medição de tamanho funcional equivalentes as medições realizadas de forma manual, alcançando uma variação média de 17%, contra a variação aceitável de mercado de 15% quando o processo realizado de forma manual.

Palavras-chave: Medição de software, Automatização, Simple Function Points - SFP, HTTP, Longest Common Subsequence - LCS

Abstract

The functional measurement is an important resource for cost prediction, duration estimation, and measurement of project indicators throughout the software development life cycle. Among the existing methods, Simple Function Points (SFP) stands out, a simplified version of Function Point Analysis (FPA), due to its broad market acceptance, community support (International Function Point Users Group - IFPUG), and extensive benchmarking. However, adopting SFP in development projects poses challenges because the measurement process is manual and prone to errors, requiring documentation of the measured system and specialists familiar with the method. Overcoming reliance on manual activities—particularly those related to measurement itself—can lead to greater efficiency in using functional size metrics in development projects. This work presents a method that automates part of the measurement process, independently of the technologies used to build the software and adhering to the guidelines defined by SFP. Through capturing and analyzing HTTP requests from WEB applications, the method filters requests related to the software's functionalities, identifies user journeys by clustering requests that occur together, and finally classifies each cluster as elementary processes following SFP rules. The results obtained demonstrate the method's ability to analyze request patterns in detail and to obtain functional size measurement values equivalent to those obtained manually, achieving an average variation of 17%, compared to the market's acceptable variation of 15% when the process is performed manually.

Keywords: Software Measurement, Automation, Simple Function Points - SFP, HTTP, Longest Common Subsequence - LCS

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

A medida de tamanho funcional (do inglês *Functional Size Measurement* – FSM) (ISO, 2007) é amplamente aceita há décadas como um mecanismo para predição de custos, duração e qualidade das atividades de software (HUIJGENS et al., 2016). Seu reconhecimento como instrumento de medição dá-se por sua usabilidade nas fases iniciais do ciclo de desenvolvimento e por sua independência em relação à linguagem de implementação, métodos e tecnologias (OZKAN; DEMIRORS, 2016). Dentre as diversas abordagens de FSM, a Análise de Pontos de Função - APF (ALBRECHT, 1979) (mantida pelo *International Function Point Users Group* - IFPUG (INTERNATIONAL. . . ,)) é uma das medidas de tamanho funcional mais aceitas atualmente (BARROS; GONCALVES, 2019). Suas medidas podem ser utilizadas para diversos propósitos tais como estimativas de tamanho funcional de projetos, avaliação da qualidade, *benchmarking* e, em especial, para a aferição de produtividade, cálculo de índice de defeitos e custos em contratos de terceirização (FERNÁNDEZ-DIEGO et al., 2020).

No Brasil, para a celebração de contratos com a administração pública federal direta, autárquica ou fundacional do Poder Executivo exige-se a utilização de um instrumento prescritivo que atualmente é regido pela Instrução Normativa nº 01/2019 (Ministério da Economia; Secretaria Especial de Desburocratização, Gestão e Governo Digital; Secretaria de Governo Digital, 2019), em substituição à Instrução Normativa nº 04/2014 (Ministério da Economia; Secretaria Especial de Desburocratização, Gestão e Governo Digital; Secretaria de Governo Digital, 2014). A IN 01/2019 obriga que contratos de desenvolvimento e manutenção de software devem prever pagamentos vinculados a resultados e/ou aos produtos entregues de modo que métricas, indicadores e níveis mínimos de serviço tenham que ser definidos como critérios de aceitação do serviço prestado (KOVAGS; FALCHI; RIVAS, 2019).

Em se tratando especificamente de contratos de desenvolvimento de software junto à administração pública federal brasileira, a IN 01/2019 (Ministério da Economia; Secre-

taria Especial de Desburocratização, Gestão e Governo Digital; Secretaria de Governo Digital, 2019) restringe à adoção da unidade de medida homem-hora ou equivalente para aferição de esforço. Para tanto, a medição/estimativa através de Pontos de Função (PF) é frequentemente utilizada para esse fim, além de ser utilizada como unidade de medida de referência para tomada de preço em processos licitatórios (Ministério da Economia; Secretaria Especial de Desburocratização, Gestão e Governo Digital; Secretaria de Governo Digital, 2010). Os valores de referência informados pelos órgãos proponentes nas licitações, como o valor estimado em pontos de função de cada sistema de seu portfólio, é uma informação crucial para as empresas interessadas e a inexatidão de tais informações é extremamente onerosa pois as induzem a realizar ofertas baseadas em números incorretos. No entanto, devido a fatores como falta de documentação de sistemas legados, ausência de profissionais capacitados para medição e falta de tempo hábil para realizar as aferições, muitas licitações resultam em precificações que não condizem com a real dimensão do portfólio dos órgãos licitantes, o que acarreta em prejuízos tanto para as empresas contratadas quanto para os órgãos contratantes.

A estrutura e aplicabilidade dos métodos de FSM, dentre eles a APF, vêm sendo investigadas para determinar distintos problemas que afetam sua consistência, confiabilidade e exatidão dos resultados de medição obtidos (SILVA; PINHEIRO; ALBUQUERQUE, 2016). Em linhas gerais, a adoção de métodos FSM é fortemente limitada pelo tempo e custo do processo de contagem (STAMBOLLIAN; ABRAN, 2016). A lentidão inerente ao processo de medição e a necessidade de intervenção de especialistas figuram como alguns dos fatores que dificultam a utilização dos métodos FSM, além da propensão a falhas inerentes a processos realizados de forma manual (QUESADA-LóPEZ, 2018). Contudo, uma vez obtido, o tamanho do software é de grande valia em processos de desenvolvimento e um dos principais indicadores utilizados para aferição de prazos e custos de projetos (GUEVARA; FERNáNDEZ-DIEGO; LOKAN, 2016) (LAVAZZA; LOCORO; MELI, 2024a).

Diante desse cenário, qualquer empresa que tenha como escopo de atuação a prestação de serviços de desenvolvimento de software para o Governo Federal Brasileiro tem a necessidade de medir de maneira mais precisa, confiável e repetível, não sendo essa realidade diferente à INDRA. A INDRA é uma das empresas líderes mundiais em tecnologia e consultoria e o parceiro tecnológico para as operações-chave dos negócios dos seus clientes em todo mundo. É líder mundial no fornecimento de soluções proprietárias em segmentos específicos dos mercados de Transporte e Defesa, e a empresa líder em transformação digital e consultoria em Tecnologia da Informação na Espanha e América Latina através da sua subsidiária Minsait. O seu modelo de negócio baseia-se numa oferta abrangente de produtos proprietários, com uma abordagem *end-to-end*, de alto valor e com um ele-

vado componente de inovação. No Brasil, tem grande atuação em clientes dos segmentos público e privado, com vários contratos de consultoria, em que muitos clientes utilizam a métrica por pontos de função como mecanismo para aferição de resultados.

Nesse contexto, a existência de um mecanismo para automatização da medição do tamanho funcional permitiria reduzir o tempo e o custo do processo de contagem, além de possibilitar uma melhora na exatidão e repetibilidade dos resultados de medição.

1.1 Definição do Problema

O Acórdão 2.362/2015 do Tribunal de Contas da União - TCU (UNIãO, 2015b) analisou contratos de prestação de serviços de fábricas de software e identificou dificuldades na utilização da métrica de pontos de função nos órgãos auditados. As principais dificuldades relatadas foram: (i) a correlação entre o valor expresso em pontos de função e o custo da prestação do serviço nem sempre se mostrava adequada em situações de software de elevada complexidade; (ii) dificuldade do uso de pontos de função, visto que os demandantes não conheciam a técnica de forma suficiente; e (iii) esses fatores resultam em um maior ônus para medição do volume de serviços prestados (ALMEIDA, 2019).

Apesar do TCU indicar a utilização de um modelo de gestão contratual orientado a resultados, durante a execução dos contratos não é raro observar discrepâncias em relação aos valores estipulados nas contratações e aos esforços empreendidos pelas empresas contratadas (UNIãO, 2007). Isso ocorre porque a medição de tamanho funcional requer a identificação de vários elementos do software como telas, campos, tabelas utilizadas, regras de negócio, além da aplicação de diversas regras para garantir que o resultado da medição atenda a critérios de qualidade (MARCÉN et al., 2024). Essas atividades são realizadas geralmente por um especialista de forma manual (LAVAZZA; MORASCA; ROBIOLO, 2013). Embora seja uma atividade técnica, o processo manual de contagem apresenta uma variação considerável. Kemerer apresenta um cenário de variação médio de 12% em contagens do mesmo produto realizada por diferentes especialistas na mesma organização (POSPIESZNY; CZARNACKA-CHROBOT; KOBYLINSKI, 2018), enquanto Low e Jeffery apontam uma variação média de 30% em contagens realizadas na mesma organização e percentual superior a esse patamar quando a mesma contagem é realizada por organizações distintas (WALLACE; SHEETZ, 2014).

O método APF do IFPUG (INTERNATIONAL FUNCTION POINT GROUP - IFPUG, 2010) é um padrão internacional que está em conformidade com as diretrizes definidas pela norma ISO/IEC 14143-1/2007 (ISO, 2007) para medição do tamanho funcional de software. O Simple Function Points (SFP) (INTERNATIONAL FUNCTION POINT GROUP - IFPUG, 2021) é uma simplificação do método APF (INTERNATIO-

NAL FUNCTION POINT GROUP - IFPUG, 2010) que mantém a conformidade com a ISO/IEC 14143-1:2007, utiliza pontos de função como métrica e é aceito pelo TCU. Embora seja mais simples, ele também está sujeito às imprecisões e subjetividades decorrentes da realização manual de suas atividades permitindo, portanto, medidas ou estimativas imprecisas. Isso exposto, o problema considerado por esse trabalho pode ser resumido na seguinte questão:

Questão de pesquisa: É possível automatizar, em partes ou em sua totalidade, a medida de tamanho funcional em SFP à partir da execução de um software?

1.2 Justificativa

De acordo com o Gartner, serão gastos US$ 5 trilhões no mercado de TI global em 2024, aumento de 6,8% em relação a 2023 (GASTOS. . . ,). Em 2023, o crescimento do mercado brasileiro foi de 5% no setor, chegando a US$ 80 bilhões (MERCADO. . . ,). Parte deste investimento refere-se a contratos de TIC na administração pública federal brasileira, em torno de R$ 2,5 bilhões (EFFECTI, 2021). O vultuoso orçamento disponível aos entes públicos para contratação em TIC é um atrativo para empresas, porém é necessária uma avaliação minuciosa de todas as exigências editalícias por parte dos proponentes fornecedores com o objetivo de analisar se determinada contratação lhes é vantajosa ou não. Nesse sentido, faz parte da estratégia comercial de diversas empresas a avaliação periódica de oportunidades de negócio junto à administração pública federal por meio da análise dos critérios estabelecidos em instrumentos licitatórios, com vistas a determinar as oportunidades que a empresa deverá empreender esforços para obtenção do contrato ofertado.

A avaliação dos critérios estipulados em oportunidades de contratações públicas não é algo trivial pois requer profissionais especializados no modelo de contratação, *expertise* quanto ao serviço ofertado, avaliação dos riscos associados, prospecção quanto ao possível retorno do investimento, dentre vários outros aspectos que precisam ser analisados com intuito de apoiar a tomada de decisão da organização para concorrência no pleito em análise. Embora o volume de oportunidades do mercado seja grande, ressalta-se a ausência de modelos que auxiliem a alta gestão na análise dos critérios (e avaliação dos resultados) de modo célere, objetivo e confiável, refletindo diretamente no sucesso ou fracasso das companhias interessadas nesse *market share*.

A APF é um critério bastante utilizado em tais oportunidades. Pelo lado dos órgãos licitantes, os pontos de função são utilizados para apresentar o tamanho funcional do

seu portfólio de projetos, a média histórica de produtividade das equipes, a definição de acordos de nível de serviço e qualidade, além do volume de atendimento projetado. As empresas participantes do processo licitatório avaliam a oportunidade com base nessas informações e ofertam o menor valor viável por ponto de função a ser produzido, sendo considerada vencedora a empresa que ofertar o menor valor. Desse modo, a realização de medições de tamanho funcional desempenha um papel importante para as organizações de desenvolvimento de software na medida em que tais medições consomem tempo e recursos para serem obtidas, além de serem propensas a falhas quando conduzidas manualmente. Contudo, cabe ressaltar que, quando obtida de forma correta e consistente, as medições funcionais representam uma vantagem competitiva para quem as tem.

Portanto torna-se necessário que exista uma maneira de gerar medições mais rápidas, com menos subjetividade (QUESADA-LÓPEZ; JENKINS, 2017) e com grau aceitável de variação da sua acurácia (SALEM; SOUBRA, 2023) para que a atividade de medição em si não onere o processo de obtenção dos tamanhos funcionais. Nesse sentido, a automatização da medição de tamanho funcional tornou-se relevante nos últimos anos. Diversas técnicas têm sido pesquisadas no intuito de diminuir o tempo e os custos e aumentar a confiança e repetibilidade das contagens (ABRAHãO; POELS; PASTOR, 2006; GIACHETTI et al., 2007; LAMMA, 2004; FETCKE; ABRAN; Tho-Hau Nguyen, 1998; POW-SANG et al., 2013; LENT; QU, 2015; KUSUMOTO et al., 2002; GUPTA; KAUSHAL; SADIQ, 2008; EDAGAWA et al., 2011), dentre as quais destacam-se o emprego de técnicas de análise de código (KUSUMOTO et al., 2002; EDAGAWA et al., 2011) e técnicas aplicáveis em arquiteturas de três camadas (GUPTA; KAUSHAL; SADIQ, 2008). Embora os resultados indiquem a viabilidade das contagens automatizadas, elas estão vinculadas a determinados padrões de projeto ou ferramentas de desenvolvimento, o que indica que a independência de tecnologia defendida pelos métodos FSM ainda não está sendo explorada.

1.3 Objetivos

O objetivo geral deste trabalho é identificar processos elementares por meio da análise automatizada de requisições HTTP, para assegurar a independência de tecnologia defendida pelo IFPUG e diminuir as falhas e variações decorrentes das contagens manuais. Para tal, foram definidos os seguintes objetivos específicos:

- Obter informações descritivas do software de forma automatizada, sem documentação formal prévia;
- identificar e desconsiderar requisições técnicas;

- analisar informações obtidas e agrupá-las com o objetivo em identificar jornadas dos usuários;
- analisar os agrupamentos identificados e aplicar as regras do SFP para classificá-los de acordo com os tipos de processos elementares.

Para as organizações que possuem contratos que utilizam pontos de função como métrica de referência, tal automatização poderá trazer um diferencial de mercado por permitir obter uma maior eficiência no processo de medição (QUESADA-LóPEZ, 2018) e possibilitar a obtenção de medidas consistentes de modo objetivo e confiável (Object Management Group, 2014).

1.4 Estrutura do Texto

O texto que segue está organizado da seguinte maneira. O Capítulo 2 apresenta o estado da arte e o modo como as referências foram obtidas. O Capítulo 3 apresenta a fundamentação teórica pela explicação dos temas relacionados a esse trabalho, a citar o método de contagem *Simple Function Points* (INTERNATIONAL FUNCTION POINT GROUP - IFPUG, 2021), o mecanismo para captura de dados das requisições HTTP *(web scraping)* e o método de medição funcional. O Capítulo 4 apresenta a metodologia proposta para identificação automatizada de processos elementares a partir de requisições HTTP. No capítulo 5 estão os resultados obtidos, a análise e discussões a respeito do método proposto. Por fim, o capítulo 6 apresenta as considerações finais desse trabalho e trabalhos futuros.

Capítulo 2

Revisão do Estado da Arte

Esse capítulo apresenta o estado da arte da medição automatizada do tamanho funcional em pontos de função, construído a partir de uma busca exploratória em bases de dados científicas e da análise e identificação dos trabalhos relacionados ao tema. Tendo em vista que a proposta desse trabalho é a medição automatizada de aplicações web, a *string* de busca foi construída pela conjunção dos termos sinônimos de cada um desses tópicos. Em se tratando especificamente dos termos relacionados à automação das medições, considerou-se que a automação está intrinsecamente ligada a métodos sistematizados ou formalizados por meio de um conjunto de regras e/ou foram implementados em ferramentas, o que justifica a inclusão de *tool, systematic, formal e rule* como seus sinônimos. Com relação ao termo medição, entende-se que ele é comumente substituído por medição funcional, tamanho funcional ou, simplesmente, ponto de função, o que justifica o uso desses termos como sinônimos. A *string* de busca resultante está apresentada na Figura 2.1.

(“software*” OR “WEB*” OR “system*” OR “application*”) AND
(“function* point*” OR “functional size* OR “function* measurement”) AND
(“automat*” OR “systematic” OR “procedure*” OR “tool*” OR “mapping*” OR “rule*” OR “formal* representation”)

Figura 2.1: Termos da String de busca

A *string* de busca foi executada na base de artigos Scopus[1] e resultou 131 artigos. Para filtrar os artigos de relevância para esse trabalho, restringiu-se a trabalhos em Inglês publicados nos últimos 10 anos em revistas e eventos científicos da área de Ciência da Computação, conforme demonstrado pela Tabela 2.1. Artigos que não atenderam a esse critério foram excluídos, resultando em 30 artigos.

Desse conjunto de artigos filtrados, foram incluídos na revisão bibliográfica aqueles que atendiam aos seguintes critérios:

- estudos relacionados à medição de tamanho funcional de software no campo da Engenharia de Software;

[1] https://www.scopus.com/home.uri

Período de publicação:	Últimos 10 anos (2015-2025)
Idioma:	Inglês
Área de Pesquisa:	Ciência da computação
Tipo de Documento:	Artigos completos de periódicos e eventos científicos

Tabela 2.1: Critérios de exclusão da pesquisa

- estudos relacionados à métodos de medição IFPUG e pontos de função;
- estudos que provêm alguma descrição do processo de medição funcional e,
- estudos que provêm algum detalhe da avaliação dos procedimentos de medição funcional.

Com base nos critérios, foram analisados os *abstracts* dos artigos e reduziu-se a 18 o número de artigos que satisfazem as condições pré-estabelecidas. Para melhorar o alcance da busca, aplicou-se o processo de busca *Forward Snowballing* por meio da ferramenta Google Scholar[2] e Scopus, em que se obteve 9 citações dos artigos previamente selecionados, sendo o número final de artigos utilizados como referência igual a 27.

2.1 Trabalhos relacionados

Armaly et. al (ARMALY; KLACZYNSKI; MCMILLAN, 2016) apresentam um estudo de caso com a aplicabilidade do Apache Lucene[3] junto do algoritmo Pagerank (PAGE et al., 1999) para mapear funcionalidades presentes no código-fonte de uma aplicação em pontos de função com o objetivo de prover estimativas de custo. Os autores destacam que obtiveram uma melhor precisão nos experimentos em que os dois foram combinados (75,5%) em detrimento de 53,5% de precisão quando foi utilizado apenas o algoritmo Lucene em um *dataset* independente. Limitações importantes são destacadas no referido estudo de caso como, por exemplo, bibliotecas utilitárias podem inflar as estimativas, não ser possível detectar e desconsiderar código-fonte depreciado, não ser possível detectar chamadas indiretas (chamadas realizadas por ponteiros ou funções anônimas) e, por fim, não ser possível gerar uma análise do projeto em diferentes linguagens de programação sem que um esforço extra seja necessário. No referido trabalho, utilizou-se aplicações em JAVA e C++.

Quesada-López et. al (QUESADA-LóPEZ et al., 2017) automatizaram o processo de medição de tamanho funcional para obter medidas em pontos de função em sistemas modelados pelo framework de desenvolvimento Fast Works. Uma ferramenta-protótipo para

[2]https://scholar.google.com/
[3]https://lucene.apache.org/

medição foi desenvolvida apresentando boa acurácia entre a medição realizada de forma automatizada e a manual, com uma assertividade de identificação de 95% dos processos elementares. Contudo, o referido estudo se deu para uma única aplicação, com padrão de projeto bem específico (aderente ao framework) utilizado pela organização. Uma evolução do estudo inicial foi realizada em 2019 em que outras seis aplicações foram mensuradas automaticamente pela ferramenta proposta (QUESADA-LóPEZ et al., 2019). Os resultados também demonstraram uma convergência entre as medições manuais e automatizadas sendo necessária a realização de mais experimentos, principalmente em aplicações de maior complexidade para validar o desempenho da solução. Posteriormente, Quesada-López et. al (QUESADA-LOPEZ et al., 2020) avaliaram a proposta de automatização de contagem de pontos de função baseada no framework Fast Works em modelos simplificados com o objetivo de obter estimativas de tamanho funcional de forma automática em fases iniciais dos projetos. Os resultados indicaram que medições realizadas em modelos simplificados e completos tiveram uma variação pequena (entre 0% a 3,2%).

