UNIVERSIDADE FUMEC
FACULDADE DE CIÊNCIAS EMPRESARIAIS – FACE
Mestrado Profissional em Sistemas de Informação e Gestão Do Conhecimento

PROPOSTA DE UM ARCABOUÇO CONCEITUAL DE INTELIGÊNCIA ANALÍTICA PARA A ENGENHARIA DE SOFTWARE

Bruno Rafael de Oliveira Rodrigues

Belo Horizonte
2016

Bruno Rafael de Oliveira Rodrigues

PROPOSTA DE UM ARCABOUÇO CONCEITUAL DE INTELIGÊNCIA ANALÍTICA PARA A ENGENHARIA DE SOFTWARE

Dissertação apresentada ao Curso de Mestrado em Sistemas de Informação e Gestão do Conhecimento da Universidade Fumec como parte dos requisitos para a obtenção do título de Mestre em Sistemas de Informação e Gestão do Conhecimento.

Área de Concentração: Gestão de Sistemas de Informação e do Conhecimento

Linha de Pesquisa: Tecnologia e Sistemas de Informação

Orientador: Prof. Dr. Fernando Silva Parreiras

Belo Horizonte
2016

Rodrigues, Bruno Rafael de Oliveira.
R696p Proposta de um arcabouço conceitual de inteligência analítica para a engenharia de software. / Bruno Rafael de Oliveira Rodrigues – Belo Horizonte, 2016.

67 f. : il. (algumas col.) ; 30 cm.

Orientador: Fernando Silva Parreiras.
Dissertação (mestrado) – Universidade FUMEC. Faculdade de Ciências Empresariais.

Inclui bibliografia.

1. Inteligência competitiva (Administração) – Estudo de casos. 2. Engenharia de software – Estudo de casos. I. Parreiras, Fernando. II. Universidade FUMEC. Faculdade de Ciências Empresariais. III. Título.

CDU: 65.011:681.3.6

Elaborada por Olívia Soares de Carvalho. CRB/6: 2070

UNIVERSIDADE
FUMEC
DE MINAS GERAIS PARA O MUNDO

Dissertação intitulada "**Proposta de um Arcabouço Conceitual para a Inteligência Analítica na Engenharia de Software**" de autoria de Bruno Rafael de Oliveira Rodrigues, aprovada pela banca examinadora constituída pelos seguintes professores:

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

Prof. Dr. João Victor Boechat Gomide – Universidade FUMEC
(Examinador Interno)

Prof. Dr. Mark Alan Junho Song – PUC Minas
(Examinador Externo)

André Alves Ferreira, Esp. – PRODEMGE
(Consultor *Ad Hoc*)

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

Belo Horizonte, 12 de julho de 2016.

Rua Cobre, 200 - Cruzeiro
30310-190 - Belo Horizonte - MG
Tel. (31) 3228-3060
www.fumec.br

A todos que se esforçam para realizar seus sonhos e fazer um mundo melhor.

AGRADECIMENTOS

A todos os meus familiares por acreditarem no meu sucesso.

Aos meus colegas de trabalho por me apoiarem em meus estudos.

Aos meus professores por sempre me incentivarem e inspirarem.

Aos colegas e amigos por me motivarem e ajudarem em mais uma vitória.

Aos participantes do LAIS (Laboratório de Sistemas de Informação Avançados) pela ajuda inestimável.

Ao ProPic Fumec, pelos recursos fornecidos, que foram fundamentais no início da pesquisa.

E aos participantes do grupo focal que foram de suma importância para este trabalho.

“Torture numbers, and they'll confess to anything.”

Gregg Easterbrook

RESUMO

As pesquisas e as práticas em inteligência analítica na Engenharia de Software têm crescido nas últimas décadas. As informações contidas em um repositório de software podem auxiliar engenheiros de software em suas atividades durante todas as fases do desenvolvimento de software. O uso da inteligência analítica está ajudando os profissionais da Engenharia de Software a obterem informações relevantes do repositório de software, direcionando-os para melhores tomadas de decisões. Por se tratar de um bem intangível, pode ser difícil compreender as informações geradas pelo software. Este trabalho realizou um mapeamento sistemático da literatura sobre inteligência analítica na Engenharia de Software, o que propiciou a elaboração de um arcabouço conceitual para utilização da inteligência analítica capaz de auxiliar nas atividades da Engenharia de Software. Com a finalidade de validar este arcabouço conceitual, foi construído um protótipo de uma aplicação que analisou dados de um software livre. Tal protótipo foi validado e comentado por um grupo focal formado por desenvolvedores e gestores de projetos de software de uma grande empresa da área de Tecnologia da Informação. Concluiu-se que a inteligência analítica é fortemente utilizada durante a fase de manutenção e vem crescendo sua utilização na área de Gestão e na Prática Profissional. Constatou-se que os *commits* podem ser bons indicadores da evolução de software e que a ferramenta desenvolvida neste trabalho permite compreender o que está sendo alterado no sistema e por que a alteração ocorreu.

Palavras-chave: Inteligência Analítica, Mineração de Repositório de Software, Engenharia de Software.

Abstract

The research and practice in mining software repositories have grown in recent decades. The existing information about a software repository can assist software engineers in their activities during all phases of software development. The use of software analytics is benefiting software engineering practitioners to obtain relevant information in the software repository, directing them to better decision making. Because it is an intangible asset, the information generated by the software can be complicated to be understood. This paper conducted a systematic mapping of literature about software analytics usage in software engineering that allowed the development of a conceptual framework to assist in the activities of software engineering. In order to validate this conceptual framework, we built a prototype of an application that analyzed data of a open source software. This prototype was validated and reviewed by a focus group of software developers and project managers from a large information technology organization. It was noticed that the software analytics is strongly used during the maintenance phase and its usage is growing in management and professional practice. Another results shows that commits can be good indicators of software evolution and that the tool developed in this work allows the understanding of what is being changed in the system and the cause of change.