Madrigal-Sánchez et. al (SáNCHEZ; LóPEZ; CORONAS, 2018) desenvolveu um protocolo formal para verificar a acurácia da medição do tamanho funcional do software, visando aferir o processo de medição. Esse protocolo permite a verificação do tamanho funcional medido sem depender de quem realizou a medição, seja um especialista ou uma ferramenta automatizada. A automação do processo reduz o tempo e o esforço necessários para a verificação, além de apoiar a calibragem dos procedimentos e ferramentas de medição. O trabalho avalia os resultados obtidos por meio de uma ferramenta protótipo, que demonstrou eficácia na identificação de defeitos e causas nas medições. Essa abordagem pode ser aplicada em ferramentas que automatizam o processo de contagem. A proposta consiste em comparar duas medições do mesmo escopo e identificar possíveis desvios, sendo necessário ter medições prévias para utilizar o protótipo.

Freitas et. al (JR. et al., 2019) propõem uma nova abordagem em relação ao método de medição tradicional do IFPUG que possibilita a automatização do processo de medição do tamanho funcional. A proposta inclui um subprocesso denominado *Function Point Tree-based Function Point Analysis* (FPT-FPA) cujo objetivo é a elaboração do artefato intitulado *Function Point Tree - FPT* durante a etapa de Engenharia de Requisitos. Ao produzir esse artefato todas as informações necessárias para a automatização da contagem em pontos de função são fornecidas, tais como campos das funcionalidades e regras de negócio, o que possibilitaria um mapeamento entre o FPT e as regras do IFPUG. Os resultados obtidos mostraram que o método proposto apresentou valores positivos quando executado manualmente no que se refere à reprodutibilidade e precisão. No entanto, os resultados não foram satisfatórios quando as avaliações foram executadas via protótipo que automatiza o processo, principalmente no que se refere à precisão dos dados obti-

dos. Segundo os autores, é necessário encontrar melhores ajustes às regras conceituais do método e do protótipo para garantir que todos os requisitos funcionais sejam completa e corretamente adicionados pelos usuários técnicos na FPT.

Neyveli et. al (NEYVELI et al., 2019) apresentam uma abordagem para estimar automaticamente aplicações web desenvolvidas em *Interactive Flow Modeling Language* (IFML), linguagem utilizada para design de conteúdo, interações com usuário e controle do comportamento do software. Os componentes utilizados para desenvolver a interface em IFML são mapeados em componentes de pontos de função e utilizados para estimativa de esforço. Assim, para aplicações desenvolvidas utilizando esse padrão, foi criado um método que relaciona os componentes de interface do IFML com processos elementares de pontos de função.

Alguns trabalhos baseiam suas propostas na medição à partir de Diagramas UML. Dentre eles, destaca-se o estudo empírico realizado por Lavazza e Liu (LAVAZZA; LIU, 2018) que demonstrou correlação entre medidas orientadas a objeto utilizando UML e os métodos de medição funcional do IFPUG e COSMIC (COSMIC. . . ,), o que possibilitaria a automatização do processo de medição, indicando que modelos UML podem ser usados para medições de tamanho funcional. Essa abordagem se mostrou bastante efetiva no que se refere a possíveis lacunas de subjetividade e variação nas medições, porém requer que os diagramas UML utilizados como insumo para análise contenham todas as informações necessárias para serem avaliadas do ponto de vista funcional.

Rusli e Abdullah (RUSLI; ABDULLAH, 2020) apresentam a *UML Point Tool*, uma ferramenta para medição de tamanho funcional que utiliza conceitos da análise de pontos de função em modelos UML. Essa ferramenta traduz os requisitos de aplicativos de jogos mobile em diagramas UML, permitindo o cálculo automático do tamanho funcional, com bons resultados em um estudo de caso, limitado à indústria de jogos para celular. Bluemke e Malanowska (BLUEMKE; MALANOWSKA, 2020) propõem uma combinação de fragmentos UML como base para o pré-processamento de informações no processo de medição automática de pontos de função, utilizando dados dos diagramas de classe e sequência da UML para obter informações necessárias à medição. Embora promissoras, as análises automatizadas baseadas em diagramas UML são realizadas em modelos (representações) do software estando, portanto, sujeitas à precisão e à finalidade com que os modelos representam o software em análise.

Shi et al. (SHI et al., 2020) apresentam um método para identificar funções transacionais com base em especificações de requisitos textuais, empregando técnicas de processamento de linguagem natural e modelos de Machine Learning. Os resultados indicam que essas técnicas têm grande potencial na identificação de funções transacionais, avaliando cerca de 1864 requisitos de 36 projetos e classificando 104.691 funções, com uma avaliação

positiva que demonstra boa acurácia na identificação e classificação. No entanto, para que seja possível utilizar esse método, é necessário que o sistema a ser medido tenha uma boa documentação para apoiar sua medição.

Outras estratégias baseadas em técnicas diferentes daquelas até então apresentadas, têm sido pesquisadas para automatizar a medição por pontos de função. Em comum a todas elas está o fato de buscarem benefícios como economia de tempo e custo, além do aumento da confiabilidade e repetibilidade das medições (LAVAZZA, 2015). Ersoy et al. (ERSOY; BAGRIYANIK; SOZER, 2024) investigam a precisão das estimativas de esforço baseadas na medição do tamanho funcional COSMIC realizadas na fase de desenho da arquitetura. Suchánek (SUCHÁNEK, 2023) propõe uma ontologia para modelos conceituais, visando normalizar sistemas, o que pode facilitar a comunicação e a compreensão em projetos de software e, consequentemente, servir de base para medição. Nhung et al. (NHUNG; SILHAVY; SILHAVY, 2024) explora uma metodologia para estimar pontos de função em projetos de desenvolvimento usando um modelo estruturado com o diagrama de fluxo de dados baseado nas especificações do sistema de internet das coisas (IoT). Lavazza et al. (LAVAZZA; LOCORO; MELI, 2024b) exploram o uso de aprendizado de máquina e medidas funcionais simplificadas para estimar o esforço de desenvolvimento de software, contribuindo para a eficiência na gestão de projetos. Fragoso-Díaz et al. (FRAGOSO-DÍAZ et al., 2022) realizam uma revisão sistemática sobre a qualidade em diagramas de classe, destacando sua importância para as competências em Engenharia de Software, este trabalho revisa 109 diagramas de classes abertos para identificar os defeitos mais comuns e também analisa alguns trabalhos relacionados para identificar quais são os atributos de qualidade que devem existir em diagramas de classes e as métricas usadas para avaliá-los. Santoso et al. (SANTOSO et al., 2021) aplicam a análise de pontos de função e o modelo de custo construtivo para medir um portal de aplicação comercial baseado na web. Já Kusumoto et al. (KUSUMOTO et al., 2002) apresenta uma abordagem de medição à partir do código-fonte de aplicações desenvolvidas em linguagem Java. Gupta et al. (GUPTA; KAUSHAL; SADIQ, 2008) apresentam uma abordagem para arquiteturas em três camadas como fonte para previsão do esforço do projeto de software usando o modelo de regressão linear. Por fim, Edagawa et al. (EDAGAWA et al., 2011) examinam a possibilidade de automatizar a medição por meio de análise estática de código-fonte. Muitas dessas estratégias se mostraram válidas, porém elas são direcionadas a aplicações implementadas em determinado padrão arquitetural e/ou direcionada a determinados tipos de artefatos.

2.2 Análise do Estado da Arte

Verifica-se com base nos trabalhos analisados a existência de diferentes abordagens com o objetivo de automatizar a contagem de pontos de função. Alguns trabalhos exploram a automatização da contagem de pontos de função a partir de código-fonte das aplicações (ARMALY; KLACZYNSKI; MCMILLAN, 2016; QUESADA-LóPEZ et al., 2017; QUESADA-LóPEZ et al., 2019; QUESADA-LOPEZ et al., 2020), enquanto outros buscam, por meio de padronizações através de diferentes protocolos, garantir a possibilidade de automatizar o processo de medição funcional (SáNCHEZ; LóPEZ; CORONAS, 2018; JR. et al., 2019; NEYVELI et al., 2019). Ainda, algumas abordagens utilizam a linguagem UML (LAVAZZA; LIU, 2018; BLUEMKE; MALANOWSKA, 2020; SHI et al., 2020) como referência para a padronização e automatização da contagem ou buscam fazê-lo através da especificação de requisitos tradicional (Processo Unificado), por meio de análises de linguagem natural e *Machine Learning* (RUSLI; ABDULLAH, 2020). Os estudos avaliados obtiveram bons resultados no escopo que se propuseram a avaliar, contudo são dependentes de um padrão específico, seja do código-fonte analisado, da documentação utilizada como insumo, ferramenta ou contexto específico. A Tabela 2.2 sintetiza as principais abordagens pesquisadas e suas lacunas.

2.3 Conclusões

Ao avaliar os trabalhos citados e suas abordagens, é possível identificar um desafio persistente na automação do processo de contagem de pontos de função: a necessidade de desenvolver soluções mais abrangentes, do ponto de vista tecnológico, que apresentem boa acurácia ao comparar medições automatizadas com aquelas realizadas manualmente. Essa dificuldade decorre, em grande parte, da complexidade em criar mecanismos que consigam interpretar dados em diferentes formatos, linguagens e padrões. Vários estudos demonstram resultados promissores (ARMALY; KLACZYNSKI; MCMILLAN, 2016; QUESADA-LóPEZ et al., 2017; LAVAZZA; LIU, 2018; SHI et al., 2020; NEYVELI et al., 2019), evidenciando que, uma vez que os dados estão estruturados, a aferição em pontos de função, independentemente da estratégia utilizada, apresenta boa assertividade. É importante ressaltar que esses trabalhos geralmente requerem acesso a algum artefato do software (código, documentação etc.), o que nem sempre está disponível. Portanto, uma alternativa que permita estimar o tamanho com base no próprio software representa uma lacuna a ser preenchida. Nesse contexto, acredita-se que uma solução capaz de obter e avaliar dados de aplicações heterogêneas contribuirá significativamente para a automação

Autor	Abordagem Utilizada	Lacunas
Armaly et. Al (ARMALY; KLACZYNSKI; MCMILLAN, 2016)	Automatização do processo de medição utilizando Apache Lucene e Pagerank	Dependência de determinado padrão de projeto
Quesada-López et. al (QUESADA-LóPEZ et al., 2017)	Automatização do processo de medição utilizando Framework próprio (Fast Works)	Dependência de determinado padrão de projeto
Lavazza e Liu (LAVAZZA; LIU, 2018; SHI et al., 2020)	Automatização de processos UML para aferição de PF	Sujeita à precisão e fidelidade do modelo gerado para bons resultados.
Madrigal-Sánchez et. al (SáNCHEZ; LóPEZ; CORONAS, 2018)	Protótipo para automatizar a acurácia de uma medição.	Requer inputs pré-definidos
Freitas et. al (JR. et al., 2019)	Propõe um novo padrão de especificação (FPT-FPA)	Requer documentação no padrão para medição
Neyveli et. al (NEYVELI et al., 2019)	Estimativa de aplicações web utilizando Interactive Flow Modeling Language (IFML)	Requer documentação no padrão para medição
Rusli e Abdullah (RUSLI; ABDULLAH, 2020)	Utiliza UML de aplicativos de jogos	Limitado a indústria de jogos
Bluemke e Malanowska (BLUEMKE; MALANOWSKA, 2020)	Automatização de processos UML para aferição de PF	Requer documentação no padrão
Shi et. al (SHI et al., 2020)	Processamento de linguagem natural - NLP	Requer documentação no padrão para medição.

Tabela 2.2: Abordagens e lacunas dos principais trabalhos avaliados.

do processo de contagem de pontos de função. Para tal, é necessário que as medições sejam realizadas com base em algum artefato que seja independente de formato de descrição ou linguagem/framework utilizado. De outro modo, é necessário que esse artefato seja comum às diferentes pilhas de tecnologia empregadas na construção de determinada aplicação. Nesse sentido, o foco deste trabalho é a identificação automatizada de processos elementares por meio da análise de requisições HTTP, o que torna o método independente da tecnologia utilizada ou da existência de documentação prévia do sistema, haja vista sua ampla utilização em aplicações WEB.

Capítulo 3

Fundamentação Teórica

Neste capítulo serão apresentados o método de contagem *Simple Function Points* (INTERNATIONAL FUNCTION POINT GROUP - IFPUG, 2021), utilizado como referência para medição de tamanho funcional, o protocolo HTTP (FIELDING et al., 1999) de onde são obtidos os dados utilizados para identificação dos requisitos funcionais da aplicação avaliada e o algoritmo *Longest Common Subsequence*(LCS) (FREDMAN, 1975) utilizado para encontrar a maior subsequencia de caracteres em comum entre duas cadeias de caracteres.

3.1 Método de contagem Simple Function Points (SFP)

Na perspectiva da ISO/IEC 14143-1:2007, os requisitos do usuário referentes a software podem ser agrupados em três classes principais: requisitos funcionais, requisitos técnicos e requisitos de qualidade (ISO, 2007). O segundo e o terceiro tipos de requisitos também são conhecidos como requisitos não-funcionais *(Non-Functional Requirements - NFR)* (INTERNATIONAL FUNCTION POINT GROUP - IFPUG, 2021). A medida de tamanho funcional de um software *(Functional Size Measurements - FSM)*, como o próprio nome indica, está relacionada exclusivamente aos requisitos funcionais (INTERNATIONAL FUNCTION POINT GROUP - IFPUG, 2021).

Os requisitos funcionais representam as funcionalidades disponibilizadas por uma aplicação aos seus usuários e possibilitam que eles realizem atividades e tarefas que satisfaçam suas necessidades (INTERNATIONAL FUNCTION POINT GROUP - IFPUG, 2021). São as funcionalidades que o sistema deve realizar para atender às necessidades dos usuários e dos processos do negócio. Em outras palavras, descrevem o comportamento do software: o que ele faz, quais ações aceita, quais entradas recebe, quais saídas produz e como reage a determinadas situações. Eles costumam ser expressos em termos de funções, regras de negócios, fluxos de trabalho e casos de uso, e são independentes de como o sis-

tema será implementado (tecnologia, arquitetura, etc.) (SOMMERVILLE, 2015) Existem três categorias básicas de requisitos funcionais: os requisitos que representam fluxos ou movimentos de dados, os que representam regras de processamento de dados e os relacionados ao armazenamento permanente ou persistência de dados (INTERNATIONAL FUNCTION POINT GROUP - IFPUG, 2021).

O SFP (*Simple Function Point*) é um método utilizado para a medição funcional de software, que busca oferecer uma avaliação mais simplificada e prática do tamanho de um sistema. Ele foi desenvolvido como uma versão mais acessível do método APF, reduzindo a complexidade na contagem de funções. Além disso, por ser aderente à norma ISO/IEC 20926 (ISO/IEC, 2009), o SFP é aceito pelo TCU para estimativas em pontos de função, conforme o acórdão 2362/2015 (UNIãO, 2015a). O objetivo do método SFP é fornecer uma medida objetiva do número de funções que um aplicativo de software oferece a seus usuários quantificando "o que" ele torna possível, em termos de dados disponíveis e operações realizadas sobre eles (INTERNATIONAL FUNCTION POINT GROUP - IFPUG, 2021).

O elemento principal do método SFP é o conceito de Componente Funcional Básico (CFB) (INTERNATIONAL...,). O padrão ISO/IEC 14143-1:2007 define o CFB como uma "unidade elementar de requisitos funcionais do usuário" (ISO, 2007). Neste contexto, o termo "elementar" é sinônimo de "atômico" no sentido original do termo filosófico, significando que "não pode ser mais decomposto". O CFB é a entidade elementar à qual são atribuídos valores numéricos com base na função de medição (INTERNATIONAL FUNCTION POINT GROUP - IFPUG, 2021).

O método SFP pressupõe que o valor do tamanho funcional de um software é proporcional ao número de transações e de arquivos lógicos necessários para execução de suas funcionalidades desconsiderando, portanto, os requisitos funcionais qualificados como sendo de processamento de dados como cálculos, transformações, algoritmos, dentre outros (INTERNATIONAL FUNCTION POINT GROUP - IFPUG, 2021). Desse modo, os CFBs podem ser classificados como Arquivo Lógico (*Logical File - LF*) ou Processo Elementar (*Elementary Process - EP*).

O CFB do tipo LF representa a funcionalidade que atende aos requisitos de armazenamento de dados internos e externos, ou seja, dentro ou fora da fronteira da aplicação. Em termos concretos, um LF é um grupo de dados logicamente relacionados e que são mantidos ou referenciados pelo sistema que está sendo medido. Para identificar um LF é preciso avaliar a relação lógica existente entre entidades que pertencem ao domínio da aplicação e isso pode ser feito, por exemplo, utilizando um Modelo Entidade-Relacionamento (MER). Os agrupamentos lógicos podem ser identificados com base em relações de dependência de modo que são classificadas como um único agrupamento aquelas entidades que possuem

alguma relação de dependência entre elas.

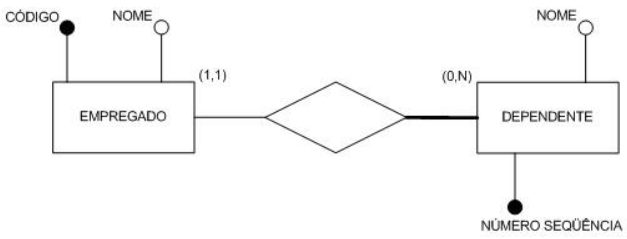

Figura 3.1: Relacionamento Empregado X Dependente

A Figura 3.1 mostra um exemplo de dependência entre entidades pois não pode haver um instância de Dependente que não esteja vinculada à uma instância de Empregado. Essas duas entidades formam, portanto, um único agrupamento lógico. Entidades também podem ser classificadas como agrupamentos lógicos distintos nos casos em que não há relação de dependência com outras entidades. A Figura 3.2 mostra uma relação entre Empregado e Projeto, cujas instâncias de uma entidade não estão necessariamente vinculadas à instâncias da outra entidade pois um empregado pode participar de vários projetos ou de nenhum projeto. O contrário também é válido pois um projeto pode ter vários empregados alocados a ele, inclusive nenhum. Nesse caso, cada entidade é considerada como um LF distinto do outro.

O CFB do tipo EP, por sua vez, representa um fluxo lógico caracterizado pela movimentação de dados. Esse tipo de CFB é a menor unidade de atividade significativa para o usuário, constitui uma transação completa, não depende de outra transação para atender ao seu objetivo (é auto-contida) e ao final de sua execução deixa a aplicação em um estado consistente (INTERNATIONAL FUNCTION POINT GROUP - IFPUG, 2021). O EP é o CFB análogo à uma jornada do usuário em que um usuário percorre um caminho ao interagir com um sistema desde o primeiro contato até a conclusão de uma tarefa. Um EP deve ser único, ou seja, deve representar uma única funcionalidade do sistema de modo a atender à premissa da unicidade do processo elementar (INTERNATIONAL. . . ,). Cada EP pode ser compreendido como uma missão ou tarefa singular realizada pelo usuário, desde sua iniciação até sua conclusão, resultando na estabilização do sistema em um estado confiável ao final do procedimento. Essa abordagem assegura que o usuário execute ações específicas de forma sequencial, sem comprometer a integridade do sistema ou gerar estados inconsistentes.

3.1.1 A medição em SFP

O processo de medição em SFP é definido como sendo uma simplificação do processo de medição em AFP e é guiado pelos CFBs do tipo LF e EP. O processo consiste de

Figura 3.2: Relacionamento Empregado X Projeto

6 atividades dispostas da maneira apresentada pela Figura 3.3. Essas atividades são realizadas por um profissional especialista no método SFP, de modo manual ou com algum suporte automatizado.

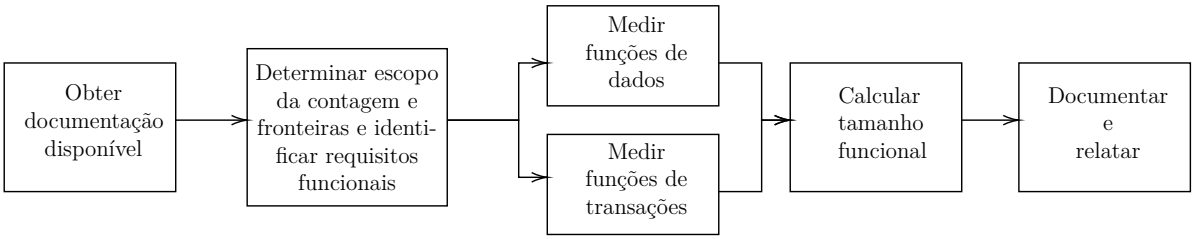

Figura 3.3: Processo de medição SFP (INTERNATIONAL. . . ,)

O primeiro passo consiste em obter toda a informação necessária para realizar a medição funcional. Artefatos como especificações funcionais, estórias de usuários, documentos de regras de negócio, modelos entidade-relacionamento e outros que descrevam as funcionalidades de um sistema que está sendo medido devem ser identificados ao final dessa etapa.

Em seguida, deve-se determinar o escopo da contagem e a fronteira da aplicação, e identificar os requisitos funcionais do usuário. A fronteira (*boundary*) de uma aplicação é a interface entre o software e seus usuários, sejam eles seres humanos ou outros sistemas. Apesar de ser, por vezes, subjetiva, a identificação da fronteira da aplicação é bastante importante pois delimita o que pertence ou não ao software sendo medido, além de capturar as interações entre o sistema e seus usuários. Nessa etapa, o escopo da medição também precisa ser bem definido pois é por meio dele que se delimitará o que será passível ou não de medição. Ressalta-se que o escopo de contagem não deve impactar a identificação da fronteira das aplicações, ou seja, a fronteira de uma determinada aplicação é independente do escopo avaliado em uma determinada contagem. Em suma, o escopo da contagem determina o que será avaliado para contagem, já a fronteira delimita o que é interno ou externo a uma aplicação. Nessa etapa, considerando-se o escopo da contagem e a fronteira delimitada, são identificados os requisitos funcionais da aplicação. Segundo Pressman (PRESSMAN, 2014), requisitos funcionais são “as funções que o sistema deve realizar, incluindo tarefas, comportamentos e respostas a eventos”. Eles representam as

ações que o sistema deve executar, como processar dados, gerar relatórios ou interagir com usuários.

O terceiro passo consiste em realizar as medições dos dois tipos de CFBs considerados pelo SFP, nomeadamente, os LFs e os EPs. Conforme apresentado anteriormente, os LFs são os conjuntos lógicos de dados que são usados de alguma maneira pelo processos elementares da aplicação e devem ser considerados de modo indiferente quanto à natureza de sua operação, isto é, não há diferenciação entre LFs acessados pelos EPs da aplicação, tanto para escrita quanto para leitura de dados. Para identificar os arquivos lógicos, é importante entender que eles representam a visão que se tem dos dados, ou seja, como os dados são organizados e acessados, independentemente de onde eles estão fisicamente armazenados. Geralmente, eles são compostos por estruturas como tabelas, arquivos, porém não há uma relação unívoca entre elas. Deve-se, portanto, analisar a relação de dependência/independência das entidades para determinar se uma ou mais tabelas compõe um ou mais LFs (INTERNATIONAL. . . ,).

A outra medição realizada no terceiro passo é a medição de funções de transação. Essa medição consiste em contar a quantidade de EPs que, como dito anteriormente, realizam a movimentação de dados pela aplicação. Para tal, é necessário identificar processos completos em um nível de detalhe que garanta que cada processo identificado represente um objetivo de negócio bem definido. As regras para identificação de EPs incorporam conceitos que representam critérios para compor ou decompor atividades de processos de negócio, atingindo a menor unidade de atividade que **seja significativa para o usuário, constitua uma transação completa, seja autocontida e deixe a aplicação em um estado consistente**. Cabe ressaltar que não é suficiente a identificação de um EP pois também é preciso garantir que após o processo elementar ser identificado não exista duplicidade em relação a outro EP semelhante identificado anteriormente, atendendo assim a premissa de unicidade do processo elementar (INTERNATIONAL FUNCTION POINT GROUP - IFPUG, 2021).

O quarto passo do método SFP é onde o tamanho funcional é efetivamente calculado, com base nos CFBs (LFs e EPs) identificados no passo anterior. Pelas regras definidas no manual do SFP, cada LF equivale a 7.0 SFPs e cada EP equivale a 4.6 SFPs (INTERNATIONAL FUNCTION POINT GROUP - IFPUG, 2021). Com base na quantidade de LFs e EPs identificados para uma aplicação, seu tamanho é dado pela soma dos pontos obtidos em cada CFB, conforme regras apresentadas em 3.1.

$$\begin{aligned} SFP_{LF} &= \sum LFs \times 7.0 \\ SFP_{EP} &= \sum EPs \times 4.6 \\ SFP_{Total} &= SFP_{LF} + SFP_{EP} \end{aligned} \tag{3.1}$$

O quinto e último passo do processo de medição consiste em realizar as atividades de documentação de todo o processo de medição. Os artefatos dessa etapa como planilhas de contagem ou relatórios técnicos devem conter todas as funcionalidades identificadas, as premissas adotadas na avaliação e qualquer outro aspecto que seja importante para o correto entendimento da contagem realizada.

3.2 Protocolo HTTP *(Hypertext Transfer Protocol)*

O *Hypertext Transfer Protocol* (HTTP) é um protocolo projetado para a transferência de conteúdo na (TANENBAUM; WETHERALL, 2011) que utiliza serviços do protocolo TCP/IP para executar suas transmissões. Ele possui mecanismos que garantem a confiança das comunicações, que o tornam um protocolo de transferência altamente confiável e um dos protocolos de aplicativos mais utilizados à ponto de servir como a base fundamental de comunicação na web (TANENBAUM; WETHERALL, 2011). Ele opera em uma arquitetura do tipo cliente/servidor em que o cliente faz solicitações ao servidor que, por sua vez, responde a tais solicitações. Em geral, essas solicitações feitas pelo cliente são para acessar recursos de diversos formatos que estão armazenados nos servidores como, por exemplo, páginas web, imagens, vídeos, dentre outros.

As mensagens trocadas entre cliente e servidor são textuais, leves e amplamente extensíveis, o que facilita a sua utilização em diversos tipos de aplicações. Embora seja orientado a conexões, o HTTP é um protocolo sem retenção de estados (*Stateless*), ou seja, ele não mantém informações sobre as interações anteriores. Além disso, as mensagens trocadas pelas requisições contêm códigos de status que revelam a natureza da resposta, permitindo que as comunicações sejam agrupadas e interpretadas de forma eficiente. (PRESSMAN, 2014; TANENBAUM; WETHERALL, 2011)

A Figura 3.4 apresenta, em alto nível, como as interações entre cliente e servidores ocorrem com o uso do protocolo HTTP. Sempre que um cliente vai solicitar algum recurso do servidor, é necessário realizar antes uma conexão com ele. Como o protocolo HTTP opera sobre o protocolo TCP/IP, a conexão com o servidor é uma conexão do tipo TCP/IP realizada, geralmente, na porta 80. Uma vez estabelecida a conexão, cliente e servidor passam a trocar mensagens entre si, sendo que tais mensagens não consideram o estados das outras mensagens. As mensagens de requisição feitas pelo cliente são sempre acompanhadas de um verbo HTTP, o recurso que se deseja acessar e a versão do protocolo HTTP utilizado. O servidor, por sua vez, envia ao cliente, em uma ou mais mensagens HTTP, o recurso solicitado ou a resposta à sua solicitação. No caso da Figura 3.4 o cli-

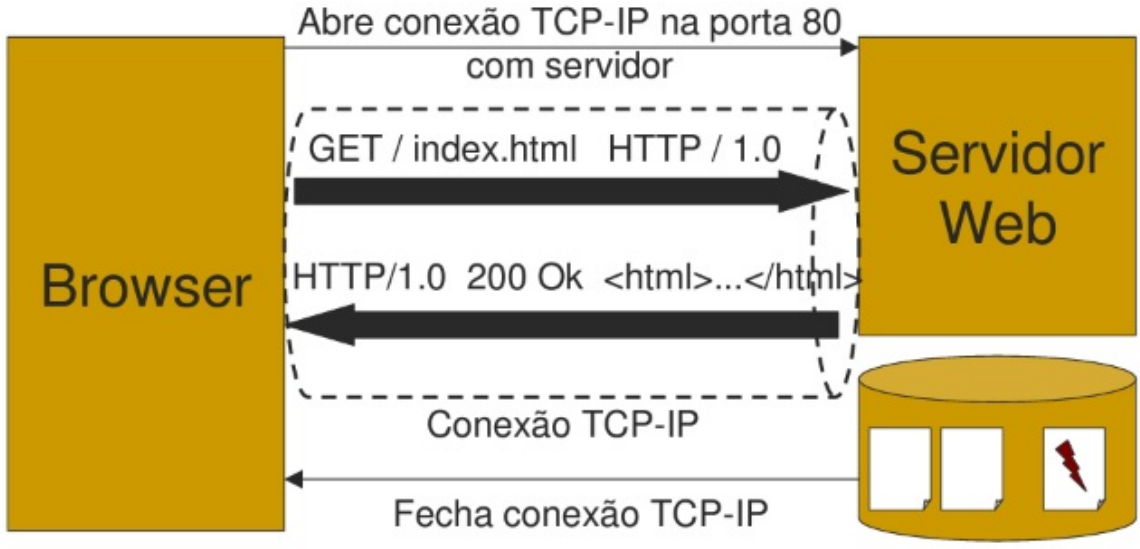

Figura 3.4: Exemplo de comunicação utilizando HTTP.

ente solicita ao servidor o arquivo `index.html` e recebe em seguida o conteúdo textual do arquivo solicitado. Ao final da interação entre cliente e servidor, a conexão é encerrada.

3.2.1 Formato das requisições HTTP

As requisições HTTP seguem uma sintaxe comum a todos tipos de requisições, é definida como um padrão pela IETF *(Internet Engineering Task Force)* (AL., 1999) e estruturada conforme ilustrado pela Figura 3.5. A primeira linha de uma requisição HTTP, conhecida como linha de comando, é onde são definidos o verbo HTTP utilizado, o recurso solicitado e a versão do protocolo em uso. Dentre todos os verbos definidos para o protocolo, os mais comuns de serem utilizados são o GET, para solicitar recursos ao servidor, e o POST e PUT, ambos para enviar dados ao servidor.

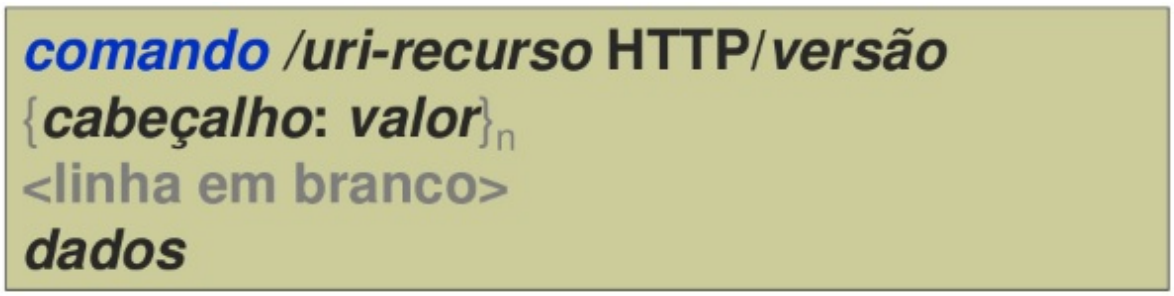

Figura 3.5: Sintaxe geral de uma requisição HTTP.

O recurso a ser requerido pelo cliente é conhecido como URI (*Uniform Resource Identifier*) cujo formato é uma sequência da caracteres que identifica de forma única um recurso na internet. O URI pode ser sub-dividido em URL (*Uniform Resource Locator*), utilizado para identificar a localização específica de um recurso na rede – um endereço web, por exemplo–, e URN (*Uniform Resource Name*), utilizado para nomear o recurso a ser acessado. A URI também pode incluir parâmetros adicionais que são passados ao servidor, sendo estes especificados por pares do tipo "chave=valor". Para distinguir o endereço da URI e seu conjunto de parâmetros utiliza-se o caractere "?" e, para diferenciar os atributos utiliza-se o caractere "&" (MASINTER; BERNERS-LEE; FIELDING, 2005). A Figura 3.6 apresenta um exemplo de URI que, para acessar o recurso "http://www.xpto.org/index.html" são passados dois parâmetros adicionais (dados1 e dados2) com seus respectivos valores.

http://www.xpto.org/index.html?**dados1=valor1**&**dados2=valor2**

Figura 3.6: Exemplo de URI.

As linhas imediatamente seguintes à linha de comando são utilizadas para definições de cabeçalhos das mensagens. Os cabeçalhos desempenham um papel crucial no controle da comunicação entre o cliente e o servidor, fornecendo detalhes essenciais para que o servidor compreenda e processe a requisição adequadamente, tais como método HTTP (que indica a ação desejada), URL (endereço do recurso desejado), versão do protocolo, *host* (específica do domínio do servidor), dentre outras. Em seguida vem uma linha em branco cuja função é separar o início (linha de comando e cabeçalhos) do corpo da mensagem que, em termos práticos, são os dados que compõem a mensagem e que são trafegados entre cliente e servidor (BUNGART, 2017).

A especificação do protocolo HTTP (RFC822...,) define oito possíveis tipos de requisições que uma aplicação cliente pode realizar ao servidor. Esses tipos estão listados na Tabela 3.1 e são apresentados em detalhe a seguir (METSCH; EDMONDS et al., 2011).

As solicitações de recursos (operações de leitura) são realizadas por meio dos verbos GET e HEAD. O GET é utilizado para solicitar qualquer recurso disponível para acesso no servidor como, por exemplo, páginas HTML, imagens, vídeos, áudios, entre outros tipos de documentos. O HEAD serve para obter os mesmos cabeçalhos que seriam retornados em uma solicitação GET, mas sem incluir o conteúdo do recurso no corpo da resposta. Esse comando é especialmente útil quando o objetivo é apenas acessar informações de cabeçalho, como tipo e tamanho do conteúdo ou código de status, contribuindo para a economia de largura de banda e tempo de transferência, especialmente em casos onde o corpo da resposta é extenso (METSCH; EDMONDS et al., 2011).

Verbo	Significado
GET	Solicita um recurso específico ao servidor. Seu retorno contém apenas dados.
HEAD	Solicita um recurso específico de modo idêntico ao GET. Seu retorno contém apenas o cabeçalho da resposta.
POST	Envia dados para inserção de novos recursos no servidor. O recurso é criado no servidor após a execução.
PUT	Similar ao PUT, contudo o recurso no servidor já é conhecido previamente e seus novos dados são enviados na requisição.
DELETE	Remove um recurso específico no servidor.
CONNECT	Estabelece um túnel de conexão com o servidor identificado como hospedeiro do recurso desejado.
OPTIONS	Utilizado para descrever as opções de comunicação com o servidor hospedeiro do recurso desejado.
TRACE	Utilizado para executar testes de chamada com o servidor de destino.
PATCH	Aplica modificações parciais em um recurso no servidor de destino.

Tabela 3.1: Verbos HTTP e seus significados.
Adaptada de (METSCH; EDMONDS et al., 2011).

Operações de escrita em recursos são realizadas pelos verbos POST e PUT. O comando POST é empregado para enviar dados ao servidor, sendo particularmente útil em situações como o *upload* de arquivos ou o envio de dados de formulários HTML. Em geral, os recursos enviados através do comando POST são novos recursos que serão inseridos no servidor. Diferentemente do GET, os dados enviados via POST não são visíveis na URI pois são transmitidos no corpo da requisição. Por sua vez, o comando PUT é utilizado para atualizar ou criar um recurso em um URI específico e já conhecido por meio de seu identificador. O comando DELETE é utilizado para solicitar a remoção no servidor de um determinado recurso, cujo identificador foi passado junto à requisição. Embora o resultado da operação seja a eliminação do recurso, essa operação é considerada uma operação de escrita (METSCH; EDMONDS et al., 2011).

Outros comandos são definidos para tarefas que não estão diretamente relacionadas aos recursos, mas que são indispensáveis ou auxiliares na comunicação com o servidor. O comando CONNECT tem a função de estabelecer uma conexão TCP com um servidor através de um *proxy*. Nesse caso, o cliente envia uma solicitação CONNECT ao *proxy*, especificando o endereço do servidor que hospeda o recurso (*host*) e a porta de destino a ser utilizada. Se a conexão for bem-sucedida, o *proxy* responde ao cliente indicando que a comunicação foi estabelecida. O método OPTIONS, definido na especificação RFC 7231 (IETF RFC 7231, 2014) do protocolo HTTP/1.1, desempenha um papel fundamental na comunicação entre clientes e servidores na web. Sua principal função é permitir que o

cliente obtenha informações sobre as capacidades do servidor ou de um recurso específico, sem a necessidade de realizar uma operação que altere o estado do recurso. Essa característica torna o método OPTIONS uma ferramenta essencial para a descoberta de funcionalidades e para a implementação de políticas de segurança, especialmente em contextos de CORS (Cross-Origin Resource Sharing). Por fim, o comando TRACE é utilizado para diagnosticar o caminho de uma requisição através da rede. Quando um cliente envia uma solicitação TRACE a um servidor, este deve ecoar a solicitação de volta, permitindo que o cliente identifique quaisquer alterações que possam ter ocorrido ao longo do trajeto (METSCH; EDMONDS et al., 2011).

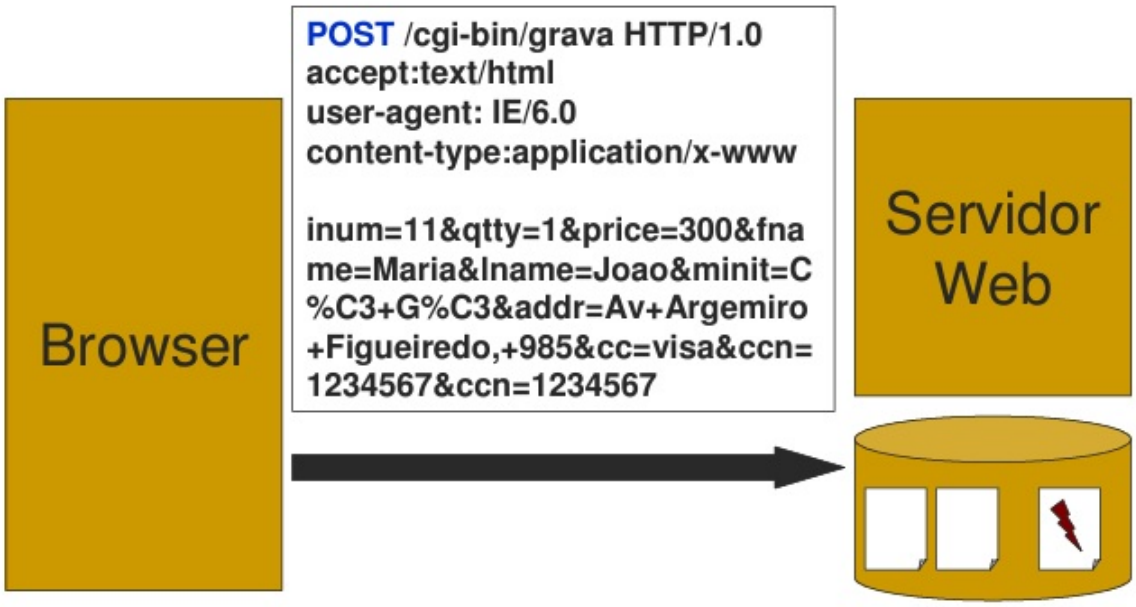

Figura 3.7: Exemplo de uma requisição HTTP do tipo POST

A Figura 3.7 apresenta um exemplo de uma mensagem HTTP. A primeira linha (linha de status) identifica, como primeiro elemento, o verbo HTTP da requisição, sendo nesse caso uma requisição do tipo POST para envio de dados ao servidor. Em seguida, estão definidos o nome do recurso a ser acessado (/cgi-bin/grava) e a versão do protocolo HTTP utilizado (HTTP/1.0). As três linhas seguintes correspondem ao cabeçalho da mensagem e trazem informações importante sobre a mensagem. Elas informam que o tipo de conteúdo aceito como resposta é uma página web (accept:text/html) e que a aplicação cliente que fez a requisição é um navegador Internet Explorer versão 6.0 (user-agent: IE/6.0). Ressalta-se que o agente do usuário pode ser qualquer aplicação que implemente um cliente HTTP como, por exemplo, navegadores web, rastreadores, *scripts* automatizados, dentre outros. O último item do cabeçalho define que o tipo de conteúdo que está sendo enviado na mensagem é no formato de uma aplicação WWW (content-type:application/x-www). No

caso de outros tipos de mídias, como fotos e vídeos, o valor desse campo seria diferente. Em seguida aparece a linha em branco que marca o fim do cabeçalho e o início do conteúdo da mensagem trafegada.

3.2.2 Formato das respostas HTTP

Código	Grupo	Descrição
1XX	Informacional	Mensagem de informação, refere-se a informações provisórias, indica que a requisição foi recebida e o processo continua. Exemplos comuns: 100 (continua) e 101 (mudança de protocolo).
2XX	Sucesso	Refere-se a mensagens de sucesso: a requisição foi recebida, entendida e aceita com sucesso pelo servidor. Exemplos comuns: 200 OK, 201 Created, 204 No Content.
3XX	Redirecionamento	Redirecionamento da requisição para outro servidor/recurso.Exemplos comuns: 301 Moved Permanently, 302 Found, 304 Not Modified
4XX	Erro do cliente	Requisição não pode ser atendida devido à erro na requisição por parte do cliente. Exemplos comuns: 400 Bad Request, 401 Unauthorized, 403 Forbidden, 404 Not Found, 408 Request Timeout.
5XX	Erro do servidor	Requisição não pode ser atendida devido à erro na requisição por parte do servidor. Exemplos comuns: 500 Internal Server Error, 502 Bad Gateway, 503 Service Unavailable, 504 Gateway Timeout.