Key Words: Software Analytics, Mining Software Repository, Software Engineering

Lista de Figuras

Lista de Tabelas

Lista de Quadros

Lista de Abreviaturas e Siglas

API	Application Programming Interface
BSII	Bug Sequence Intensity Index
CF	Commodification factor
CK	*Chidamber and Kemerer*
CVS	Concurrent Version System
FDCI	File De- pendency Churn Index
ICSE	International Conference on Software Engineering
IBK	Instance-Bases learning with parameter k
ITIL	Information Technology Infrastructure Library
LDA	Latent Dirichlet Allocation
MSR	Mining Software Repositories
MPS.BR	Melhoria de Processos do Software Brasileiro
NoC	Number of commits
PMP	Project Management Professional
RPI	Risky Profile Index
SBES	Simpósio Brasileiro de Engenharia de Software
SCM	Software Configuration Management
SVM	Support Vector Machines
SVN	Subversion
SWEBOK	Software Engineering Body of Knowledge
Weka	Waikato Environment for Knowledge Analysis

Sumário

1 INTRODUÇÃO

Durante o desenvolvimento de um software, são produzidos diversos artefatos, que contêm dados importantes a respeito de seu projeto. Contudo, nem sempre esses dados são explorados com a finalidade de auxiliar desenvolvedores e gestores de software em suas atividades. Muitas decisões tomadas no desenvolvimento de software baseiam-se na intuição [3] e poucas ferramentas são utilizadas para direcionar a condução de projetos de software. Diante de uma grande quantidade de dados disponíveis, engenheiros de software têm dificuldade para usar e interpretar dados sobre o projeto de software [1]. O propósito da inteligência analítica é transformar grandes quantidades de dados em informações reais e viáveis [1], por exemplo, fazendo uso de técnicas de aprendizado de máquina ou estatística para analisar dados. A inteligência analítica na Engenharia de Software permite que dados coletados em repositórios possam ser explorados e analisados [5], ajudando os engenheiros de software a obterem e compartilharem uma visão dos seus dados que são úteis para embasar a tomada de decisões [4]. Essa técnica é utilizada, por exemplo, na redução de defeitos e no gerenciamento de riscos no desenvolvimento de software [1], bem como para entender sistemas, propagação de mudança de código e entender dinâmicas das equipes e melhorar a experiência do usuário, entre outros benefícios [2].

Com a intenção de auxiliar os engenheiros de software em suas atividades, este trabalho visa responder ao problema de pesquisa: Quais são os elementos tecnológicos de um arcabouço conceitual da inteligência analítica aplicados à Engenharia de Software? Este trabalho propõe-se a elaborar um arcabouço conceitual que utilize a inteligência analítica na Engenharia de Software que facilite a exploração das informações disponíveis em repositórios de software, com a finalidade de obter informações a respeito do projeto de software que sejam úteis para embasar decisões.

Para compreender o que está sendo produzido sobre inteligência analítica na Engenharia de Software, este trabalho contempla um mapeamento sistemático da literatura. Com base nos dados extraídos dos artigos analisados no mapeamento sistemático, propõe-se um arcabouço conceitual que descreve as tecnologias para utilização da inteligência analítica na Engenharia de Software. Tal arcabouço visa suprir as necessidades de informações sobre o projeto pertinentes à Engenharia de Software. Para validar este arcabouço conceitual, desenvolveu-se um protótipo analisando os dados de um sistema de software livre, o qual foi avaliado por um grupo focal constituído por desenvolvedores e gerentes de software de uma grande empresa da área de Tecnologia da Informação do Estado de Minas Gerais.

O Programa de Pós-graduação em Sistemas de Informação e Gestão do Conhecimento da FUMEC possui caráter profissional; ou seja, tem foco na realização de pesquisas aplicadas e práticas em áreas gerenciais e tecnológicas. Possui formação interdisciplinar nas linhas de pesquisa em Tecnologia e Sistemas de Informação e em Gestão da Informação e do Conhecimento. Com base nesse programa interdisciplinar, este trabalho se alinha ao escopo da linha de pesquisa de Tecnologia em Sistemas de Informação, a qual compreende os estudos do processo de desenvolvimento de tecnologias.

A interdisciplinaridade deste trabalho pode ser constatada pela presença da criação de um arcabouço conceitual em Engenharia de Software que se adequa à área do conhecimento da Ciência da Computação. Por possuir características de análise de dados, adéqua-se, também à área de Estatística.

1.1 OBJETIVOS

1.1.1 Objetivo Geral

Propor um arcabouço conceitual de inteligência analítica na Engenharia de Software, com a finalidade de fornecer múltiplas visões do projeto de software, a partir das informações geradas pelos artefatos produzidos durante o seu desenvolvimento, para auxiliar os engenheiros de software a compreenderem melhor o software que está sendo desenvolvido ou mantido.

1.1.2 Objetivos Específicos

- OBJ1 - Identificar os elementos de um arcabouço conceitual produzidos na literatura sobre inteligência analítica na Engenharia de Software.
- OBJ2 - Identificar as tecnologias propostas na literatura que utilizem a inteligência analítica na prática da Engenharia de Software.
- OBJ3 - Elaborar um arcabouço conceitual da inteligência analítica para a Engenharia de Software.
- OBJ4 - Validar o arcabouço conceitual desenvolvido neste trabalho.

Integram este trabalho dois artigos submetidos ao Simpósio Brasileiro de Engenharia de Software 2016 (SBES). O Capítulo 1 contém o primeiro artigo, que apresenta o mapeamento sistemático da literatura sobre inteligência analítica na Engenharia de Software. Neste artigo foi possível identificar como a inteligência analítica está sendo utilizada na Engenharia de Software, descobrir as lacunas existentes e as tendências das pesquisas nessa área. O segundo artigo, que se encontra no Capítulo 2, apresenta um arcabouço conceitual que permite que seja utilizada a inteligência analítica na Engenharia de Software. Esse arcabouço é composto pelos elementos encontrados no mapeamento sistemático e visa suprir as diversas necessidades dos engenheiros de software. Com a finalidade de validar esse arcabouço, este trabalho implementa o arcabouço apresentado e realiza um grupo focal com desenvolvedores e gestores de projeto de software, no qual foi possível encontrar os pontos fortes e os pontos fracos do arcabouço.