Tabela 3.2: Grupos de status de requisições HTTP.
Adaptada de (METSCH; EDMONDS et al., 2011).

As respostas aos comandos HTTP também possuem uma sintaxe geral (RFC822...,). A primeira linha é chamada linha de status e é composta por um código do status com três dígitos que indica o resultado da requisição. Cada código de status tem um significado bem específico, contudo, eles são agrupados conforme o tipo de retorno da mensagem. A especificação do protocolo HTTP (RFC822...,) define cinco grupos de mensagens (METSCH; EDMONDS et al., 2011), conforme apresentado pela Tabela 3.2. Em resumo, esses grupos têm como propósito agregar as mensagens que são meramente informativas (Grupo 1), que indicam os diferentes tipos de sucesso ou de redirecionamentos realizados durante o atendimento da requisição (grupos 2 e 3, respectivamente), além de diferenciar as mensagens de erros como sendo erros de origem na aplicação do cliente ou no servidor (grupos 4 e 5, respectivamente).

As mensagens apresentam, imediatamente após a linha de status, os cabeçalhos de resposta que fornecem informações adicionais sobre a resposta ou sobre o servidor. Os cabeçalhos de resposta HTTP são informações que o servidor envia ao navegador após receber uma solicitação. Eles ajudam a entender o que está sendo enviado e como o navegador deve lidar com ele (METSCH; EDMONDS et al., 2011).

A última parte de uma mensagem de resposta HTTP é o conteúdo do recurso solicitado. Esse conteúdo está localizado no corpo da mensagem e pode ser representado como HTML, JSON, XML, ou qualquer outro formato de dados que esteja sendo solicitado. Embora o conteúdo das mensagens HTTP sejam textuais, as respostas das requisições podem ser de diversos tipos ou formatos como, por exemplo, textos, imagens, ou outros. Imagens, vídeos e sons podem ser transformados em textos codificados (como Base64) para serem enviados pelo HTTP. Assim, mesmo sendo arquivos binários complexos, eles podem trafegar pela rede como textos, garantindo que sejam recebidos corretamente e possam ser exibidos ou reproduzidos no dispositivo do usuário (METSCH; EDMONDS et al., 2011).

A Figura 3.8 apresenta um exemplo de uma resposta HTTP a uma requisição. Os cabeçalhos contém os metadados enviados juntos com a resposta da requisição HTTP e fornecem informações adicionais sobre a mensagem ou adicionam lógica extra sobre como o cliente deve fazer solicitações subsequentes. No caso do exemplo, o item de cabeçalho *Server* inclui informações sobre o software do servidor (no caso um servidor Apache), enquanto o item *Date* indica quando a resposta foi gerada. Também há informações sobre o recurso retornado, como seu tipo de conteúdo *(Content-Type)*, ou como ele deve ser armazenado em cache *(Cache-Control)*. Os cabeçalhos de representação são incluídos quando a mensagem possuir um corpo e servem para descrever a forma dos dados da mensagem e qualquer codificação aplicada. Por exemplo, o mesmo recurso pode estar formatado em um tipo de mídia específico, como XML ou JSON, localizado para uma determinada língua escrita ou região geográfica, e/ou compactado ou codificado de outra forma para transmissão. Isso permite que o destinatário entenda como reconstruir o recurso como era antes de ser transmitido pela rede. No caso do exemplo em questão, trata-se de uma mensagem textual, em formato HTML.

3.3 Algoritmo de Maior Subsequência Comum

O algoritmo *Longest Common Subsequence* (LCS) é amplamente utilizado em Ciência da Computação para resolver problemas relacionados a encontrar a subsequência comum mais longa entre duas ou mais sequências de caracteres (JORGE, 2023). Ele é bastante utilizado na área de processamento de textos para realização de atividades como comparação

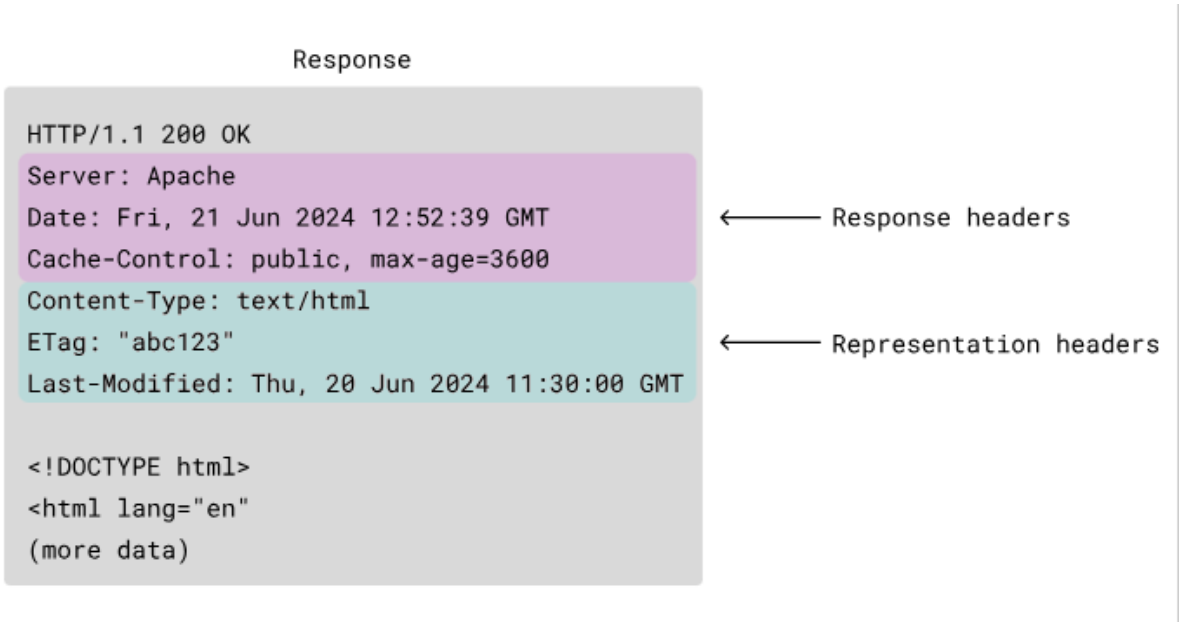

Figura 3.8: Exemplo de sintaxe de uma resposta HTTP

de strings, edição e revisão de documentos, sistemas de detecção de plágio, compressão de dados, recuperação de informações e reconhecimento de padrões (SUGUIMOTO; PAULA, 2009). Outras áreas que empregam a tarefa de identificação de semelhanças entre conjuntos de dados também se beneficiam do uso do LCS. O exemplo mais significativo está na área da Bioinformática, em que se busca identificar a semelhanças existentes entre seqüências de DNA/RNA e proteínas (BERGROTH; HAKONEN; RAITA, 2000).

Nesse contexto, a comparação de sequências em busca de similaridades é uma das tarefas importantes na identificação de relações de homologia, ou seja, de mesma descendência entre seres (NEEDLEMAN; WUNSCH, 1970). Essa comparação pode ser realizada par a par em um aminoácido de cada proteína ou através da busca pela maior sequência de aminoácidos de uma proteína que pode coincidir com outra (NEEDLEMAN; WUNSCH, 1970). Essa tarefa em biologia molecular é chamada de alinhamento de sequências e, para realizar tal alinhamento, emprega-se massivamente o algoritmo LCS, dado que os tamanhos das cadeias de aminoácidos comparadas entre si são muito grandes (COSTA et al., 2021). A Figura 3.9 mostra um exemplo de alinhamento de duas seqüências de DNA, formadas pelos aminoácidos Adenosina(A), Citosina(C), Timina(T) e Guanina(G). Os caracteres destacados nas duas primeiras seqüências são os caracteres em comum e que formam a terceira sequência (sequencia resultante de 11 caracteres). Portanto, o uso de LCS é eficaz e eficiente na comparação de similaridade entre grandes cadeias de dados.

A utilização do LCS tem como efeito secundário a identificação de similaridade estrutural em strings que representam textos semi-estruturados (NAVARRO, 2001). Textos semi-estruturados são compostos por campos cujos valores não possuem uma estrutura

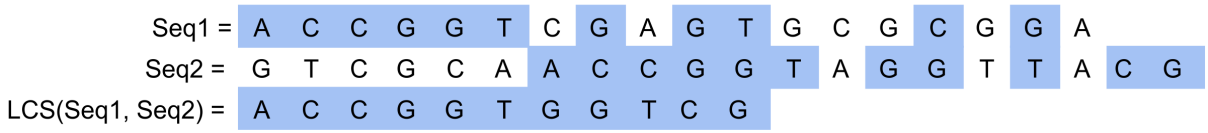

Figura 3.9: Exemplo de alinhamento de sequência de caracteres usando o LCS.
Fonte: o autor.

rígida ou definida. Isso significa que, ao contrário de dados totalmente estruturados, como tabelas relacionais, os textos semiestruturados apresentam uma organização flexível, onde os elementos podem variar em formato e conteúdo, permitindo uma maior adaptabilidade na representação de informações (ABITEBOUL; HULL; VIANU, 2011). Por similaridade estrutural entende-se a comparação da estrutura de dois ou mais registros com base nos campos que eles contém em comum e de modo independente dos valores desses campos. Considerando que os campos seguem uma ordem específica, isto é, eles aparecem na mesma ordem em todos os registros em que são usados, tem-se como efeito que a string resultante do algoritmo LCS vai conter, necessariamente, os campos em comum.

A Figura 3.10 mostra um exemplo simples de comparação de dois registros em JSON. O primeiro registro (a) é composto pelos campos nome e matricula, enquanto o segundo registro (b) é composto pelos campos nome, sexo e matrícula. Na string resultante da execução do LCS entre os registros estão contidos os caracteres em comum e, dentre eles, nota-se a presença dos campos compartilhados pelos registros, no caso, nome e matrícula, destacados em negrito.

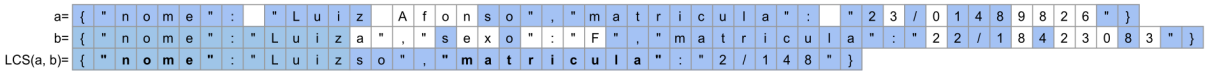

Figura 3.10: Exemplo de alinhamento de seqüência de caracteres usando o LCS.
Fonte: o autor.

O exemplo da Figura 3.10 ilustra a capacidade do algoritmo LCS de, ao mesmo tempo, calcular o grau de similaridade em número de termos iguais em sequencia entre duas strings e capturar os campos de dados semiestruturados em comum dentro da subsequência resultante. Ao empregar o algoritmo nas sequências A e B, de comprimentos 49 e 52 caracteres respectivamente, a subsequência resultante tem tamanho igual a 36 caracteres. Contidas nessa sequencia resultante, destaca-se as subsequências `"nome"` e `"matricula"` as quais, por serem delimitadas pelo caractere de aspas (`"`), estão relacionados a campos de dados representados em JSON. Considerando a premissa que os campos de dados semiestruturados aparecem seguindo uma ordem (não variam sua posição em strings diferentes), o algoritmo LCS consegue, indiretamente, capturar a semelhança estrutural entre duas strings de texto expressas segundo determinado padrão de representação.

3.4 Conclusões

Esse capítulo tratou dos temas sobre os quais esse trabalho se fundamenta. Inicialmente foi apresentado o método de contagem *Simple Function Points* e seus componentes funcionais básicos, com destaque para os processos elementares (EPs). Os EPs são elementos aos quais estão associadas as funcionalidades disponibilizadas pelo software através de suas fronteiras e que são a principal fonte para a medição do tamanho do software em pontos de função. Posteriormente, o protocolo HTTP foi apresentado em seu funcionamento e formato de mensagens trafegadas entre sistemas. Por se tratar de um protocolo de comunicação, o HTTP é independente de tecnologia e é utilizado por diferentes linguagens e frameworks de desenvolvimento garantindo, portanto, a interoperabilidade de sistemas heterogêneos. Suas mensagens são representadas em formato textual e possuem uma estrutura flexível de campos chave-valor, variável conforme o tipo da requisição HTTP realizada. Por fim, apresentou-se o algoritmo LCS que, de maneira eficaz e eficiente, calcula uma *string* formada por uma sequência de caracteres em comum entre duas ou mais *strings*. Esse algoritmo, quando aplicado em formatos semi-estruturados, tem como efeito adicional a identificação dos campos em comum entre as *strings* comparadas.

Em resumo, se por um lado a ampla utilização do protocolo HTTP por diferentes linguagens e frameworks de desenvolvimento atende à premissa de independência de tecnologias defendida pelo método SFP, por outro, suas mensagens textuais podem ser analisadas por algoritmos de manipulação de *strings* como o LCS. Nesse contexto, a análise de mensagens HTTP por tais algoritmos pode auxiliar na medição de aplicações web ao automatizar (ao menos em parte) as etapas previstas no método SFP.

Capítulo 4

Método de identificação de processos elementares a partir de requisições HTTP

Considerando a premissa de independência de tecnologia defendida pelo SFP, as atividades relacionadas à medição de software devem se basear em informações das aplicações sem a necessidade de conhecer seus projetos em detalhe. Isso significa que tais atividades devem ser agnósticas em relação a tecnologias e por esse motivo devem, o quanto for possível, extrair informações a partir de artefatos que sejam comuns a diferentes linguagens ou frameworks. O protocolo HTTP, como visto na Seção 3.2, é adotado amplamente pelas aplicações WEB e configura-se como uma linguagem comum (no sentido que o formato de todas suas mensagens é definido pelo padrão RFC 2616 (AL., 1999)) adotada por diferentes fornecedores de tecnologias. Portanto, a utilização das requisições HTTP como fonte de informações para a automatização das atividades de medição apresenta-se como uma alternativa viável e que satisfaz o requisito de independência de tecnologias defendida pelo SFP.

O método proposto nessa pesquisa busca automatizar uma parte do SFP ao propor que a identificação de processos elementares seja realizada com base em dados extraídos de requisições HTTP. Para tal é necessário capturar de maneira automatizada a sequencia de requisições HTTP envolvidas na execução de uma aplicação WEB, analisar tais sequências de modo a identificar subconjuntos de requisições recorrentes e, por fim, classificá-los como processos elementares segundo as regras do SFP. Essa análise automatizada de requisições também deve garantir que requisições consideradas técnicas sejam descartadas e que requisições repetidas sejam consideradas uma única vez, para satisfazer o requisito de unicidade de processos elementares definido pelo SFP.

4.1 Método proposto

O método para a identificação de processos elementares é composto de 4 atividades realizadas sequencialmente de modo que os artefatos resultantes de uma atividade são passados como entradas para a atividade imediatamente seguinte. A estrutura do método está representada pela Figura 4.1. Nessa representação, os elementos destacados com ícones em amarelo representam as atividades e os demais elementos representam os artefatos produzidos e consumidos ao longo da execução do método.

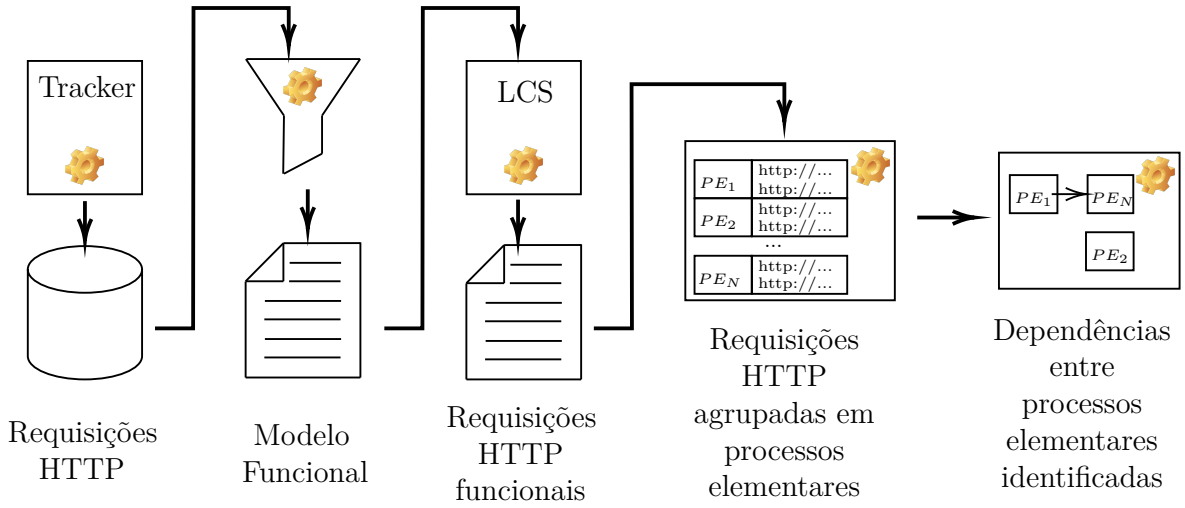

Figura 4.1: Método de identificação de processos elementares.

Em termos gerais, a execução do método para a identificação de processos elementares de uma aplicação WEB ocorre da seguinte maneira. Inicialmente todas as requisições HTTP da aplicação, independentemente de seu tipo (verbo), são capturadas por meio de uma ferramenta denominada *Tracker* e armazenadas em um repositório, sem sofrer nenhum tipo de análise ou transformação em seu conteúdo. Após a captura, tais requisições são filtradas de forma manual da ferramenta *tracker*, por meio de um interface web Figura 4.4, de modo a separar as requisições consideradas como funcionais das requisições consideradas como técnicas. Tal separação é necessária para atender ao preceito do SFP que diz que aspectos técnicos não são considerados para efeitos de medição. Após essa filtragem tem-se como resultado o modelo funcional do software que, em teor, é um artefato composto apenas por requisições HTTP de caráter funcional. Em seguida, o método emprega o algoritmo LCS para identificar as requisições que são estruturalmente semelhantes sendo que tal semelhança se dá, em sua maior parte, pelos verbos, pelas URLs e pelos nomes dos parâmetros que estão contidos nas mensagens HTTP. O resultado do LCS é um artefato formado por agrupamentos de requisições semelhantes, de modo que cada agrupamento é considerado como um PE. A última etapa do método consiste na identificação das relações de dependências entres os agrupamentos de requisições que significa,

em última análise, as dependências entre os PEs identificados. A maneira detalhada da execução de cada uma dessas etapas está descrita nas seções seguintes.

4.2 Captura de requisições HTTP

Conforme apresentado anteriormente, o objetivo da primeira etapa é capturar automaticamente as requisições HTTP de uma aplicação WEB e persisti-las em um repositório para que possam ser processadas pelas etapas seguintes do método. A ferramenta *Tracker* foi desenvolvida com o objetivo de capturar as requisições HTTP geradas por uma aplicação durante a interação dos usuários com um determinado sistema. Essa captura permite o registro detalhado do comportamento dos usuários, coletando as sequências de requisições realizadas em diferentes sessões.

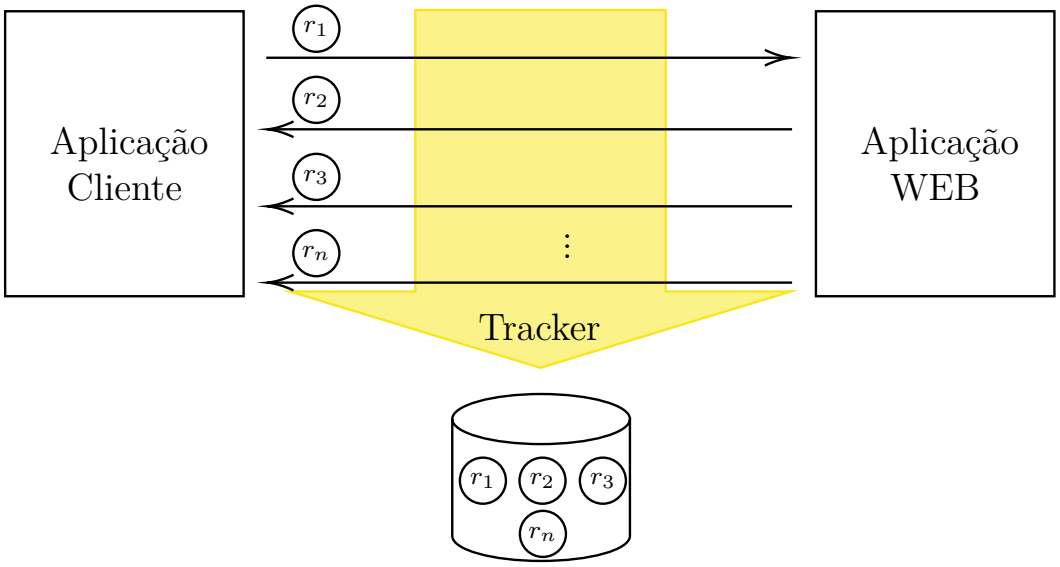

Figura 4.2: Captura de dados da requisição HTTP.

O modo como essa ferramenta opera está ilustrado na Figura 4.2. A figura mostra uma sequência de requisições HTTP trocadas entre uma aplicação cliente e uma aplicação WEB, em que cada requisição (ou respostas à requisições) está representada por uma seta indicando o sentido da mensagem (emissor $\rightarrow$ destinatário) e um índice r_n que informa seu número sequencial dentro da cadeia de mensagens. A solução Tracker, representada pela seta amarela, quando acionada passa a interceptar todas as requisições realizadas, capturar e persistir qual foi a origem da requisição (aplicativo/cliente), a URL, o tipo do método HTTP empregado (GET, POST, PUT, DELETE), o corpo da requisição e o usuário que a realizou. Com relação à origem da requisição, além do nome da aplicação que a originou, também é contabilizada a quantidade de requisições realizadas. Essas informações são extraídas do cabeçalho das requisições. Outras informações detalhadas das

requisições como, por exemplo, registro de horário da mensagem requisição (*timestamp*) e parâmetros passados e seu valores (quando existirem) são extraídas do cabeçalho ou do corpo das requisições, à depender do tipo da requisição capturada.

Cabe salientar que o monitoramento realizado pela ferramenta *Tracker* é realizado com bastante cautela. Por esse monitoramento entende-se como sendo a capacidade da solução em obter, capturar e analisar todas as requisições realizadas por usuários de uma determinada aplicação, com o único objetivo de mapear as funcionalidades existentes no escopo avaliado para aferição do tamanho funcional, sem comprometer aspectos inerentes à sua segurança e à privacidade dos dados. Para garantir a segurança e a confidencialidade das informações obtidas pelo *Tracker*, todas as informações coletadas são registradas na mesma base de dados do sistema avaliado, o que evita a transferência ou armazenamento de dados em ambientes externos. Além disso, o acesso a esses dados é rigorosamente controlado por mecanismos de autenticação e autorização, garantindo que apenas usuários autorizados possam visualizá-los ou manipulá-los. O sistema também utiliza protocolos de criptografia durante a transmissão e o armazenamento dos dados, assegurando que informações sensíveis não sejam acessadas por terceiros não autorizados. Dessa forma, o *Tracker* opera de maneira a proteger a privacidade dos usuários e a integridade das informações, alinhando-se às melhores práticas de segurança da informação e às exigências de confidencialidade.

4.3 Filtro de requisições não-funcionais

Essa etapa do método consiste de uma pré-análise das requisições capturadas pelo *Tracker* para filtrar as requisições consideradas como funcionais e descartar aquelas que são consideradas como requisições técnicas. Tal passo é necessário pois, de acordo com o SFP, somente as transações de caráter funcional devem ser consideradas efetivamente na contagem de pontos. Essa filtragem leva em consideração o teor das requisições HTTP para classificá-las como funcionais ou técnicas de modo que o conjunto de requisições funcionais componham o modelo funcional do software ao final da filtragem.

As requisições funcionais são aquelas que têm o objetivo de atender a um requisito de negócio ou que executam uma determinada tarefa ou serviço funcional para o usuário. Já as requisições técnicas estão ligadas a requisitos não-funcionais como, por exemplo, desempenho e segurança. Como exemplos desses últimos pode-se citar as requisições de validação de *tokens*, de observabilidade de aplicações, de obtenção de métricas, dentre outras. A diferenciação entre os dois tipos de requisição se dá, basicamente, pela estrutura e conteúdo das requisições analisadas. Requisições que acesssam recursos de negócio e realizam operações criação, leitura, atualização ou exclusão, são requisições funcionais. Já

requisições que acessam *endpoints* de autenticação, validação, métricas ou monitoramento, são não-funcionais. Alguns outros aspectos são avaliados para diferenciar as requisições como funcionais e não funcionais. O conteúdo do *payload* e dos cabeçalhos também auxiliam nessa diferenciação. De acordo com a RFC 7231 (IETF RFC 7231, 2014), que define o protocolo HTTP/1.1, o *payload* é a parte da mensagem que contém os dados de aplicação enviados pelo cliente ou servidor, dependendo do método utilizado. Com base nos dados presentes na mensagem é possível identificar requisições potencialmente não funcionais, sendo aquelas com *payloads* com *tokens*, *headers* de autenticação, ou dados de validação, por exemplo. Por fim, requisições com contexto de uso que suportam aspectos não-funcionais, tais como segurança, desempenho e monitoramento são identificadas nas primeiras avaliações das requisições obtidas e desconsideradas do processo de análise.

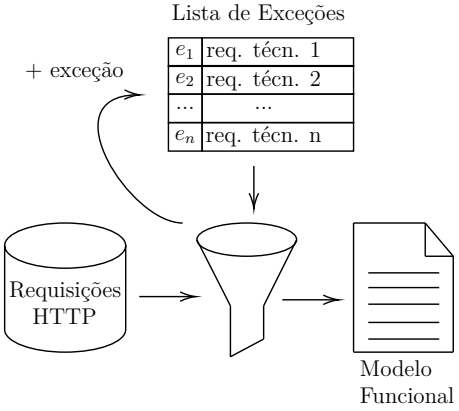

Figura 4.3: Filtro de requisições HTTP consideradas como requisições técnicas.

Com base nos critérios de diferenciação de requisições apresentados anteriormente, o método realiza a filtragem das requisições através de uma lista de exceções, conforme ilustrado pela Figura 4.3. Para cada uma das requisições capturadas pelo *Tracker*, avalia-se se ela é aderente a alguma entrada da lista de exceções. Em teor, cada entrada da lista de exceções é uma URL que passa a ser identificada como tal para todas as iterações subsequentes da ferramenta. Tal aderência se dá pela identificação da expressão regular formada pela URL e seus dados de modo que, nos casos em que há essa correspondência, a requisição é filtrada (excluída) do modelo funcional.

É importante ressaltar que a lista de exceções é um artefato que evolui à medida que novas requisições de caráter técnico são identificadas dentro do conjunto de requisições capturadas pelo *Tracker*. Sempre que nova requisição técnica é identificada, uma nova regra de exceção é adicionada à lista de exceções. Portanto, caso uma requisição atenda às características de uma requisição não funcional com base nos critérios pré-estabelecidos, esta requisição passará a ser identificada como tal e não será computada.

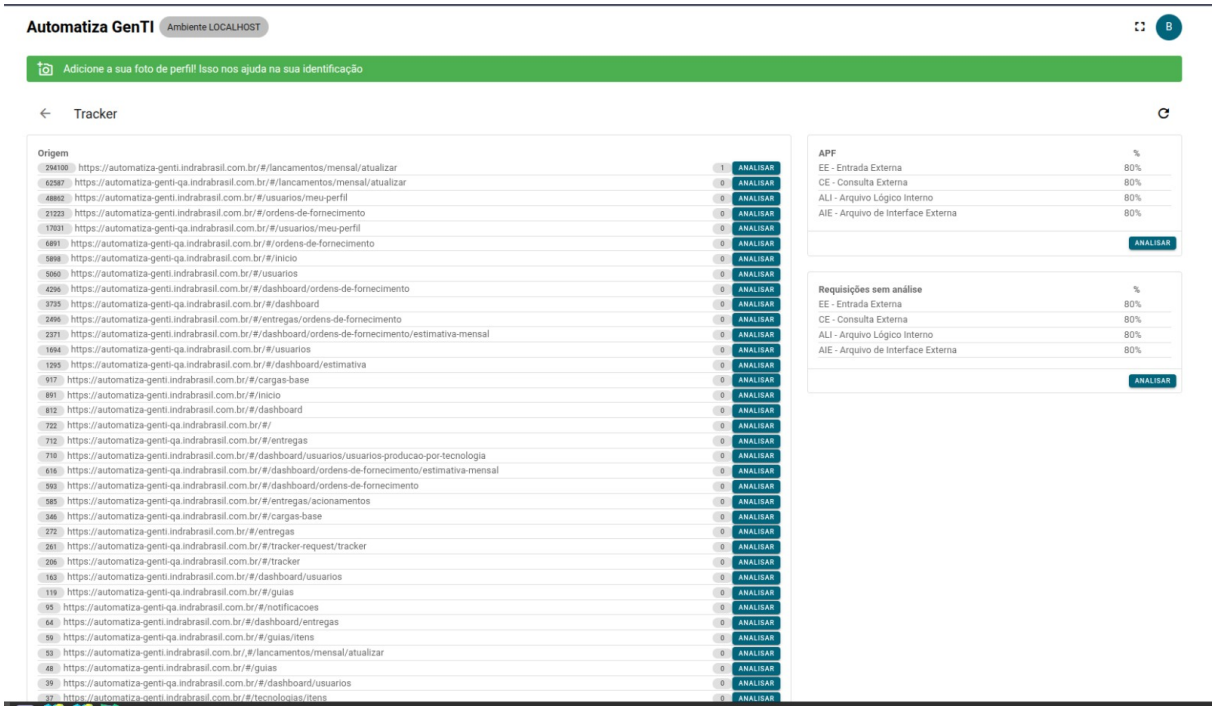

Figura 4.4: Tela inicial aplicação *Tracker*

A Figura 4.4 apresenta a tela principal da ferramenta *Tracker*, cuja operação é relativamente simples. O cliente proprietário da aplicação a ser monitorada realiza a instalação do *Tracker* em seu ambiente, e, a partir do momento em que ela é ativada, todas as requisições ao sistema sob observação passam a ser interceptadas e catalogadas automaticamente. Na interface exibida pela Figura 4.4, é possível visualizar a lista de URLs capturadas durante o monitoramento. Ao final de cada URL há um contador que indica o número de usuários distintos que acionaram aquela URL por meio da aplicação. Além disso, há a opção de identificar uma URL como técnica com base nas informações obtidas durante o monitoramento e na análise do objetivo primário do acionamento. Para isso, um analista capacitado analisa se a URL em questão foi executada para atender a um requisito não-funcional, classificando-a como técnica. Uma vez identificada como técnica, as requisições subsequentes que acionarem a mesma URL são agrupadas na lista de exceções e não passam por nova avaliação. Embora a classificação de URLs técnicas seja uma atividade manual, ela é realizada apenas no início do processo de medição de uma aplicação de modo que, após o mapeamento, essas URLs não são submetidas a novas análises. Na parte direita da tela encontram-se dois quadros: o primeiro lista os processos já avaliados e classificados de acordo com as regras do SFP, enquanto o segundo apresenta as requisições ainda não submetidas à avaliação. Essa divisão é especialmente importante em cenários nos quais novas funcionalidades são implementadas na aplicação, mas ainda não passaram por análise, facilitando o gerenciamento e a organização do processo de

avaliação.

4.4 Identificação de requisições HTTP similares

O terceiro passo do método consiste na análise de similaridade das requisições HTTP que compõem o modelo funcional do software. Essa análise é necessária para identificar as requisições que compõem os processos elementares e, principalmente, para satisfazer ao requisito de unicidade dos processos elementares definido pelo IFPUG e garantir que não haja contagem em duplicidade. Considerando que uma aplicação WEB é utilizada por diversos usuários ao mesmo tempo, espera-se que exista uma grande quantidade de requisições HTTP com similaridade considerável entre elas. Espera-se ainda que as diferenças entre as diversas requisições para uma determinada funcionalidade da aplicação com origem em interações de diferentes usuários estejam concentradas apenas nos valores dos parâmetros de cada requisição.

Intuitivamente, a equivalência de requisições HTTP pode ser definida com base na igualdade dos elementos que formam sua estrutura e desconsidera os valores de seus parâmetros pois estes representam interações específicas de cada usuário. Desse modo, as requisições que possuem a mesma origem, verbo HTTP empregado, endereço URL, método chamado e lista de nomes de variáveis são consideradas idênticas pois cada uma delas representa uma diferente execução de uma mesma funcionalidade da aplicação. Essa noção de equivalência de requisições está ilustrada na Figura 4.5.

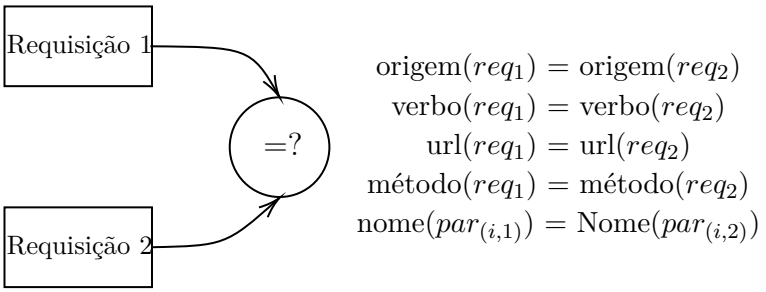

Figura 4.5: Equivalência de requisições HTTP.

Como visto na Seção 3.2, as requisições HTTP são expressas em formato texto, o que permite que algoritmos de manipulação de *strings* sejam aplicáveis em seus conteúdos. Nesse passo o método emprega o algoritmo LCS para a comparação das requisições e o cálculo de similaridades entre elas (descrito em números de caracteres sequenciais em comum), de modo que, ao final da execução do algoritmo as *strings* que apresentarem valores de similaridade aproximados entre si têm alta probabilidade de serem diferentes execuções de uma mesma funcionalidade. Como os elementos considerados para o cálculo

da equivalência (conforme Figura 4.5) e que compõem as requisições estão inseridos no conteúdo textual das mensagens HTTP, as comparações dessas requisições é, em última análise, uma comparação estrutural entre elas.

A Figura 4.6 apresenta um exemplo de 2 requisições capturadas pelo *Tracker* com as partes semelhantes destacadas em azul. O comprimento da sequencia de caracteres semelhantes entre as duas requisições é igual a 37. Dentre esses caracteres nota-se que a maior parte deles é composta pelos elementos que definem as estruturas das requisições. São eles o verbo empregado (POST), a URL (url.com), o recurso acessado (xpto) e as variáveis (var1, var2 e var3). Outros caracteres foram identificados como comuns às duas *strings*, alguns deles utilizados como marcadores de início e fim de parâmetros (?, & e =) e outros como partes dos valores dos parâmetros (m e c). Embora os valores das variáveis sejam diferentes, estruturalmente tratam-se de requisições iguais, portanto agrupadas como um requisição única.

```
a= POST url.com/xpto?var1=opq&var2=klm&var3=abc
b= POST url.com/xpto?var1=wer&var2=bnm&var3=zxc
sequence_length(a,b)=  37
```

Figura 4.6: Exemplo de similaridades entre requisições HTTP.

Convém ressaltar dois aspectos fundamentais dessa noção de equivalência de requisições HTTP e o uso do algoritmo LCS. O primeiro está relacionado à independência dos valores dos parâmetros das requisições. Requisições HTTP que possuem o mesmo formato, mesmo que sejam distintas, podem ser consideradas, em resumo, como a mesma funcionalidade atendendo a duas chamadas diferentes, variando apenas os elementos de dados passados para o processamento. Essa afirmação se justifica pelo fato de que, na arquitetura de aplicações WEB, as APIs expõem um conjunto de recursos acessíveis, onde as funcionalidades disponíveis diferenciam-se principalmente pelos parâmetros e valores utilizados em cada requisição. Dessa forma, diferentes interações de usuários ou diferentes contextos de uso podem gerar requisições semelhantes em sua estrutura, mas distintas nos dados enviados, refletindo a mesma operação ou funcionalidade subjacente. Essa distinção nos elementos de dados explica, por exemplo, como múltiplos usuários podem realizar ações similares, porém com informações específicas que diferenciam suas requisições, mantendo a consistência na estrutura da API enquanto atendem a diferentes necessidades de processamento.

O segundo está relacionado à ordem com que os nomes dos parâmetros estão dispostos nas requisições. Embora em uma chamada de método de API cada parâmetro seja passado em par do tipo chave-valor não havendo, portanto, uma necessidade de ordenamento fixo desses parâmetros, a ordem com que esses parâmetros estão listados nas requisições é

importante para o método aqui proposto. Como essa ordem de parâmetros vai ser expressa em uma *string* que representa a requisição HTTP, as ordens com que elas aparecem em todas as requisições associadas a uma mesma funcionalidade deve ser a mesma, para que não haja prejuízo no cálculo de similaridade realizado pelo algoritmo LCS.

Portanto, em resumo, a comparação de requisições HTTP feitas pelo LCS baseia-se nos elementos que compõem as requisições e deve ser, o quanto for possível, insensível aos valores de seus parâmetros. Para que tal comparação seja efetiva, pressupõe-se que existe um ordenamento fixo dos parâmetros de modo que não ocorra alternância de posição de parâmetros entre requisições similares, o que faz com que requisições que possuem estruturas muito parecidas tendem a ter valores de números de caracteres em comum muito próximos.

4.5 Identificação de Processos Elementares com base em requisições

O passo seguinte consiste na classificação das requisições HTTP funcionais como processos elementares, conforme definições do SFP. Para tal, estabeleceu-se um conjunto de regras de classificação entre o formato da mensagem e o tipo de processo elementar. É importante destacar que um processo elementar pode ser composto em uma ou várias requisições. Determinar se um processo elementar é formado por uma única requisição ou por um conjunto de requisições depende do objetivo negocial a ser atingido. Caso o objetivo seja alcançado com uma única requisição, essa requisição constitui um processo elementar isolado. Por outro lado, se a realização da jornada do usuário requer múltiplas requisições, então, possivelmente, esse processo elementar necessita da execução de todas essas requisições para que o objetivo do negócio seja atingido. Assim, a classificação das requisições em processos elementares deve considerar essa relação, de modo a refletir a complexidade e a estrutura do fluxo de interação. O conjunto de regras de classificação de requisições em processos elementares está apresentado na Tabela 4.1.

Verbo	**Regra de classificação**
POST	que possuam dados transmitidos serão classificados como processos de inclusão
POST	que possuam apenas um identificador (ID) serão classificados como alteração
PUT	que possuam dados transmitidos serão classificados como processos de alteração
DELETE	que possuam apenas um identificador (ID) transmitidos serão classificados como processo de exclusão
GET	serão classificados como consulta

Tabela 4.1: Tabela verbos HTTP e critérios de classificação

Como exemplo pode-se citar que uma requisição do tipo POST, contendo dados transmitidos através de um conjunto de variáveis, é considerada uma inclusão, contudo, será

considerada uma alteração se ela contém apenas um identificador como conteúdo. Outro exemplo são requisições do tipo GET, normalmente identificadas como processos de consulta, que podem preceder ou não outros processos, como inclusão ou alteração.

O segundo objetivo pertinente a esse passo é identificar processos elementares completos. Do ponto de vista do método SFP, a propriedade de completude está associada ao alcance de um objetivo unitário do usuário. Um processo elementar deve incluir todas as etapas de processamento obrigatórias e opcionais que são consideradas indispensáveis para atingir esse objetivo. Eles incluem os requisitos especificamente solicitados pelo usuário para concluir um processo elementar, como validações, algoritmos ou cálculos e leitura ou manutenção de uma função de dados (INTERNATIONAL FUNCTION POINT GROUP - IFPUG, 2021). Para tanto, faz-se necessário mapear as jornadas dos usuários a fim de se obter, de forma automática, processos considerados completos. Essa etapa visa agrupar uma ou mais requisições realizadas em um único processo elementar, quando essas não fazem sentido se executadas de forma separada.

Por exemplo, se for identificado que dois métodos de consulta (GET) sempre precedem um processo de inclusão (POST) é um forte indício que essas requisições só fazem sentido se executadas juntas, como ilustrado na Figura 4.7, em que para inclusão de um novo endereço o usuário sempre precisa realizar obrigatoriamente a pesquisa de um CEP.

Se o referido comportamento também for observado na navegação realizada por outros usuários, a probabilidade de tal afirmação ser verdadeira torna-se consideravelmente maior. Nesse sentido, a solução desenvolvida registra as requisições realizadas pelos usuários na aplicação e compara o comportamento de navegação dos demais usuários que acessarem a aplicação por meio do algoritmo LCS. Assim, com base em comportamentos comuns de usuários, será determinada a existência ou não de um determinado processo elementar.