1.2 REFERÊNCIAS

[1] Raymond P.L. Buse and Thomas Zimmermann. 2010. Analytics for Software Development. *Proceedings of the FSE/SDP Workshop on Future of Software Engineering Research*, ACM, 77–80. DOI= http://doi.org/10.1145/1882362.1882379

[2] A.E. Hassan. 2008. The road ahead for Mining Software Repositories. *Frontiers of Software Maintenance, 2008. FoSM 2008.*, 48–57. DOI= http://doi.org/10.1109/FOSM.2008.4659248

[3] Ahmed E. Hassan and Tao Xie. 2010. Software Intelligence: The Future of Mining Software Engineering Data. *Proceedings of the FSE/SDP Workshop on Future of Software Engineering Research*, ACM, 161–166. DOI= http://doi.org/10.1145/1882362.1882397

[4] Tim Menzies and Thomas Zimmermann. 2013. Software Analytics: So What? *IEEE Softw.* 30, 4: 31–37. DOI= http://doi.org/10.1109/MS.2013.86

[5] Dongmei Zhang, Yingnong Dang, Jian-Guang Lou, Shi Han, Haidong Zhang, and Tao Xie. 2011. Software Analytics As a Learning Case in Practice: Approaches and Experiences. *Proceedings of the International Workshop on Machine Learning Technologies in Software Engineering*, ACM, 55–58. DOI= http://doi.org/10.1145/2070821.2070829

2 CAPÍTULO 1

Uso de Inteligência Analítica na Engenharia de Software – Um Mapeamento Sistemático da Literatura

Resumo

Informações existentes em repositórios de software podem auxiliar engenheiros de software em suas atividades durante todas as fases do desenvolvimento. O uso da inteligência analítica em repositórios de software beneficia engenheiros de software a obterem informações relevantes a respeito do projeto que está sob sua responsabilidade. Com a finalidade de fomentar os estudos do uso da inteligência analítica na Engenharia de Software, este artigo realizou um mapeamento sistemático da literatura, para descobrir a aplicabilidade da inteligência analítica na Engenharia de Software. Foram analisados 235 artigos extraídos de cinco bases científicas da computação e de duas conferências da Engenharia de Software. Com base nesses artigos, foram categorizados os tipos de análise e as necessidades dos engenheiros de software que estão utilizando a inteligência analítica. Descobriu-se que métodos são frequentemente propostos com a finalidade de quantificar as mudanças dos artefatos de software na prática de desenvolvimento, seja relacionado ao código-fonte do projeto ou como na estimativa de esforço e risco do projeto. A inteligência analítica está fortemente relacionada à manutenção de software, mas sua utilização vem crescendo na gestão de projetos e no gerenciamento de equipes.

Categoria:

Sistemas de informação → Aplicações de sistemas de informação → Sistemas de apoio à decisão → Análise de dados

Palavras-chave

Inteligência Analítica; Mineração de Repositórios de Software; Engenharia de Software.

2.1 INTRODUÇÃO

A inteligência analítica é descrita como uma técnica que faz uso de análises, dados e raciocínio sistemático para tomar decisões [37]. Demonstrou ser uma técnica valiosa para a análise de processos de negócios [176]. Na Engenharia de Software seu uso tem crescido de forma substancial na identificação de padrões e no apoio à tomada de decisão em cada fase do desenvolvimento de software [24] [98].

Estudos utilizando dados de repositórios de desenvolvimento software têm coberto várias áreas da Engenharia de Software, como arquitetura, processo de desenvolvimento e reuso, entre outras [94]. A inteligência analítica está permitindo aos gerentes e engenheiros de software melhor compreensão a respeito dos sistemas sob sua responsabilidade [156].

Neste contexto, emerge o problema de pesquisa: Quais são as aplicações da inteligência analítica na Engenharia de Software? Este artigo visa identificar como a inteligência analítica está sendo utilizada para embasar decisões nas áreas da Engenharia de Software.

Para responder ao problema de pesquisa, procedeu-se a um mapeamento sistemático da literatura sobre inteligência analítica na Engenharia de Software. Para isso, foram pesquisados artigos em cinco bases científicas da computação - *ACM Digital Library, IEEE Xplore, Springer, Elsevier e Wiley* - e realizada a busca manual em duas conferências da área: *International Conference on Software Engineering (ICSE*) e *International Conference on Mining Software Repositories* (*MSR*). Os artigos resultantes deste mapeamento foram classificados com base em aspectos encontrados na literatura, apresentando uma dimensão da produção científica nessa área.

O artigo se encontra estruturado em sete seções, incluindo esta introdução. A Seção **2.2** contém a fundamentação teórica deste artigo. A Seção **2.3** apresenta os trabalhos relacionados. A Seção **2.4** descreve a metodologia utilizada para este mapeamento sistemático e os descritores de busca. A Seção **2.5** apresenta os resultados do mapeamento sistemático e responde às perguntas deste trabalho. A Seção **2.6** trata das ameaças à validade desta pesquisa. A Seção **2.7** discute as conclusões e oferece sugestões para trabalhos futuros.

2.2 FUNDAMENTAÇÃO TEÓRICA

Repositórios de software contêm dados do sistema que, muitas vezes, podem ser difíceis de serem interpretados, porém podem ser utilizados pelos engenheiros de software para embasar decisões. A inteligência analítica na Engenharia de Software analisa e cruza dados disponíveis em repositórios de software, para descobrir informações sobre o sistema [96]. Como base para este mapeamento sistemático, foram considerados os conceitos de inteligência analítica apresentados nesta Seção. Esta Seção contempla os conceitos de inteligência analítica, os tipos de análise que podem ser feitos com base na inteligência analítica, as necessidades de utilização da inteligência analítica por engenheiros de software e as definições das áreas do conhecimento da Engenharia de Software.