Conforme apresentado na Seção 3.3, o algoritmo LCS é utilizado no reconhecimento de padrões, identificando a subsequência mais longa. Com isso, a medida que as subsequências forem identificadas, elas são agrupadas e consideradas um processo elementar único, do ponto de vista do SFP. Situações como consultas implícitas (consultas que antecedem as alterações e são contabilizadas como processos independentes), consultas para carregar combos (também identificadas como independentes), dentre outras situações que podem ocorrer de forma sequencial a outras requisições serão tratadas isoladamente e contabilizadas de forma independente, ou seja, são situações que o método proposto não foi efetivo em identificá-las da maneira correta, portanto serão tratados como itens não identificados de forma automática. A lista de exceção mencionada será retroalimentada a medida que possíveis agrupamentos identificados de forma automática pelo algoritmo forem avaliados por usuários como requisitos independentes, sendo assim contabilizados

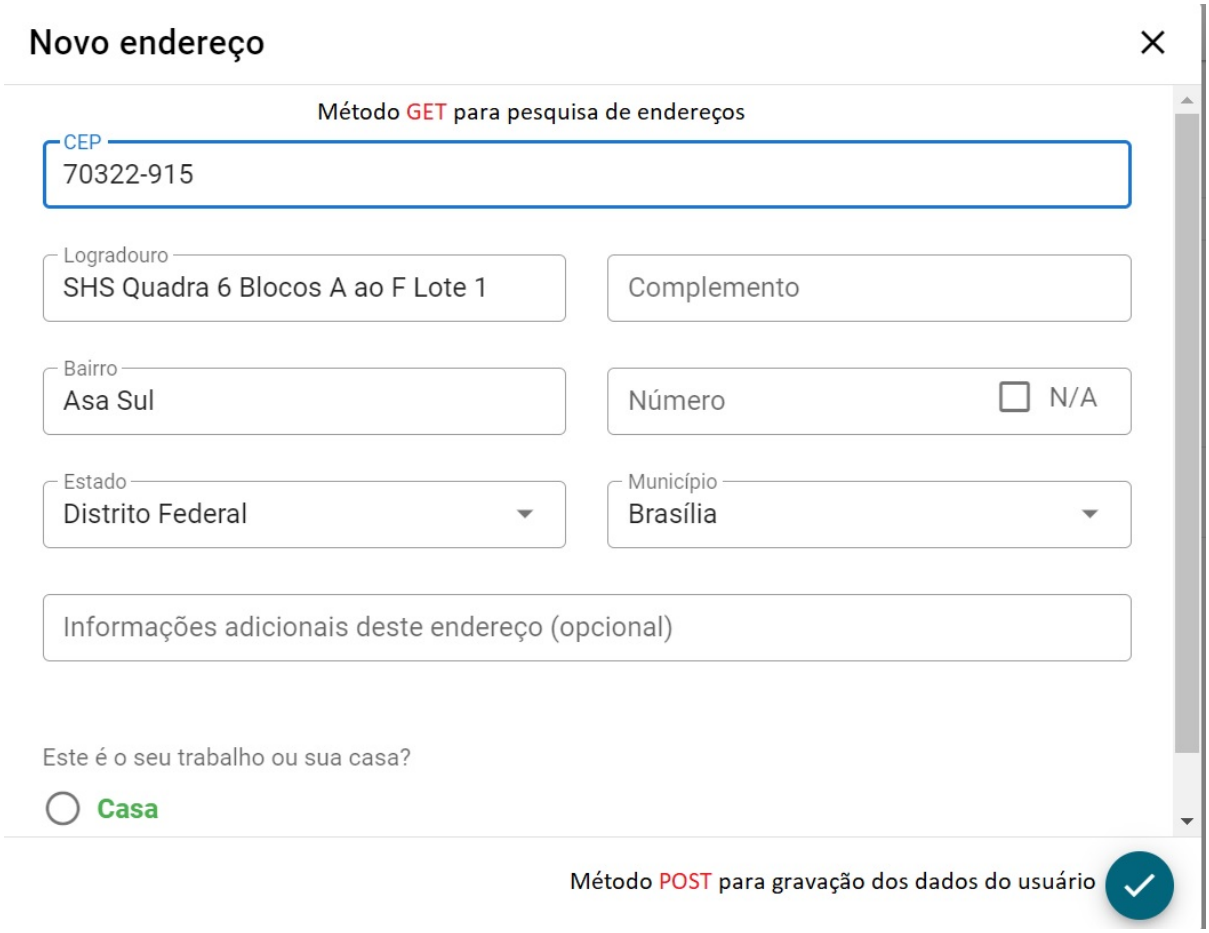

Figura 4.7: Consulta (GET) dependente do processo de inclusão (POST)

de forma separada.

4.5.1 Identificação de Arquivos Lógicos com base nas requisições HTTP

Embora a proposta concentre na identificação de PEs, é possível inferir a existência de funções de dados com base nos processos elementares identificados. O método proposto não avalia as entidades e relacionamentos da aplicação sendo medida, mas infere a existência de funções de dados com base nas características dos processos elementares identificados de forma automática. Para ser considerada uma função de dado, um arquivo lógico precisa ser reconhecido pelo usuário, ser mantido por qualquer aplicação, agrupado se for dependente de outra entidade, não se referir a um grupo de dados não funcional e não ser uma entidade que contenha atributos não requisitados pelos usuários (INTERNATIONAL FUNCTION POINT GROUP - IFPUG, 2021).

No escopo desse trabalho, os arquivos lógicos são identificados como se segue. Existindo uma requisição para incluir/alterar/excluir dados, será contabilizado um arquivo lógico interno (ALI) para o agrupamento de dados mantido. Caso uma requisição de consulta não recupere dados de um arquivo lógico interno mantido no escopo da aplicação sendo medida, este dado recuperado será classificado como Arquivo de Interface Externa (AIE), pois refere-se a um dado que não é mantido pela aplicação medida. Por exemplo, um cadastro de cliente mantido em uma tabela ou arquivo que armazena informações, como nome, endereço, telefone e histórico de compras, mantido pela aplicação para gerenciar os dados dos clientes será contabilizado como um ALI. Já os dados de um relatório, que recupera informações de um sistema externo, não mantido pela aplicação em análise, será o arquivo lógico acessado classificado como AIE.

4.6 Identificação de dependências entre Processos Elementares

Um processo elementar constitui a menor unidade de atividade significativa para o usuário, descrevendo um comportamento completo do software que não depende de ações adicionais para atingir um objetivo negocial específico (INTERNATIONAL FUNCTION POINT GROUP - IFPUG, 2021). De forma análoga, a jornada do usuário refere-se ao percurso completo realizado pelo usuário ao interagir com um produto, serviço ou sistema, desde o primeiro contato até a consecução de seus objetivos finais (LEMON; VERHOEF, 2016). Desse modo, é possível estabelecer uma relação entre ambos: um processo elementar pode ser interpretado como uma jornada do usuário.

Nesse contexto, a utilização do algoritmo LCS neste trabalho tem como objetivo identificar padrões de utilização do sistema, analisando os dados de navegação gerados pelos usuários por meio das requisições HTTP realizadas e, consequentemente, identificar as jornadas dos usuários. A identificação de dependências entre processos elementares é fundamental para evitar o *overcounting*, uma vez que não se pode estabelecer uma relação 1:1 entre uma requisição HTTP e um processo elementar. Em linhas gerais, se um conjunto de requisições é executado para atingir um objetivo negocial específico, não há justificativa para contar cada etapa desse processo como uma unidade independente. Essa abordagem só faz sentido do ponto de vista negocial se as ações forem realizadas de forma conjunta, o que caracteriza uma visão de jornada do usuário alinhada às premissas de processos elementares.

Pelo método proposto, são identificadas sequências de acessos semelhantes realizadas por parte dos usuários em uma determinada aplicação, por meio de sua utilização habitual do sistema. Com base nas sequências de acessos obtidas, por usuário, são analisadas

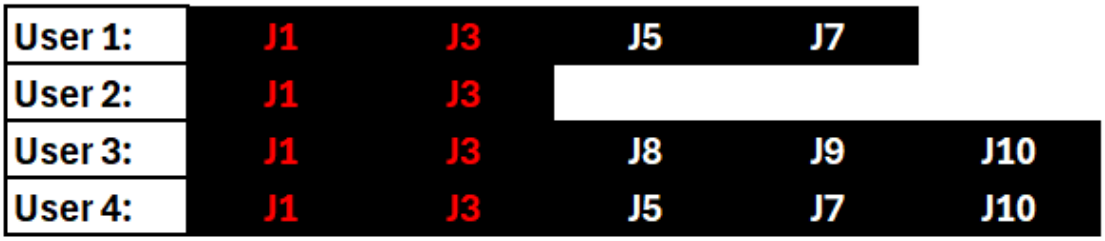

Figura 4.8: Exemplos de jornadas dos usuários em ordem de execução

as semelhanças de acesso entre os usuários, onde as semelhanças são interpretadas como jornadas do usuário e, consequentemente, como processos elementares. Em paralelo às metodologias ágeis, uma ou mais histórias do usuário podem compor um ou vários processos elementares. Quando as histórias são independentes e completas por si só — isto é, quando uma determinada história atende a um problema negocial de forma integral — essa história refere-se a um processo elementar. Contudo, em um cenário no qual um conjunto de histórias só resolve um problema negocial se todas as etapas, ou todas as histórias, forem executadas, esse conjunto de histórias do usuário comporá um único processo elementar.

Observa-se na Figura 4.8, por exemplo, que as ações **J1** e **J3** são comuns às jornadas realizadas por quatro usuários diferentes. Com o objetivo de ilustrar a análise realizada, no contexto do exemplo apresentado, as ações **J1** e **J3** seriam consideradas dependentes, por sempre ocorrerem juntas, não podendo ser consideradas como processos elementares independentes. Portanto, o presente trabalho propõe uma abordagem para implementar e executar a identificação de jornadas do usuário por meio dos dados de navegação obtidos pelo *Tracker* e, através do uso do algoritmo LCS, realizar a identificação de requisições dependentes ou não de outras requisições. Com base nessas informações, é possível realizar a identificação da dependência entre os processos elementares e identificá-los de forma correta.

4.7 Conclusões

Este capítulo apresentou detalhadamente o método proposto para a identificação automatizada de processos elementares a partir da análise de requisições HTTP, alinhando-se à premissa de independência de tecnologia do *Simple Function Points*. A abordagem capitaliza o protocolo HTTP como uma linguagem comum para aplicações WEB, permitindo a extração de informações para medição funcional sem a necessidade de conhecimento aprofundado dos projetos internos das aplicações.

O método foi estruturado em cinco atividades sequenciais e interdependentes. Inicialmente, a ferramenta *Tracker* foi desenvolvida para interceptar e persistir todas as requisições HTTP de uma aplicação WEB, coletando dados essenciais como URL, verbo HTTP, corpo da requisição e usuário, sempre com foco na segurança e privacidade dos dados. Subsequentemente, foi implementada a etapa de filtragem para remover requisições de caráter técnico, garantindo que apenas aquelas consideradas funcionais, aderentes a requisitos de negócio, componham o modelo funcional do software, utilizando uma lista de exceções evolutiva. Para satisfazer o requisito de unicidade do SFP, o algoritmo *Longest Common Subsequence* é empregado para identificar requisições estruturalmente semelhantes dentro do modelo funcional, desconsiderando, tanto quanto possível, os valores dos parâmetros e atentando-se à ordem dos mesmos. A etapa final classifica as requisições funcionais em processos elementares, utilizando regras baseadas nos verbos HTTP e no objetivo negocial, permitindo que um processo elementar seja composto por uma ou múltiplas requisições, dependendo da completude da jornada do usuário. O uso do LCS também auxilia na identificação de padrões de navegação e dependências entre processos elementares, visando evitar a contagem duplicada (*overcounting*) e refletir a complexidade real do fluxo de interação do usuário. Adicionalmente, foi explorada a inferência de Arquivos Lógicos (ALI e AIE) a partir das características dos processos elementares identificados.

Em suma, esse capítulo definiu as bases do método proposto para a automatização da identificação de processos elementares, abordando desde a coleta de dados brutos até a sua transformação em artefatos alinhados aos preceitos do SFP.

Capítulo 5

Resultados e Análises

Através do desenvolvimento do MVP *(Minimum Viable Product)*, formado pela ferramenta *Tracker* e o emprego do algoritmo LCS, cujo objetivo principal foi validar a viabilidade e a aplicabilidade do método proposto, foi possível realizar uma análise detalhada sobre a utilização de requisições HTTP como fonte de informações para a identificação de processos elementares e seu alinhamento às premissas do modelo SFP. Essa abordagem permitiu explorar de forma prática e eficaz a capacidade de captura de dados em ambientes reais e contribuiu para a validação empírica do método e das hipóteses levantadas inicialmente. Além disso, o uso do *Tracker* possibilitou a coleta contínua de informações, o que favoreceu uma análise de diferentes cenários e a identificação de padrões de comportamento dos usuários ao longo do tempo, aspectos essenciais para a compreensão aprofundada dos processos de interação com as aplicações monitoradas.

Durante a implementação e os testes do *Tracker*, observou-se, em análise preliminar, que as requisições HTTP geradas pelos usuários ao interagirem com a aplicação fornecem dados com detalhamento suficiente para mapear as jornadas dos usuários. Essas jornadas correspondem às sequências de ações realizadas por cada usuário em um intervalo de tempo, permitindo compreender o fluxo de navegação e as funcionalidades acessadas. Para cada ação, o *Tracker* atribui um identificador único, marcando a ação tanto para o usuário que a realizou originalmente quanto para os demais usuários que a executarem posteriormente. Desse modo, é possível gerar, para cada usuário, uma sequência ordenada de ações que reflete as funcionalidades acessadas ao longo do tempo. Essas sequências estruturam o percurso individual de cada usuário na aplicação, viabilizando uma análise detalhada do comportamento de navegação e do uso das funcionalidades disponíveis. A abordagem facilita a identificação de padrões de uso recorrentes e de potenciais pontos de melhoria na experiência do usuário, além de fornecer base sólida para análises comparativas entre diferentes perfis de usuários ou períodos. Para tanto, o *Tracker* foi instalado nas aplicações a serem medidas para que fosse realizada a captura das requisições. A partir

do momento da instalação, o *Tracker* passa a monitorar e armazenar os dados pertinentes a todas as requisições. Para avaliação dos resultados, deixou-se o *Tracker* habilitado nas aplicações em avaliação pelo período inicial de um mês. Cabe salientar, que não foi observada perda de desempenho siginificativa em decorrência do processamento realizado pelo *Tracker*. Posteriormente, todas sequências de ações referentes à todos os usuários, foram comparadas semanalmente utilizando o algoritmo LCS. O objetivo do LCS é identificar a subsequência comum mais longa entre várias sequências de mensagens HTTP de modo a permitir a detecção de padrões recorrentes na utilização pelos usuários da aplicação sob análise. Cada conjunto de requisições recorrentes e capturadas pelo método foram considerados como uma jornada de usuário. Essa análise é fundamental para compreender as rotinas de navegação e identificar etapas essenciais que se repetem ao longo do tempo, contribuindo para uma compreensão mais aprofundada do comportamento dos usuários e das funcionalidades mais utilizadas. Além disso, a utilização do LCS possibilita a identificação de padrões de uso que podem indicar processos subjacentes, facilitando a posterior associação com os processos elementares definidos pelo modelo SFP.

O ponto central deste trabalho foi avaliar a relação entre as subsequências identificadas — ou seja, as jornadas dos usuários — e os processos elementares que foram identificados com suporte do método baseado em requisições HTTP e nas regras de contagem do SFP. Essa análise visa verificar se as ações mais frequentes e as sequências recorrentes de fato correspondem às etapas fundamentais de um processo elementar, conforme definido pelo modelo SFP, e em caso afirmativo, validar se a abordagem proposta oferece contribuição eficaz às práticas de contagem, mantendo sua aderência ao método SFP. Para tanto, foram realizadas análises quantitativas e qualitativas, buscando estabelecer relações entre os padrões de navegação detectados e os processos elementares identificados com suporte do método automatizado. Essa etapa é crucial para validar a hipótese de que as jornadas de usuários identificadas podem ser classificadas como processos elementares.

5.1 Sujeitos, tratamentos e métricas

Para a avaliação dessa proposta, classificam-se como sujeitos os sistemas WEB que utilizam o protocolo HTTP, e os tratamentos são a identificação de processos elementares com base no método automatizado proposto nesse trabalho, bem como a identificação manual feita por um especialista em medição de pontos de função. A avaliação foi realizada com 2 aplicações distintas, a primeira utilizando o framework javascript Vue.js na camanda de apresentação e backend em Node.js (aplicação para controle de ordens de serviço) e a segunda utilizando o framework Angular para a camada de apresentação e springboot no backend (aplicação de gestão de contratos).Optou-se pelas aplicações indicadas pelo

fato de utilizarem linguagens distintas e ambas serem padrões de mercado para aplicações WEB. Para efeitos de comparação entre as abordagens manual e automatizada, foram avaliadas a identificação de processos elementares realizadas de forma manual por mais de um especialista, em relação aos processos identificados automaticamente pelo método proposto. Essa comparação visa possibilitar uma análise detalhada de eventuais desvios entre os dois métodos, bem como estabelecer o nível de assertividade dos valores obtidos automaticamente. Nesse contexto, a avaliação foi conduzida de maneira tanto quantitativa quanto qualitativa: a quantitativa refere-se à quantidade de processos elementares identificados por cada abordagem, ao passo que a qualitativa diz respeito à correção na classificação desses processos, ou seja, se as categorias atribuídas estão alinhadas com as definições e critérios estabelecidos pelo método de pontos de função. Essa análise comparativa é necessária para validar a confiabilidade do método automatizado e para identificar possíveis melhorias ou ajustes necessários na sua aplicação.

Com o objetivo de validar a eficácia do método proposto nesta pesquisa, foram definidos e estruturados três objetivos principais de modo a garantir que seus resultados obtidos sejam consistentes, reprodutíveis e relevantes para o avanço do conhecimento na área. Esses objetivos orientam toda a investigação, promovendo uma abordagem sistemática e transparente, que possibilita a replicação do processo de medição e a análise crítica dos resultados. A seguir, apresenta-se o detalhamento de cada um desses objetivos, incluindo as ações realizadas e os critérios utilizados para sua avaliação, de modo a garantir que o processo de investigação seja completo e coerente com os propósitos do estudo.

O primeiro objetivo consistiu em avaliar as informações contidas nas requisições HTTP capturadas de uma aplicação web, com o propósito de extrair conhecimento e descrever o software analisado. Essa etapa buscou eliminar a dependência de documentação prévia, frequentemente incompleta ou desatualizada, explorando exclusivamente os dados trafegados entre cliente e servidor como fonte para identificação de processos elementares. Tal abordagem visa tornar a análise mais ágil, automatizada e aplicável a sistemas legados ou com documentação limitada.

A partir da avaliação qualitativa das requisições, foi possível distinguir as requisições funcionais, associadas a ações de negócio que alteram o estado do sistema (ex.: criar, atualizar, excluir dados), das requisições não funcionais, voltadas a aspectos de desempenho, disponibilidade ou compatibilidade, que não modificam diretamente o estado da aplicação. Essa diferenciação baseou-se em critérios como verbo HTTP utilizado, efeito observado, contexto de negócio e códigos de status retornados. Requisições que exigem validações complexas ou acionam fluxos de negócio foram classificadas como funcionais, enquanto chamadas de monitoramento, cache ou telemetria foram identificadas como não funcionais.

O segundo objetivo, diretamente relacionado ao primeiro, visou identificar e descartar requisições técnicas, associadas a requisitos não funcionais (autenticação, controle de sessão, cache, etc.), que não representam processos de negócio e, portanto, não devem compor a identificação de processos elementares. Com base nos critérios definidos, a precisão inicial na identificação dessas requisições foi de aproximadamente 62% sem o contexto da aplicação, aumentando para 83% à medida que esse contexto foi incorporado. Essa evolução ocorreu devido à classificação e rotulação progressiva das requisições recorrentes, que passaram a ser automaticamente reconhecidas em análises subsequentes. O índice de falso-positivos, correspondentes a requisições funcionais indevidamente classificadas como técnicas, foi de cerca de 18%, principalmente envolvendo métodos GET voltados à recuperação de informações negociais. Não foram observados falso-negativos nas aplicações avaliadas.

O terceiro objetivo consistiu em identificar, a partir do histórico de navegação dos usuários, jornadas de uso comuns no software analisado, utilizando o algoritmo Longest Common Subsequence (LCS). Considerou-se jornada comum a sequência de ações executada repetidamente por diferentes usuários em um período, indicando padrões de comportamento ou processos de negócio recorrentes. Essa análise permitiu compreender o uso real do sistema, as funcionalidades mais acessadas e a ordem de execução das ações, além de apoiar a validação dos processos elementares identificados. Nos experimentos, o método automatizado identificou 55 processos elementares no primeiro sistema (contra 48 manualmente) e 38 no segundo (contra 32 manualmente), englobando todos os processos reconhecidos manualmente. As diferenças observadas decorreram principalmente de particularidades dos fluxos de cadastro — em que o método automatizado distinguiu blocos de ações relacionados a permissões ou etapas intermediárias — e de decomposições excessivas em processos compostos por múltiplas abas. Ainda assim, os resultados demonstram boa correspondência entre as abordagens e confirmam a consistência da identificação automatizada.

5.2 Procedimento de avaliação

A fase de avaliação da abordagem proposta nesta pesquisa tem como objetivo principal estabelecer um comparativo preciso e confiável entre a identificação de processos elementares realizado de forma manual por especialistas em pontos de função e a forma automatizada. Essa comparação é necessária para verificar a consistência, a precisão e a validade do método automatizado desenvolvido, bem como identificar possíveis limitações ou pontos de melhoria na sua aplicação. Para alcançar esse objetivo, realizou-se uma análise detalhada, que envolve a comparação direta entre os resultados da identificação de processos

elementares manual, executada por um especialista certificado na metodologia SFP, e os valores obtidos por meio da solução automatizada, apresentada no Capítulo 4.

A comparação entre os valores das abordagens automatizada e manual foi realizada item a item, ou seja, cada processo elementar identificado manualmente é confrontado individualmente com os processos detectados automaticamente a partir da análise das requisições HTTP. Essa abordagem permite uma avaliação granular, possibilitando identificar com maior precisão as semelhanças e diferenças entre os dois conjuntos de dados. No procedimento, cada processo elementar manual, ilustrado na Figura 5.1, é identificado como uma função onde lhe é atribuído um tipo correspondente a sua intenção primária, que pode ser: EE (entrada externa), CE (consulta externa) e SE (saída externa). Esses processos elementares identificados manualmente são comparados aos processos identificados pelo sistema automatizado, representados na Figura 5.2 onde é possível verificar as seguintes informações: id que refere-se a sequência de acesso realizada por determinado usuário, , id tracker que refere-se ao identificador único de uma determinada requisição realizada por um usuário, usuario id identificador do usuário, requisicao URL da requisição, metodo que refere-se ao método HTTP da requisição e corpo que possui os dados tramitidos pela requisição capturada Essa comparação é realizada de forma sistemática, registrando-se todas as correspondências encontradas, bem como as discrepâncias observadas ao longo do processo.

Planilha de contagem de ponto de função - Versão 2.0

Aplicação : Tracker			Projeto : Tracker				
Responsável :			Revisor :				
Empresa : Big Brain	R$/PF = 0		Custo= R$ 0,00			PF = 24	
Função	Tipo	(I/A/E)	TD	AR/TR	Complex.	PF	PF Local
Ordem de fornecedimento - Consultar	CE	I			Média	4	4,00
Acionamento - Consultar	CE	I			Média	4	4,00
Acionamento - Incluir	EE	I			Média	4	4,00
Acionamento - Alterar	EE	I			Média	4	4,00
Acionamento - Excluir	EE	I			Média	4	4,00
Consultar lista funcionalidades tracker	CE	I			Média	4	4,00

Figura 5.1: Processos identificados de forma manual

É possível observar a correspondência dos processos elementares obtidos de forma manual e automatiza conforme 5.3. A última linha da mesma figura apresenta uma requisição técnica, não funcional, portanto sem correspondência de processo elementar identificado de forma manual. Durante essa etapa, são considerados diversos aspectos, como a quantidade de processos identificados por cada método, a classificação atribuída a cada processo, bem como a sua relevância dentro do contexto funcional do sistema. As correspondências indicam que ambos os métodos reconheceram processos semelhantes ou idênticos, reforçando a validade do método automatizado. Por outro lado, as divergências podem indicar

id	tracker_id	usuario_id	requisicao	metodo	corpo
1	1	3	http://automatiza-backend:3000/v1/admin/u/auth/login	post	{"keyCloakToken": "eyJhbGciOiJSUzI1NiIsInR5cCIgOiAiSldUIiwia2lkIiA6ICI5
2	2	3	http://automatiza-backend:3000/v1/admin/u/auth/login	post	{"keyCloakToken": "eyJhbGciOiJSUzI1NiIsInR5cCIgOiAiSldUIiwia2lkIiA6ICI5
3	3	3	http://automatiza-backend:3000/v1/admin/ordens-de-fornecimento?page=1&size=5	get	
4	3	3	http://automatiza-backend:3000/v1/admin/u/auth/login	post	{"keyCloakToken": "eyJhbGciOiJSUzI1NiIsInR5cCIgOiAiSldUIiwia2lkIiA6ICI5
5	2	3	http://automatiza-backend:3000/v1/admin/u/auth/login	post	{"keyCloakToken": "eyJhbGciOiJSUzI1NiIsInR5cCIgOiAiSldUIiwia2lkIiA6ICI5
6	4	3	http://automatiza-backend:3000/v1/admin/acionamentos?page=1&size=5&excluidos=false	get	
7	4	3	http://automatiza-backend:3000/v1/admin/acionamentos	post	{"numero": "12334", "contrato": "Contrato 12334"}
8	4	3	http://automatiza-backend:3000/v1/admin/acionamentos?page=1&size=5&excluidos=false	get	
9	4	3	http://automatiza-backend:3000/v1/admin/acionamentos/ativar-inativar/1	put	{"st_ativo": false}
10	4	3	http://automatiza-backend:3000/v1/admin/acionamentos?page=1&size=5&excluidos=false	get	
11	4	3	http://automatiza-backend:3000/v1/admin/acionamentos/1	delete	
12	4	3	http://automatiza-backend:3000/v1/admin/acionamentos?page=1&size=5&excluidos=false	get	
13	5	3	http://automatiza-backend:3000/v1/admin/u/auth/login	post	{"keyCloakToken": "eyJhbGciOiJSUzI1NiIsInR5cCIgOiAiSldUIiwia2lkIiA6ICI5
14	5	3	http://automatiza-backend:3000/v1/admin/tracker/all	get	
15	5	3	http://automatiza-backend:3000/v1/admin/tracker/all	get	
16	5	3	http://automatiza-backend:3000/v1/admin/tracker/all	get	
17	5	3	http://automatiza-backend:3000/v1/admin/tracker/all	get	

Figura 5.2: Processos identificados com base em requisições HTTP

Processo elementar identificado de forma manual	Processo elementar identificado de forma automatizada]	Verbo HTTP
Ordem de fornecedimento - Consultar	http://automatiza-backend:3000/v1/admin/ordens-de-fornecimento	GET
Acionamento - Consultar	http://automatiza-backend:3000/v1/admin/acionamentos	GET
Acionamento - Incluir	http://automatiza-backend:3000/v1/admin/acionamentos	POST
Acionamento - Alterar	http://automatiza-backend:3000/v1/admin/acionamentos/ativar-inativar	PUT
Acionamento - Excluir	http://automatiza-backend:3000/v1/admin/acionamentos/1	DELETE
Consultar lista funcionalidades tracker	http://automatiza-backend:3000/v1/admin/tracker/all	GET
N/A	http://automatiza-backend:3000/v1/admin/auth/login	POST

Figura 5.3: Comparação de processos elementares identificados de forma manual e automatizada

limitações na detecção automática, diferenças na interpretação dos processos ou identificação de requisições não-funcionais, conforme exemplo. As discrepâncias e as semelhanças encontradas foram sistematicamente registradas em uma base de dados específica, permitindo uma análise quantitativa e qualitativa detalhada. Essa análise buscou não apenas verificar a acuracidade do método automatizado, mas também compreender as razões por trás de eventuais diferenças, considerando aspectos como a granularidade da identificação, a definição de processos elementares, e a sensibilidade do algoritmo utilizado. As divergências mais relevantes foram objetos de uma análise aprofundada em uma fase subsequente, na qual avaliou-se o desempenho do modelo de medição, suas limitações e possibilidades de aprimoramento.

5.3 Análise de resultados

A análise de resultados é uma etapa fundamental no ciclo de validação e aprimoramento da identificação automatizada de processos elementares, desempenhando um papel importante em busca de precisão e confiabilidade da solução. Essa etapa caracteriza-se por uma análise dos desvios identificados na fase anterior de validação, com o objetivo de compreender as causas de cada divergência observada entre os resultados obtidos de forma manual e automatizada. Para tanto, os desvios identificados foram avaliados, buscando-se determinar a causa da possível não conformidade. No total, foram identificados 80

processos elementares de forma manual e 93 processos elementares de forma automatizada, uma diferença de 16%. Do ponto de vista de mercado, a diferença aceitável entre uma mesma medição realizada por dois profissionais diferentes é de 15% (KAMPSTRA; VERHOEF, 2010). Logo, em uma análise inicial, a diferença obtida está em um limiar considerado aceitável pelo mercado. Quando analisado no detalhe, observou-se que na medição manual, os profissionais que fizeram a identificação dos processos elementares realizaram um agrupamento maior de processos, ou seja, alguns processos elementares que foram identificados como distintos na abordagem automatizada, na abordagem manual foram considerados únicos. Com base nas divergências analisadas, verificou-se que jornadas dos usuários mais longas, como cadastros com abas, ou telas de cadastro com algumas funcionalidades de consulta associadas, na abordagem manual foram consideradas únicas, já na abordagem automatizada, dada a característica dos processos com muitas requisições acabaram sendo identificados de forma separada. Em alguns dos casos avaliados a abordagem automatiza em relação a manual poderia até gerar uma dupla interpretação o que, em tese, poderia considerar os dois cenários como válidos. Outros casos, entende-se que a abordagem fez um nível de decomposição funcional a mais do que preconizado pelo manual do SFP. Contudo, dada as características específicas dos processos onde ocorreram as divergências, entende-se que tal desvio pode ser considerado aceitável. Outro ponto importante refere-se a unicidade do processo elementar, um dos ponto chave na identificação de processos elementares vide regras do SFP. Via de regra, processos elementares que possuam o mesmo conjunto de dados, mesmo agrupamento lógico utilizado para acessar/gravar dados e mesma lógica de processamento são considerados o mesmo processo elementar, logo não podem ser contabilizados/identificados mais de um vez para um mesmo sistema. Dada a utilização do algoritmo LCS, que faz a identificação de subsequências semelhantes, essas subsequências identificadas como jornadas do usuário e, portanto, processos elementares, não são contabilizadas mais de uma vez, deste modo, garantindo a adesão as premissas para unicidade do processo elementar. Com a utilização do LCS, não foram identificados processos elementares em duplicidade. Por fim, cabe ressaltar, que a utilização da estratégia de uma lista de exceção para as requisições rotuladas como não-funcionais se mostrou eficiente. Cerca de 78% das requisições não-funcionais foram catalogadas como tal pelo próprio tracker em sua execução. O percentual não coberto, quando identificado em uma atividade de curadoria humana, é inserido manualmente na referida tabela e passa a ser considerado como tal.

5.4 Mecanismo de monitoramento de requisições HTTP

A compreensão do comportamento dos usuários em aplicações WEB é peça chave para a identificação de processos elementares. Com base nesse comportamento, entende-se as jornadas dos usuários realizadas e deriva-se disso o entendimento dos processos elementares existentes. Uma jornada do usuário é um fluxo que negocialmente faz sentido com a existência de um início (gatilho/ação/momento), meio (dados/processamento/lógica) e fim (conclusão/mudança de estado), o que encaixa perfeitamente ao conceito técnico de processo elementar. No contexto desta pesquisa, foi desenvolvida uma ferramenta denominada *Tracker* cujo objetivo é monitorar requisições HTTP de uma determinada aplicação. Todas as requisições realizadas, a partir do momento que a ferramenta é ativada, passam a ser capturadas e registradas em uma base de dados própria. O registro do acionamento das requisições é realizado por usuário, atribuindo a cada requisição acionada um identificador único (ID), o usuário que a acionou, o verbo do método HTTP a que se refere tal requisição e o referido corpo (dados). A medida que outros usuários acionem uma mesma requisição já acionada anteriormente por outro usuário, será atribuída a esses usuários que a acessam posteriomente o mesmo ID atribuído a o usuário que efetuou o acesso originário ou inicial. Com essa estratégia, é possível estabelecer a sequência de requisições realizadas por um coniunto de usuários em um determinado espaço de tempo. Com base nas sequências geradas pelo *Tracker*, utiliza-se o algoritmo LCS com o objetivo de identificar as subsequências comuns entre as sequências dos usuários e, consequentemente, os potenciais processos elementares identificados. Uma vez identificados os potenciais processos, é realizada pelo *Tracker* a análise das requisições para verificar se são funcionais ou não. Caso possuam característica de requisições não funcionais, ou seja, relacionadas a medição de desempenho, sincronização, helth checks, permissão de acesso/autenticação, entre outras), essas requisições são inseridas em uma lista de exceção e são desconsideradas das sequências de IDs de requisições atribuídas aos usuários. Portanto, além do papel de obtenção e registro das requisições da aplicação em análise, o *Tracker* é o mecanismo utilizado para diferenciação de requisições consideradas funcionais das não funcionais.

5.5 Resultados

O método proposto para a identificação automatizada de processos elementares foi aplicada em dois sistemas distintos, com o objetivo de avaliar sua eficácia e potencial de complementação ou substituição aos métodos tradicionais de análise manual. A primeira aplicação avaliada utiliza o framework javascript Vue.js na camanda de apresentação e backend em Node.js. Já a segunda aplicação utiliza o framework Angular para a camada

Sistema	Quantidade de PEs		Variação (%)
	# PEs manual	# PEs automatizado	
Sistema de gestão de OS	48	55	15%
Sistema de controle de ativos	32	38	19%

Tabela 5.1: Resultado da identificação de processos elementares pelos métodos automatizado e manual.

de apresentação e springboot no backend. Optou-se pelas aplicações indicadas pelo fato de utilizarem linguagens distintas e ambas serem padrões de mercado para aplicações WEB.

No primeiro sistema avaliado, inicialmente foram identificados manualmente 48 processos elementares, por meio de análise detalhada de documentação e entrevistas com os responsáveis pelo sistema. Em seguida, aplicando-se o método automatizado baseado na análise de requisições HTTP, foi possível identificar 55 processos elementares. Essa quantidade representa um aumento de aproximadamente 15% em relação ao número de processos identificados manualmente. Essa diferença pode ser atribuída à capacidade do método de analisar padrões de requisições de forma detalhada, capturando nuances e processos que não estavam explicitamente documentados ou facilmente identificáveis por análise manual ou conforme indicado na Seção 5.3 na contagem manual os processos elementares foram mais agrupados, assim processos elementares considerados distintos pela abordagem automatizada, foram considerados únicos na avaliação manual. Os processos elementares que tiveram tais divergências tinham características de serem mais longos e com várias etapas necessárias para executá-los, na sua maioria cadastros extensos.

No segundo sistema, o cenário foi semelhante, porém com uma quantidade menor de processos inicialmente identificados manualmente: 32 processos. A aplicação do método automatizado resultou na identificação de 38 processos elementares, o que corresponde a uma variação de aproximadamente 19%. Essa diferença reforça a hipótese de que o método automatizado é especialmente útil em ambientes onde a documentação é escassa ou desatualizada, pois consegue captar processos adicionais que podem não estar formalmente registrados. Além disso, a análise dos resultados demonstra que a abordagem baseada em requisições HTTP é viável e eficiente para a identificação de processos elementares, mesmo em contextos de sistemas menos documentados.

Outro aspecto importante a ser destacado é a utilização do algoritmo LCS, que desempenhou papel fundamental na identificação de jornadas de usuários e no mapeamento dos processos elementares. A aplicação do LCS permitiu comparar sequências de requisições e identificar padrões recorrentes, resultando em uma correspondência próxima aos processos identificados manualmente, conforme as premissas estabelecidas pelo IFPUG. Essa técnica contribuiu para uma maior precisão na delimitação dos processos, facilitando a distinção

entre diferentes jornadas de usuário e garantindo maior confiabilidade na medição.

Os resultados obtidos demonstram que a abordagem de identificação de processos elementares com base em requisições HTTP é eficaz e viável, especialmente em ambientes onde a documentação formal é limitada ou desatualizada. Além de explorar a independência tecnológica, conforme preconizado pelo IFPUG, esse método mostrou-se particularmente útil para aplicações web onde a dinâmica dos processos muitas vezes não é completamente refletida na documentação tradicional. Assim, a combinação do método automatizado com técnicas de análise de sequências, como o LCS, possibilita uma medição funcional mais rápida, precisa, repetível e menos dependente de documentação formal, contribuindo significativamente para projetos que demandam medições funcionais em ambientes complexos e em constante evolução.

Capítulo 6

Considerações finais

Este trabalho apresentou e validou um método para a identificação automatizada de processos elementares em aplicações WEB, fundamentado na análise de requisições HTTP. O objetivo primordial foi desenvolver uma abordagem que permitisse automatizar, ao menos em parte, o processo de medição funcional, aderindo aos princípios de independência de tecnologia preconizados pelo SFP. A solução proposta, composta pelas fases de captura, filtragem de requisições não-funcionais, identificação de similaridades via algoritmo LCS e classificação de processos elementares, demonstrou o potencial de transformar dados brutos de interação em informações estruturadas e úteis para a medição de software. A principal contribuição deste trabalho reside, portanto, na capacidade de automatizar a identificação de processos elementares, um estágio complexo e propenso a erros da medição funcional. Ao utilizar as requisições HTTP como fonte primária de informação, a metodologia desenvolvida oferece uma alternativa agnóstica à tecnologia de desenvolvimento da aplicação, superando barreiras inerentes a outras abordagens que dependem de acesso ao código-fonte ou documentação detalhada. A ferramenta *Tracker* e o processo de filtragem e agrupamento de requisições pelo emprego do algoritmo LCS demonstraram a viabilidade técnica dessa proposta, enquanto a fase de validação buscou assegurar a acurácia do modelo.

A complexidade da identificação de processos elementares, especialmente na diferenciação entre requisições funcionais e não-funcionais e na composição de requisições em processos completos, revelou-se um desafio significativo, evidenciando a necessidade de uma lista de exceções evolutiva e a dependência da consistência na ordenação de parâmetros. A importância da curadoria de dados também se mostrou crucial, pois a qualidade e abrangência dos dados capturados impactam diretamente a acurácia do modelo. No estágio atual desse trabalho, identificam-se algumas oportunidades de investigação adicionais à título de trabalhos futuros. Dentre elas, sugere-se investigar a aplicação de técnicas de *Machine Learning* para automatizar e aprimorar o filtro de requisições não-funcionais,

otimizando-o e tornando-o mais adaptável a diferentes contextos de modo a permitir um grau adicional de automação. É também pertinente explorar algoritmos de agrupamento mais sofisticados para a identificação de processos elementares, considerando não apenas a similaridade estrutural, mas também a sequência temporal, a frequência de uso e o contexto de negócio. A realização de estudos de caso em escala maior e com maior diversidade de aplicações web seria fundamental para avaliar a generalização do método. A pesquisa na inferência automatizada de funções de dados (ALI/AIE) a partir das requisições HTTP, talvez com o uso de técnicas de processamento de linguagem natural (PLN), representa outra fronteira a ser explorada. Adicionalmente, investigar a integração da solução *Tracker* e do método em pipelines de Integração Contínua/Entrega Contínua (CI/CD) poderia permitir a medição funcional contínua. Por fim, analisar o impacto da variabilidade do comportamento do usuário na identificação de processos elementares e realizar comparações com outras métricas de tamanho de software poderiam oferecer novas perspectivas sobre a aplicabilidade e complementaridade da abordagem proposta. Em suma, este trabalho representa um passo significativo na direção da automação da medição funcional, fornecendo uma base sólida para futuras investigações e desenvolvimentos na área de engenharia de software e métricas.