2.2.1 Inteligência Analítica

A inteligência analítica permite aos engenheiros de software realizarem a exploração e análise de dados, a fim de obter informações detalhadas e baseadas em dados do próprio software, para tomar de decisão sobre a condução de seu projeto e seus serviços [241]. Pesquisas têm relatado o uso da inteligência analítica para entender sistemas de software, propagação de mudanças, predição e identificação de defeitos e entender a dinâmica de equipes de desenvolvimento, melhorar a experiência do usuário, a reusabilidade de código, a automação de técnicas de mineração de repositórios [96] e a qualidade do software [242] e capacitar os indivíduos e equipes de desenvolvimento [156], entre outras atividades.

A inteligência analítica na Engenharia de Software alcançou sucesso substancial tanto em pesquisa quanto na prática de desenvolvimento [98]. Um dos papéis da inteligência analítica na Engenharia de Software é oferecer recomendações práticas para melhorar o projeto [156]. Ou seja, uma ferramenta de inteligência analítica pode recomendar aos programadores que realizem uma ação baseada no

histórico de versões, por exemplo, "programadores que alteraram esta função também alteraram..." [248]. Dessa maneira, tem-se a intenção de reduzir erros por alterações incompletas no código e nos problemas de acoplamento.

Projetos de software tendem a continuar crescendo em tamanho e complexidade. Assim, o esforço para analisar os dados disponíveis em repositórios de software é crescente [36]. A dificuldade de interpretar dados tem sido solucionada mediante a extração das métricas de software e a utilização de técnicas estatísticas [246].

Pesquisadores do campo da Engenharia de Software têm desenvolvido abordagens para extrair informações pertinentes e descobrir relações e tendências de repositórios no contexto da evolução do software [120]. A inteligência analítica no desenvolvimento de software torna possível a utilização de indicadores para a tomada de decisão. Sem esses indicadores, desenvolvedores e gestores contam apenas com intuição e a experiência para embasar decisões. Ferramentas online e ou integradas ao ambiente de desenvolvimento de software estão sendo utilizadas para esse fim [98]. Ou seja, oportunidades estão sendo criadas para serem exploradas nesta área.

2.2.2 Tipos de Análise da Inteligência Analítica

A inteligência analítica fornece aos engenheiros de software a possibilidade de analisar os projetos sob sua responsabilidade. Essas análises são úteis para saber o que está acontecendo, como está acontecendo, quais são melhores ações a serem tomadas e o que pode acontecer ao projeto de software.

Nesta Seção, apresentam-se os tipos de análise fornecidos pelas ferramentas de inteligência analítica aos engenheiros de software, com base no trabalho de Buse e Zimmermann [37].

Buse e Zimmermann [37] realizaram um *survey* com desenvolvedores e gerentes da indústria de software para descobrir como a inteligência analítica pode contribuir para embasar decisões no desenvolvimento de sistemas. Os autores categorizaram os tipos de analise, os quais se encontram descritos conforme o Quadro 2.1.

Os tipos de analise são caracterizados de acordo com o tempo (passado, presente e futuro) e a técnica aplicada (exploração, análise e experimentação). Com base nessa categorização, é possível responder a questões como: O que está acontecendo? Como e por que isso aconteceu? O que está acontecendo agora? Qual a melhor ação? O que acontecerá? O que de melhor ou pior pode acontecer? [37].

Quadro 2.1 Tipos de Análise da Inteligência Analítica [37]

	Passado	Presente	Futuro
Exploração	**Tendência** Quantifica como os artefatos estão mudando.	**Alerta** Informa mudanças incomuns nos artefatos quando elas acontecem.	**Previsão** Prevê eventos baseados em tendências.
Análise	**Sumarização** Características sucintas de artefatos-chaves ou grupos de artefatos.	**Sobreposição** Compara artefatos ou históricos de desenvolvimento de forma interativa.	**Objetivos** Descobre como os artefatos estão mudando com os respectivos objetivos.
Experimentação	**Modelagem** Caracteriza o comportamento normal do desenvolvimento.	**Comparação** Compara artefatos para obter as melhores práticas.	**Simulação** Testa condições antes de acontecerem.

2.2.3 Necessidades dos Engenheiros de Software

Com o intuito de descobrir quais são os questionamentos sobre o projeto de software, a inteligência analítica pode ajudar a responder aos engenheiros de software. Begel e Zimmermann [27] realizaram dois *surveys* na Microsoft e categorizaram 145 questões que os engenheiros de software gostariam que os cientistas de dados respondessem. Dessa maneira, torna-se possível verificar como a inteligência analítica pode auxiliar gerentes e desenvolvedores em suas atividades. Essas 145 questões foram categorizadas em 12 grupos. O resultado desta categorização é apresentado no Quadro 2.2.

Quadro 2.2 Necessidades dos Engenheiros de Software com a Inteligência Analítica [27]

Medições de defeitos (Bugs) Onde os defeitos são encontrados, os mais comuns, seu ciclo de vida e o custo para consertar defeitos.	**Prática de Desenvolvimento** Relacionado ao código, depuração, perfil de desempenho, refatoração, branching em repositório, revisão de código, comentários e documentação de código. Estimativa de esforço e riscos devido as mudanças no código.	**Melhores Práticas** Melhores conduções no projeto de software. Como técnicas adequadas de migrações entre as versões de software, a melhor maneira de controlar os itens de trabalho, o momento certo para usar métodos formais para a análise de software ou quais critérios devem influenciar a decisão de usar uma linguagem de programação específica ou API
Prática de Teste Cobre automação de teste, estratégia de teste, teste unitário, desenvolvimento orientado a testes, processos de teste,	**Avaliação da Qualidade** Trata sobre otimização de código, melhores métricas, código duplicado e	**Serviços** Forte relação com o desenvolvimento na *nuvem*, os efeitos sobre a retenção de clientes e monetização