Referências Bibliográficas

ABITEBOUL, S.; HULL, R.; VIANU, V. **Foundations of Databases: The Logical Level**. [S.l.]: Addison-Wesley, 2011. 27

ABRAHãO, S.; POELS, G.; PASTOR, O. A functional size measurement method for object-oriented conceptual schemas: design and evaluation issues. **Software & Systems Modeling**, v. 5, n. 1, p. 48–71, abr. 2006. ISSN 1619-1366, 1619-1374. Disponível em: <http://link.springer.com/10.1007/s10270-005-0098-x>. 5

AL., R. et. **Hypertext Transfer Protocol – HTTP/1.1**. 1999. <https://tools.ietf.org/html/rfc2616>. Obsoleted by RFC 7230, RFC 7231, RFC 7232, RFC 7233, RFC 7234, and RFC 7235. 20, 29

ALBRECHT, A. Measuring application development productivity. In: **Proceedings of the IBM Appliction Development Symposium**. Monterey: [s.n.], 1979. p. 83–92. 1

ALMEIDA, W. H. C. "análise sobre métricas em contratos de fábricas de software no âmbito da administação pública". In: . Albatroz, 2019. v. 1. ISBN 9788571451629. Disponível em: <https://biblioteca.tse.jus.br:443/F/?func=direct&doc_number=000125278&local_base=TSE01>. 3

ARMALY, A.; KLACZYNSKI, J.; MCMILLAN, C. A Case Study of Automated Feature Location Techniques for Industrial Cost Estimation. In: **2016 IEEE International Conference on Software Maintenance and Evolution (ICSME)**. Raleigh, NC, USA: IEEE, 2016. p. 553–562. ISBN 9781509038060. Disponível em: <http://ieeexplore.ieee.org/document/7816508/>. 8, 12, 13

BARROS, M. d. O.; GONCALVES, V. P. A Function Point Formulation for the Software Release Planning Problem. In: **2019 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)**. Porto de Galinhas, Recife, Brazil: IEEE, 2019. p. 1–11. ISBN 9781728129686. Disponível em: <https://ieeexplore.ieee.org/document/8870188/>. 1

BERGROTH, L.; HAKONEN, H.; RAITA, T. A survey of longest common subsequence algorithms. In: **Proceedings Seventh International Symposium on String Processing and Information Retrieval. SPIRE 2000**. [S.l.: s.n.], 2000. p. 39–48. 26

BLUEMKE, I.; MALANOWSKA, A. Usage of UML Combined Fragments in Automatic Function Point Analysis:. In: **Proceedings of the 15th International Conference on Evaluation of Novel Approaches to Software Engineering**. Prague, Czech Republic: SCITEPRESS - Science and Technology Publications, 2020. p. 305–312. ISBN

9789897584213. Disponível em: <http://www.scitepress.org/DigitalLibrary/Link.aspx?doi=10.5220/0009348303050312>. 10, 12, 13

BUNGART, J. W. **Redes de computadores: Fundamentos e protocolos**. [S.l.]: Editora SESI-Serviço Social da Indústria, 2017. 21

COSMIC - The standard methodology for sizing software. <https://cosmic-sizing.org/>. Acessado em: 01 de Dezembro de 2023. 10

COSTA, J.; PEREIRA, M.; BELLOZE, K.; BEZERRA, E. Longest common subsequence aplicada à comparação de proteínas. In: SBC. **Anais da IV Escola Regional de Informática do Rio de Janeiro**. [S.l.], 2021. p. 103–110. 26

EDAGAWA, T.; AKAIKE, T.; HIGO, Y.; KUSUMOTO, S.; HANABUSA, S.; SHIBAMOTO, T. Function point measurement from Web application source code based on screen transitions and database accesses. **Journal of Systems and Software**, v. 84, n. 6, p. 976–984, jun. 2011. ISSN 01641212. Disponível em: <https://linkinghub.elsevier.com/retrieve/pii/S0164121211000215>. 5, 11

EFFECTI. **Governo Federal contratou 2,5 bilhões de reais em TI em 2020**. 2021. <https://cio.com.br/noticias/governo-federal-contratou-r-25-bilhoes-em-ti-em-2020/>. Acessado em: 06 de Junho de 2022. 4

ERSOY, E.; BAGRIYANIK, S.; SOZER, H. On the accuracy of effort estimations based on cosmic functional size measurement: A case study. In: **Proceedings of the 18th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement**. [S.l.: s.n.], 2024. p. 528–537. 11

FERNÁNDEZ-DIEGO, M.; MÉNDEZ, E. R.; GONZÁLEZ-LADRÓN-DE-GUEVARA, F.; ABRAHÃO, S.; INSFRAN, E. An update on effort estimation in agile software development: A systematic literature review. **IEEE Access**, IEEE, v. 8, p. 166768–166800, 2020. 1

FETCKE, T.; ABRAN, A.; Tho-Hau Nguyen. Mapping the OO-Jacobson approach into function point analysis. In: **Proceedings of TOOLS USA 97. International Conference on Technology of Object Oriented Systems and Languages**. Santa Barbara, CA, USA: IEEE Comput. Soc, 1998. p. 192–202. ISBN 9780818683831. Disponível em: <http://ieeexplore.ieee.org/document/654721/>. 5

FIELDING, R.; GETTYS, J.; MOGUL, J.; FRYSTYK, H.; MASINTER, L.; LEACH, P.; BERNERS-LEE, T. **RFC2616: Hypertext Transfer Protocol–HTTP/1.1**. [S.l.]: RFC Editor, 1999. 14

FRAGOSO-DÍAZ, O. G.; SANDOVAL-ACOSTA, J. A.; ÁLVAREZ-RODRÍGUEZ, F. J.; ROJAS-PÉREZ, J. C.; SANTAOLAYA-SALGADO, R. Systematic review of quality in class diagrams for software engineering competencies. **IEEE Revista Iberoamericana de Tecnologias del Aprendizaje**, IEEE, v. 17, n. 4, p. 351–357, 2022. 11

FREDMAN, M. L. On computing the length of longest increasing subsequences. **Discrete Mathematics**, v. 11, n. 1, p. 29–35, 1975. ISSN 0012-365X. Disponível em: <https://www.sciencedirect.com/science/article/pii/0012365X7590103X>. 14

GASTOS mundiais com TI crescerão 6,8<https://itforum.com.br/noticias/gastos-com-ti-6-2024-gartner/>. Acessado em: 25 de Março de 2024. 4

GIACHETTI, G.; MARIN, B.; CONDORI-FERNANDEZ, N.; MOLINA, J. C. Updating OO-Method Function Points. In: **6th International Conference on the Quality of Information and Communications Technology (QUATIC 2007)**. Lisbon: IEEE, 2007. p. 55–64. ISBN 9780769529486. Disponível em: <https://ieeexplore.ieee.org/document/4335234/>. 5

GUEVARA, F. González Ladrón de; FERNáNDEZ-DIEGO, M.; LOKAN, C. The usage of ISBSG data fields in software effort estimation: A systematic mapping study. **Journal of Systems and Software**, v. 113, p. 188–215, mar. 2016. ISSN 01641212. Disponível em: <https://linkinghub.elsevier.com/retrieve/pii/S0164121215002642>. 2

GUPTA, D.; KAUSHAL, S. J.; SADIQ, M. Software estimation tool based on three -layer model for software engineering metrics. In: **2008 4th IEEE International Conference on Management of Innovation and Technology**. [S.l.: s.n.], 2008. p. 623–628. 5, 11

HUIJGENS, H.; BRUNTINK, M.; DEURSEN, A. van; STORM, T. van der; VOGELEZANG, F. An exploratory study on functional size measurement based on code. In: **2016 IEEE/ACM International Conference on Software and System Processes (ICSSP)**. [S.l.: s.n.], 2016. p. 56–65. 1

IETF RFC 7231. **Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content**. 2014. <https://tools.ietf.org/html/rfc7231>. Obtenido de IETF. 22, 33

INTERNATIONAL Fuction Point Users Group. <https://ifpug.org/>. Acessado em: 25 de Fevereiro de 2024. vii, 1, 15, 16, 17, 18

INTERNATIONAL FUNCTION POINT GROUP - IFPUG. **Function Point Counting Practices Manual**. 4.3.1. ed. [S.l.], 2010. 3, 4

INTERNATIONAL FUNCTION POINT GROUP - IFPUG. **Simple Function Point (SFP) Counting Practices Manual**. 2.1. ed. [S.l.], 2021. 3, 6, 14, 15, 16, 18, 38, 39, 40

ISO. **Information technology — Software measurement — Functional size measurement — Part 1: Definition of concepts**. 2007. <https://www.iso.org/standard/38931.html>. Acessado em: 26 de Abril de 2023. 1, 3, 14, 15

ISO/IEC. standard, **Software and systems engineering – Software measurement – IFPUG functional size measurement method 2009**. 2009. ISO/IEC 20926:2009. 15

JORGE, C. A. C. Comparação paralela de sequências biológicas em plataformas de hardware uniformes e híbridas. 2023. 25

JR., M. F.; FANTINATO, M.; SUN, V.; THOM, L.; GARAJ, V. Improving Reproducibility whilst Maintaining Accuracy in Function Point Analysis:. In: **Proceedings of the 21st International Conference on Enterprise Information Systems**. Heraklion, Crete, Greece: SCITEPRESS - Science and Technology Publications, 2019. p. 61–72. ISBN 9789897583728. Disponível em: <http://www.scitepress.org/DigitalLibrary/Link.aspx?doi=10.5220/0007671700610072>. 9, 12, 13

KAMPSTRA, P.; VERHOEF, C. Reliability of function point counts. **Department of Computer Science, VU University Amsterdam, Amsterdam, The Netherlands**, 2010. 49

KOVAGS, D.; FALCHI, F. L.; RIVAS, A. R. Analysis of the Utilization of Scrum Framework Effort Estimation Metrics in Federal Public Administration. In: **Proceedings of the XVIII Brazilian Symposium on Software Quality**. Fortaleza Brazil: ACM, 2019. p. 30–38. ISBN 9781450372824. Disponível em: <https://dl.acm.org/doi/10.1145/3364641.3364645>. 1

KUSUMOTO, S.; IMAGAWA, M.; INOUE, K.; MORIMOTO, S.; MATSUSITA, K.; TSUDA, M. Function point measurement from Java programs. In: **Proceedings of the 24th international conference on Software engineering - ICSE '02**. Orlando, Florida: ACM Press, 2002. p. 576. ISBN 9781581134728. Disponível em: <http://portal.acm.org/citation.cfm?doid=581339.581412>. 5, 11

LAMMA, E. A System for Measuring Function Points from an ER-DFD Specification. **The Computer Journal**, v. 47, n. 3, p. 358–372, mar. 2004. ISSN 0010-4620, 1460-2067. Disponível em: <https://academic.oup.com/comjnl/article-lookup/doi/10.1093/comjnl/47.3.358>. 5

LAVAZZA, L. Automated Function Points: Critical Evaluation and Discussion. In: **2015 IEEE/ACM 6th International Workshop on Emerging Trends in Software Metrics**. Florence, Italy: IEEE, 2015. p. 35–43. ISBN 9781467371032. Disponível em: <http://ieeexplore.ieee.org/document/7181589/>. 11

LAVAZZA, L.; LIU, G. A study of the correlation between functional size measures and object-oriented measures from uml requirements models. In: . [S.l.: s.n.], 2018. 10, 12, 13

LAVAZZA, L.; LOCORO, A.; MELI, R. Software development and maintenance effort estimation using function points and simpler functional measures. **Software**, v. 3, n. 4, p. 442–472, 2024. ISSN 2674-113X. Disponível em: <https://www.mdpi.com/2674-113X/3/4/22>. 2

LAVAZZA, L.; LOCORO, A.; MELI, R. Using machine learning and simplified functional measures to estimate software development effort. **IEEE Access**, IEEE, 2024. 11

LAVAZZA, L.; MORASCA, S.; ROBIOLO, G. Towards a simplified definition of function points. **Information and Software Technology**, Elsevier, v. 55, n. 10, p. 1796–1809, 2013. 3

LEMON, K. N.; VERHOEF, P. C. Understanding customer experience throughout the customer journey. **Journal of Marketing**, v. 80, n. 6, p. 69–96, 2016. 40

LENT, J. S.; QU, Y. On XML Based Automated Function Point Analysis: An Effective Method to Assess Developer Productivity. **Lecture Notes on Software Engineering**, v. 3, n. 4, p. 245–250, 2015. ISSN 23013559. Disponível em: <http://www.lnse.org/show-39-240-1.html>. 5

MARCÉN, A. C.; IGLESIAS, A.; LAPEÑA, R.; PÉREZ, F.; CETINA, C. A systematic literature review of model-driven engineering using machine learning. **IEEE Transactions on Software Engineering**, IEEE, 2024. 3

MASINTER, L.; BERNERS-LEE, T.; FIELDING, R. T. Uniform resource identifier (uri): Generic syntax. **Network Working Group**, Fremont, CA, 2005. 21

MERCADO brasileiro de TI tem alta projetada de 6,2<https://einvestidor.estadao.com.br/ultimas/mercado-ti-crescimento-2023-projecao/>. Acessado em: 28 de Novembro de 2023. 4

METSCH, T.; EDMONDS, A. et al. Open cloud computing interface-restful http rendering. In: **Open Grid Forum-OCCI Working group technical report**. [S.l.: s.n.], 2011. 21, 22, 23, 24, 25

Ministério da Economia; Secretaria Especial de Desburocratização, Gestão e Governo Digital; Secretaria de Governo Digital. **Portaria SLTI/MP no. 31**. 2010. Disponível em: <http://www.comprasnet.gov.br/legislacao/legislacaoDetalhe.asp?ctdCod=704>. 2

Ministério da Economia; Secretaria Especial de Desburocratização, Gestão e Governo Digital; Secretaria de Governo Digital. **Instrução Normativa no. 4**. 2014. Disponível em: <https://www.gov.br/compras/pt-br/acesso-a-informacao/legislacao/instrucoes-normativas-revogadas/instrucao-normativa-no-4-de-12-de-novembro-de-2010-revogada-pela-in-no-4-de-2014>. 1

Ministério da Economia; Secretaria Especial de Desburocratização, Gestão e Governo Digital; Secretaria de Governo Digital. **Instrução Normativa no. 1**. 2019. Disponível em: <https://www.in.gov.br/materia/-/asset_publisher/Kujrw0TZC2Mb/content/id/70267659/do1-2019-04-05-instrucao-normativa-n-1-de-4-de-abril-de-2019-70267535>. 1, 2

NAVARRO, G. A guided tour to approximate string matching. **ACM Computing Surveys (CSUR)**, ACM, v. 33, n. 1, p. 31–88, 2001. 26

NEEDLEMAN, S. B.; WUNSCH, C. D. A general method applicable to the search for similarities in the amino acid sequence of two proteins. **Journal of molecular biology**, Elsevier, v. 48, n. 3, p. 443–453, 1970. 26

NEYVELI, V. R. N.; SIVAKUMAR, S. S.; ARUNAGIRI, D.; ARUMUGAM, C.; VEERAMANI, A. M. An Approach to Estimate the Size of Web Application Using IFML User Interface Model. In: **2019 Amity International Conference on Artificial**

Intelligence (AICAI). Dubai, United Arab Emirates: IEEE, 2019. p. 292–295. ISBN 9781538693469. Disponível em: <https://ieeexplore.ieee.org/document/8701268/>. 10, 12, 13

NHUNG, H. L. T. K.; SILHAVY, R.; SILHAVY, P. Estimating function points of development project in iot systems case study. In: SPRINGER. **Computer Science On-line Conference**. [S.l.], 2024. p. 282–294. 11

Object Management Group. **Automated Function Points (AFP), version 1.0**. 2014. Disponível em: <https://www.omg.org/spec/AFP/1.0/About-AFP/>. 6

OZKAN, B.; DEMIRORS, O. On the Seven Misconceptions about Functional Size Measurement. In: **2016 Joint Conference of the International Workshop on Software Measurement and the International Conference on Software Process and Product Measurement (IWSM-MENSURA)**. Berlin, Germany: IEEE, 2016. p. 45–52. ISBN 9781509041473. Disponível em: <http://ieeexplore.ieee.org/document/7809590/>. 1

PAGE, L.; BRIN, S.; MOTWANI, R.; WINOGRAD, T. **The PageRank Citation Ranking: Bringing Order to the Web.** [S.l.], 1999. Previous number = SIDL-WP-1999-0120. Disponível em: <http://ilpubs.stanford.edu:8090/422/>. 8

POSPIESZNY, P.; CZARNACKA-CHROBOT, B.; KOBYLINSKI, A. An effective approach for software project effort and duration estimation with machine learning algorithms. **Journal of Systems and Software**, Elsevier, v. 137, p. 184–196, 2018. 3

POW-SANG, J. A.; VILLANUEVA, D.; FLORES, L.; RUSU, C. A Conversion Model and a Tool to Identify Function Point Logic Files Using UML Analysis Class Diagrams. In: **2013 Joint Conference of the 23rd International Workshop on Software Measurement and the 8th International Conference on Software Process and Product Measurement**. [S.l.: s.n.], 2013. p. 126–134. 5

PRESSMAN, R. S. **Engenharia de Software**. 8th. ed. [S.l.]: McGraw-Hill Education, 2014. 17, 19

QUESADA-LÓPEZ, C.; JENKINS, M. Procedimientos de medición del tamaño funcional: Un mapeo sistemático de literatura. In: **CIbSE**. [S.l.: s.n.], 2017. p. 141–154. 5

QUESADA-LOPEZ, C.; JENKINS, M.; SALAS, L. C.; GOMEZ, J. C. Una herramienta para la estimación temprana del tamaño funcional del software. In: **2020 15th Iberian Conference on Information Systems and Technologies (CISTI)**. Sevilla, Spain: IEEE, 2020. p. 1–6. ISBN 9789895465903. Disponível em: <https://ieeexplore.ieee.org/document/9141112/>. 9, 12

QUESADA-LóPEZ, C. **Automatización de la medición del tamaño funcional del software para modelos funcionales obtenidos a partir del análisis dinámico del código fuente**. Tese (Ph.D. Thesis) — Universidad de Costa Rica, 2018. 2, 6

QUESADA-LóPEZ, C.; JENKINS, M.; SALAS, L. C.; GóMEZ, J. C. Towards an automated functional size measurement procedure: an industrial case study. In: **Proceedings of the 27th International Workshop on Software Measurement and 12th International Conference on Software Process and Product Measurement**. Gothenburg Sweden: ACM, 2017. p. 138–144. ISBN 9781450348539. Disponível em: <https://dl.acm.org/doi/10.1145/3143434.3143460>. 8, 12, 13

QUESADA-LóPEZ, C.; MARTíNEZ, A.; JENKINS, M.; SALAS, L. C.; GóMEZ, J. C. Automated Functional Size Measurement: A Multiple Case Study in the Industry. In: FRANCH, X.; MäNNISTö, T.; MARTíNEZ-FERNáNDEZ, S. (Ed.). **Product-Focused Software Process Improvement**. Cham: Springer International Publishing, 2019. v. 11915, p. 263–279. ISBN 978-3-030-35332-2 978-3-030-35333-9. Series Title: Lecture Notes in Computer Science. Disponível em: <http://link.springer.com/10.1007/978-3-030-35333-9_19>. 9, 12

RFC822: Standard for ARPA Internet Text Messages. <https://www.w3.org/Protocols/rfc822/>. Acessado em: 01 de Dezembro de 2023. 21, 24

RUSLI, N. I. A.; ABDULLAH, N. A. S. Functional Size Measurement Tool-based Approach for Mobile Game. **International Journal on Advanced Science, Engineering and Information Technology**, v. 10, n. 3, p. 993, jun. 2020. ISSN 2460-6952, 2088-5334. Disponível em: <http://ijaseit.insightsociety.org/index.php?option=com_content&view=article&id=9&Itemid=1&article_id=10185>. 10, 12, 13

SALEM, S.; SOUBRA, H. Functional size measurement automation for iot edge devices. In: **IWSM-Mensura**. [S.l.: s.n.], 2023. 5

SANTOSO, I.; WARNARS, H. L. H. S.; GAOL, F. L.; ABDURACHMAN, E.; SOEWITO, B. Measurement of web-based merchant application portal (map) using function point analysis and constructive cost model ii. In: **Proceedings of the 4th Asia Pacific Conference on Research in Industrial and Systems Engineering**. [S.l.: s.n.], 2021. p. 639–644. 11

SHI, L.; LI, M.; XING, M.; WANG, Y.; WANG, Q.; PENG, X.; LIAO, W.; PI, G.; WANG, H. Learning to extract transaction function from requirements: an industrial case on financial software. In: **Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering**. Virtual Event USA: ACM, 2020. p. 1444–1454. ISBN 9781450370431. Disponível em: <https://dl.acm.org/doi/10.1145/3368089.3417053>. 10, 12, 13

SILVA, A.; PINHEIRO, P.; ALBUQUERQUE, A. A Brief Analysis of Reported Problems in the Use of Function Points. In: SILHAVY, R.; SENKERIK, R.; OPLATKOVA, Z. K.; SILHAVY, P.; PROKOPOVA, Z. (Ed.). **Software Engineering Perspectives and Application in Intelligent Systems**. Cham: Springer International Publishing, 2016. v. 465, p. 117–126. ISBN 9783319336206 9783319336220. Disponível em: <http://link.springer.com/10.1007/978-3-319-33622-0_11>. 2

SOMMERVILLE, I. **Software Engineering**. 10. ed. Boston (ou outra cidade conforme edição, se conhecida): Pearson, 2015. 15

STAMBOLLIAN, A.; ABRAN, A. Foundations of measurement tools with cosmic: Survey of automation tools supporting cosmic ffp-iso 19761. **COSMIC Function Points: Theory and Advanced Practices**, p. 299–319, 04 2016. 2

SUCHÁNEK, M. **Towards a Normalized Systems Gateway Ontology for Conceptual Models**. [S.l.]: Czech Technical University, 2023. 11

SUGUIMOTO, G. M.; PAULA, F. S. de. Extração de assinaturas de protocolos de aplicação através de análise de subsequências comuns. **ANAIS DO ENIC**, n. 1, 2009. 26

SáNCHEZ, D. M.; LóPEZ, C. U. Q.; CORONAS, M. J. Towards the automation of a defect detection protocol for functional size measurements. 2018. Disponível em: <https://www.kerwa.ucr.ac.cr/handle/10669/76926?show=full>. 9, 12, 13

TANENBAUM, A. S.; WETHERALL, D. **Rede de computadores**. São Paulo :: Pearson Prentice Hall,, 2011. Tradução de Computer networks, 5. ed. americana. 19

UNIãO, T. de Contas da. **Avaliação do Modelo de Contratação de Desenvolvimento de Software**. 2007. <https://pesquisa.apps.tcu.gov.br>. Acessado em: 22 de Março de 2022. 3

UNIãO, T. de Contas da. **Acordão TCU 2362/2015**. 2015. <https://pesquisa.apps.tcu.gov.br/redireciona/acordao-completo/ACORDAO-COMPLETO-1539760>. 15

UNIãO, T. de Contas da. **Avaliação do Modelo de Contratação de Desenvolvimento de Software**. 2015. <https://portal.tcu.gov.br/biblioteca-digital/avaliacao-do-modelo-de-contratacao-de-desenvolvimento-de-software.htm>. Accessado em: 30 de dezembro de 2021. 3

WALLACE, L. G.; SHEETZ, S. D. The adoption of software measures: A technology acceptance model (tam) perspective. **Information & Management**, Elsevier, v. 51, n. 2, p. 249–259, 2014. 3