escritas, infraestruturas, medidas e impactos nos defeitos.	problemas de qualidade de software.	causadas nas versões do software.
Clientes e Requisitos Relacionado aos interesses do cliente, gosto do cliente, compatibilidade com versões anteriores. Investimento em especificações técnicas, custo com o aumento de clientes.	**Ciclo de Vida do Desenvolvimento de Software** Como o tempo de desenvolvimento deve ser alocado entre planejamento, projeto, codificação e teste, impacto dessas alocações no software.	**Processo de Desenvolvimento de Software** Funcionamento das metodologias de processos de software e seus impactos na empresa.
Produtividade Investigar a métricas de qualidade para medir produtividade dos desenvolvedores, como monitorar a produtividade da equipe ou do indivíduo.	**Equipe e Colaboração** Formação de equipes, funções de desenvolvimento, quantidade de recursos e práticas para gestão do conhecimento.	**Reuso e Compartilhamento de Componentes** Reuso de código, seus componentes, custo de reaproveitamento de código.

2.2.4 Áreas do Conhecimento da Engenharia de Software

Com a finalidade de reconhecer as áreas da Engenharia de Software em que estão sendo aplicadas as técnicas de inteligência analítica, este artigo utilizou como base quinze áreas do conhecimento da engenharia de software descritas no guia SWEBOK (*Software Engineering Body of Knowledge*) [171]. Cada área pode ser descrita como:

- **Requisitos de Software -** concentra a elicitação, análise, especificação, validação e gerenciamento de requisitos de software.
- **Projeto de Software** - define a arquitetura, componentes, interfaces, características do sistema e seus componentes.
- **Construção de Software –** refere-se aos detalhes de criação de software, codificação, verificação, testes de unidade, integração e *debugging*.
- **Teste de Software -** atividades de teste de software, objetivos de teste, técnicas, avaliação, performance e medidas de teste, processos de teste e ferramentas.
- **Manutenção de Software -** recorrente às modificações no software, preservando sua integridade.
- **Gerenciamento de Configuração de Software -** configuração física e funcional do sistema de software que armazena e organiza artefatos arquivados do projeto de software, controlando suas alterações.
- **Gestão da Engenharia de Software -** gerencia as atividades de planejamento, coordenação, medidas, monitoramento, controle e relatórios dos produtos e serviços de software, eficiência das entregas e os benefícios às partes interessadas.
- **Processos de Engenharia de Software -** consistem em atividades inter-relacionadas da Engenharia de Software durante o ciclo de vida do software. Facilitam o entendimento, comunicação e coordenação dos envolvidos para o gerenciamento de projetos de software.
- **Modelos e Métodos da Engenharia de Software -** impõem estrutura sobre a Engenharia de Software, com o objetivo de fazer as atividades sistemáticas com maior sucesso.
- **Qualidade de Software -** garante que as especificações do produto sejam satisfeitas.
- **Prática Profissional em Engenharia de Software –** concentra os conhecimentos, habilidades, atitudes, responsabilidades e maneiras éticas dos profissionais de software, bem como a dinâmica do grupo e sua comunicação.
- **Economia em Engenharia de Software -** relacionada à tomada de decisão no contexto de negócio. Relaciona-se ao custo e ao risco de produtos de software e dos ecossistemas do projeto de software.
- **Fundamentos da Computação** - propõem as práticas computacionais para a Engenharia de Software. Ou seja, é o núcleo da Engenharia de Software com o entendimento da Ciência da Computação.
- **Fundamentos Matemáticos -** cobrem as técnicas básicas para identificar o conjunto de regras para o raciocínio no contexto de sistemas.
- **Fundamentos da Engenharia -** resultam em habilidades e técnicas usadas sistematicamente na Engenharia de Software. Relacionadas a estatística, medidas, modelos, simulações, protótipos e análises de causa raiz.

2.3 TRABALHOS RELACIONADOS

Trabalhos de revisão e mapeamento sistemáticos da literatura têm contribuído para entender o que está sendo pesquisado na Engenharia de Software. Heckman e Williams [101] realizaram uma revisão sistemática da literatura para identificar as técnicas de análise estática automatizada que identifiquem potenciais anomalias em código-fonte. De 17.571 artigos, selecionaram 21 trabalhos relevantes para a análise das técnicas de análise estática automatizada. Semelhante ao trabalho de Heckman e Williams [101], este mapeamento sistemático da literatura procurou identificar técnicas que auxiliem os engenheiros de software. Porém, o foco deste mapeamento está associado à utilização da inteligência analítica como técnica que apoie as atividades dos engenheiros de software..

Demeyer et al. [61] aplicaram a abordagem de mineração de textos *N-Gram* sobre os artigos da Conferência MSR, analisando as tendências que estão seguindo os artigos da conferência de 2004 a 2012. Os autores verificaram os termos mais frequentes e os menos frequentes, os casos mais utilizados e os menos utilizados, as infraestruturas utilizadas e informações práticas que estão sendo estudadas. Como resultados, descobriram que muitos estudos sobre evolução de software estão sendo realizados e poucos estudos sobre

padrões de projeto. Nas pesquisas, são frequentemente utilizados softwares livres, como o Eclipse, que dominam as pesquisas. Porém, poucos casos da indústria são relatados. O CVS e o Bugzilla são geralmente utilizados, sendo que o SVN, o Git e o Jira estão ganhando popularidade. Existe a tendência de obter dados de repositórios de comunicação, como e-mail e o *Stackoverflow*. O uso de técnicas de desenvolvimento e testes são amplamente discutidas nas pesquisas, contudo o ganho da utilização das técnicas para as organizações nem sempre é considerada. Ao contrário do trabalho de Demeyer et al. [61], o presente mapeamento não focou em uma única conferência, ampliando a busca para várias bases. Ele teve a intenção de descobrir o direcionamento das pesquisas nessa área. Contudo, a ideia deste trabalho é revelar a aplicabilidade da inteligência analítica na Engenharia de Software, e não apenas visualizar as tendências da conferência.

Novais et al. [168] mapearam 146 artigos sobre a visualização da evolução de software, descrevendo as características da visualização de software utilizadas na Engenharia de Software. Novais et al. [168] perceberam a utilização de múltiplas métricas, estratégias e perspectivas e da carência da validação das abordagens propostas. Já o presente mapeamento sistemático direcionou seus esforços em descobrir as carências de pesquisas sobre inteligência analítica na Engenharia de Software.

A taxonomia na mineração de repositórios de software foi tratada por Kagdi [120]. Ele analisou 80 trabalhos para investigar as taxonomias da mineração de repositório na evolução de software. Esse trabalho teve a intenção de facilitar as comparações das abordagens na mineração de repositórios, para entender a evolução de sistemas de software. O entendimento da evolução de sistemas de software é um tópico relevante na mineração de repositórios. A mineração de repositórios de software é a base da inteligência analítica na Engenharia de Software. O presente mapeamento sistemático analisou trabalhos de mineração repositórios de software, para compreender a técnica de inteligência analítica na Engenharia de Software.

Robles [189] realizou uma revisão dos artigos publicados no *Workshop on Mining Software Repositories* (MSR) entre 2004 a 2006 e na *Working Conference on MSR* de 2007 a 2009. O intuito dessa revisão foi verificar os experimentos que pudessem ser replicados. Para isso, ele analisou os artigos empíricos, verificando se os dados e as ferramentas utilizadas nos artigos estavam disponíveis ao público. Do total de 171 artigos analisados, apenas 2 se mostraram aptos para replicação. No presente mapeamento foram analisados trabalhos empíricos e teóricos, não se preocupando com os dados para replicação.

2.4 METODOLOGIA DE PESQUISA

Como suporte no mapeamento sistemático [175] empreendido contou-se com o uso do protocolo de mapeamento de estudos proposto por Kitchenham [134, 135], o qual auxilia no direcionamento das pesquisas de revisões sistemáticas e dos mapeamentos sistemáticos da Engenharia de Software. Nesse protocolo, foram declaradas: necessidades e escopo da pesquisa, estratégia de busca dos artigos, os critérios de seleção e exclusão, dados extraídos e limitações e validações do trabalho.

2.4.1 Questões de Pesquisa

Como objetivo desta pesquisa, pretende-se descobrir o que está sendo produzido sobre a inteligência analítica no âmbito da Engenharia de Software. Para investigar as pesquisas realizadas nessa área, este artigo propõe-se a responder as seguintes questões de pesquisa:

Questão de pesquisa 1 (QP1): Quais são os tipos de análises a inteligência analítica está proporcionando à Engenharia de Software? Esta questão tem por objetivo apontar os tipos de análise dos dados de repositórios de software que têm sido frequentes na literatura. Pretende-se, por meio desta questão, descobrir como essas informações ajudam na compreensão de projetos de software.

Questão de pesquisa 2 (QP2): Quais são as áreas de conhecimento da Engenharia de Software que têm maior predominância no uso da inteligência analítica? A Engenharia de Software é composta por diversas áreas do conhecimento. Para compreender o modo como a inteligência analítica está sendo utilizada na Engenharia de Software, é importante direcionar a área de predominância desta técnica no desenvolvimento de software. Esta questão tem por objetivo nortear a utilização da inteligência analítica nas áreas de conhecimento da Engenharia de Software.

Questão de pesquisa 3 (QP3): As pesquisas em inteligência analítica têm suprido as necessidades de informações dos engenheiros de software? Por meio desta questão, pretende-se descobrir se as pesquisas disponíveis na literatura estão alinhadas com as necessidades reais dos engenheiros de software apresentadas por Begel e Zimmermann [27] e descrita na Seção 2 deste trabalho.

2.4.2 Estratégia de Busca

Este mapeamento buscou pelos descritores *software analytics, software intelligence, mining software repositories e software development analytics* nas bases da *ACM Digital Library, IEEE Xplore, Springer, Elsevier* e *Wiley*. Tais descritores foram referenciados por Zhang [240] em *Software analytics in practice: approaches and experiences,* o termo *software analytics* foi tratado por Buse e Zimmermann [36] em seu trabalho *Analytics for Software Development*. O termo *analytics foi* também usado por Hullett et al. [112] para tratar os dados que direcionam o desenvolvimento de software, mais especificamente o desenvolvimento de jogos digitais. Inspirados no termo b*usiness intelligence*, Hassan e Xie [98] chamaram de s*oftware intelligence* o uso de dados no desenvolvimento de software para a tomada de decisão. Devido ao fato de as pesquisas de inteligência analítica na Engenharia de Software tratarem sobre mineração de repositório software, o termo m*ining software repositories* foi utilizado para ampliar as possibilidades de encontrar trabalhos relevantes sobre o uso da inteligência analítica na Engenharia de Software. Dessa maneira, os descritores de busca utilizados neste artigo foram formulados de acordo com o Quadro 2.3.

Quadro 2.3 Descritores de Busca

Base de Busca	Descritor
ACM Digital Library	(Abstract:"software analytics" or Abstract:"software intelligence" or Abstract:"Mining Software Repositories" or Abstract:"Software Development Analytics")or (Title:"software analytics" or Title:"software intelligence" or Title:"Mining Software Repositories" or Title:"Software Development Analytics") for: ((Abstract:"software analytics" or Abstract:"software intelligence" or

	Abstract:"Mining Software Repositories" or Abstract:"Software Development Analytics")or (Title:"software analytics" or Title:"software intelligence" or Title:"Mining Software Repositories" or Title:"Software Development Analytics")) and ("software analytics" or "software intelligence" or "Mining Software Repositories" or "Software Development Analytics")
IEEE Xplore	((("software analytics") OR "software intelligence") OR "Mining Software Repositories") OR "Software Development Analytics")
Springer	("software analytics" OR "software intelligence" OR "Mining Software Repositories" OR "Software Development Analytics")
Elsevier	("software analytics") or ("software intelligence") or ("Mining Software Repositories") or ("Software Development Analytics")
Wiley	"software analytics" in All Fields OR "software intelligence" in All Fields OR "Mining Software Repositories" in All Fields OR "Software Development Analytics" in All Fields

Visto que grande parte das pesquisas na busca automática retornou das conferências MSR (*Working Conference on Mining Software Repositories*) e a ICSE (*International Conference on Software Engineering*), foram adicionadas a esta pesquisa os seus artigos premiados como mais influentes para ampliar a possibilidade de encontrar artigos relevantes.

2.4.3 Critérios de Inclusão/Exclusão

Como critério de inclusão, foram verificados os artigos que tratavam de temas da Engenharia de Software utilizando a inteligência analítica; ou seja, artigos que usaram as técnicas de análise dos dados de repositórios de software ou artigos teóricos que tratavam sobre o tema. Neste artigo, contou-se com apenas artigos escritos na língua inglesa. Foram incluídos os artigos que continham os descritores de busca no título, resumo e palavras-chaves e que no corpo do texto tratavam explicitamente da Engenharia de Software. Devido à falta de padronização dos termos para se tratar o tema, os artigos colhidos manualmente foram incluídos pela interpretação do título e do resumo, e não pelos descritores de busca. Foram analisados artigos completos ou que exibam resultados parciais da pesquisa.

Foram excluídos artigos que não tratavam de forma explícita da Engenharia de Software e cujos descritores de busca estavam apenas citados na referência bibliográfica, no cabeçalho ou no rodapé do artigo. Foram excluídos também livros, seções de livros, entrevistas, resumos e chamadas para conferências e *workshops* e editoriais em geral. Também, foram eliminadas as duplicidades de artigos que estavam em mais de uma base. Apurou-se que 5 dos artigos retornados na busca da base *Wiley* e 1 na base da *Springer* não estavam com seus textos disponíveis, razão pela qual não foram considerados na análise deste mapeamento.

2.4.4 Seleção dos Artigos

A quantidade de artigos encontrados nas bases de busca por meio da estratégia de busca e após a aplicação dos critérios de seleção está retratada na Tabela 2.1. A Figura 2.1 apresenta o fluxo como os artigos foram refinados.

Tabela 2.1 Quantidade de Artigos por Base de Dados

Base de Busca	Quantidade de Artigos encontrados	Quantidade de Artigos Após Critério de Seleção
ACM Digital Library	78	29
IEEE Xplore	350	107
Springer	182	27
Elsevier	436	27
Wiley	228	12
MSR	31	30
ICSE	27	3
Total	1332	235

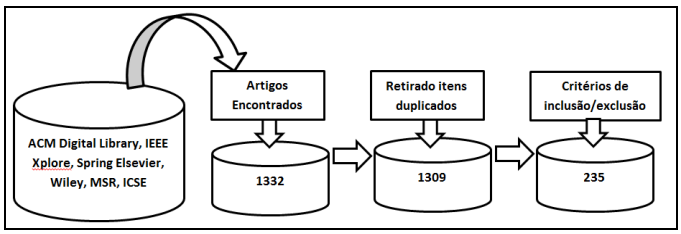


Figura 2.1 Refinamento dos Artigos para o Mapeamento Sistemático da literatura.

2.4.5 Extração dos Dados

Para melhor analisar os artigos selecionados pelo critério de inclusão deste mapeamento sistemático, foram extraídos dos artigos os seguintes dados: título da publicação, autores, local de publicação, ano de publicação, tipo de questão da pesquisa, área de conhecimento da Engenharia de Software, condução da pesquisa, tipos de artefatos produzidos, tipos de análise da inteligência analítica e necessidades dos engenheiros de software.

Como suporte para a classificação dos artigos deste mapeamento foram utilizadas categorias disponíveis na literatura. Os trabalhos de Shaw [200] [201] forneceram base para classificar os tipos de questões de pesquisa. Estes trabalhos citam que as pesquisas em Engenharia de Software respondem às perguntas sobre método ou meio de desenvolvimento, projeto, avaliação ou análise de um caso particular; generalização ou caracterização, método para análise ou avaliação e estudo de viabilidade ou exploração.

Para realizar a classificação das áreas de conhecimento da Engenharia de Software foram utilizadas suas 15 áreas referenciadas pelo SWEBOK v3 no *Guide to the Software Engineering Body of Knowledge* [171], descritos na Seção 2 desse trabalho.

Para determinar as conduções das pesquisas dos artigos analisados, foram utilizados os conceitos apresentados no trabalho

Experimentation in Software Engineering de Wohlin et al. [233], no qual os autores citam as conduções de pesquisa em Engenharia de Software, como experimento, estudo de caso, *survey*, revisão sistemática da literatura e teoria em Engenharia de Software.

A classificação dos tipos de artefatos produzidos foi baseada no trabalho *Design Science Research Evaluation*, de Peffers et al. [174], no qual os autores descrevem que os artefatos podem ser classificados como: algoritmo, construtor, *framework*, instanciação, método ou modelo.

Buse e Zimmermann [37] caracterizaram os tipos de análise mais comuns realizados com as ferramentas de inteligência analítica descritos no Quadro 2.1.

Begel e Zimmermann [27] categorizaram as necessidades dos engenheiros de software que podem ser supridas pela inteligência analítica. As descrições dessas necessidades podem ser vista conforme o Quadro 2.2.

2.5 RESULTADOS

2.5.1 Análises Descritivas

Os artigos analisados no mapeamento apontam o aumento do número de pesquisas sobre a inteligência analítica na Engenharia de Software nas últimas décadas. O primeiro *workshop* sobre mineração repositórios de software foi realizado em conjunto com a conferência ICSE, em 2004 [61]. Sete artigos que constam na base deste mapeamento datam do início de 2015, porém foram retirados da análise da Figura 2.2. No momento em que este artigo foi escrito, ainda não era possível analisar a real tendência das pesquisas de inteligência analítica na Engenharia de Software no ano de 2015.

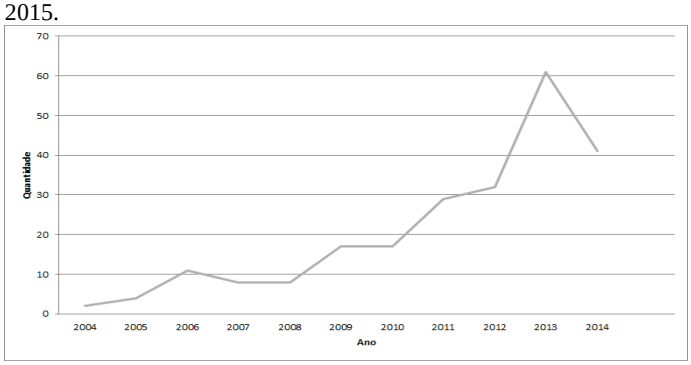


Figura 2.2 Quantidade de Publicações por Ano.

Dentre os tipos de questões de pesquisas utilizados nos artigos analisados 35% foram caracterizados como generalização ou caracterização, 18% procuraram um método para análise ou avaliação e 16% um método ou meio de desenvolvimento, estudo de viabilidade ou exploração e projeto, avaliação ou análise de um caso particular.

O tipo de artefato mais frequente nas pesquisas foram os métodos. A condução de pesquisa mais utilizadas para validar esses artefatos foi o estudo de caso, como exibido pela Figura 2.3.

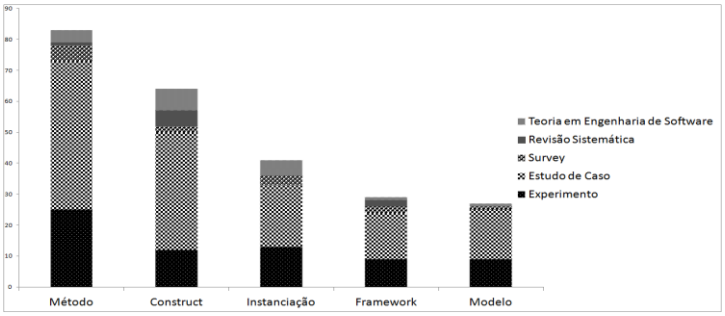


Figura 2.3 Artefato versus Validação da Pesquisa.

Novos métodos têm sido validados por meio de estudos de caso para entender os problemas enfrentados por desenvolvedores [19] caracterizar *commits* [2], estimar esforço [42], gerir conhecimento [60] e caracterizar defeitos [206] entre outros.

2.5.2 QP1 – Quais são os tipos de análise que a inteligência analítica está proporcionando à Engenharia de Software?

Os tipos de análise de Tendências, referenciados na Seção 2.2, estão presentes nas análises de 48 artigos deste mapeamento e a modelagem em 45 artigos. Entre os 235 artigos analisados, apenas 1 referenciou a inteligência analítica como análise de simulação, conforme a Figura 2.4 e Tabela 2.2. De acordo com os resultados obtidos a maioria das pesquisas vislumbram entender o que tem acontecido com o sistema, indicando um interesse em entender as mudanças dos artefatos dos repositórios de software, apresentando múltiplas visões ou construindo uma representação do projeto de software. Poucos se arriscaram a realizar análises que detectem eventos que impliquem no funcionamento do software ou indiquem o que aconteceria ao projeto por meio de condições alternativas.

Este mapeamento confirma os resultados do trabalho de Buse e Zimmermann que dizem que tanto desenvolvedores quanto gerentes acham mais importante compreender o passado do que tentar prever o futuro [37].

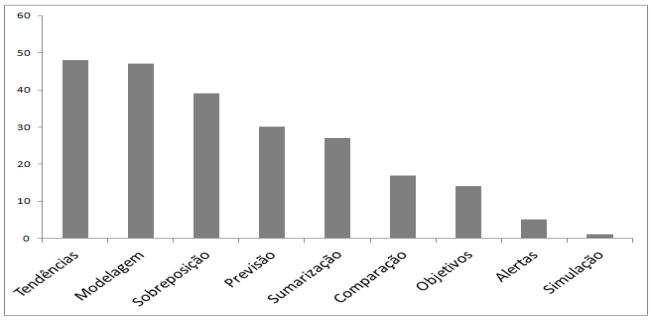


Figura 2.4 Tipos de Análises na Inteligência Analítica.

Tabela 2.2 Artigos do Mapeamento Sistemático por Tipos de Análises

Tipos de Análise	Artigos
Tendências	[63] [218] [90] [130] [120] [143] [188] [114] [59] [13] [195] [140] [72] [25] [184] [22] [8] [2] [58] [47] [74] [170] [185] [23] [17] [162] [227] [245] [164] [73] [219] [79] [28] [178] [16] [128] [20] [82] [187] [237] [199] [215] [19] [203] [39] [38] [193] [211]
Modelagem	[191] [71] [229] [165] [241] [110] [60] [205] [103] [10] [123] [106] [161] [239] [62] [208] [86] [78] [222] [111] [24] [131] [198] [155] [95] [242] [144] [99] [192] [9] [42] [228] [147] [230] [53] [249] [14] [46] [91] [88] [89] [202] [146] [34] [172] [87] [85]
Sobreposição	[196] [194] [116] [139] [69] [1] [52] [232] [50] [51] [223] [236] [12] [29] [104] [67] [55] [125] [127] [93] [214] [167] [129] [124] [176] [115] [4] [6] [126] [221] [220] [100] [234] [57] [154] [109] [179] [105] [40]
Previsão	[43] [173] [132] [98] [31] [159] [166] [5] [32] [158] [117] [118] [30] [64] [151] [216] [113] [33] [122] [48] [119] [68] [66] [224] [217] [56] [65] [248] [45] [244]
Sumarização	[142] [177] [18] [181] [149] [81] [153] [212] [207] [206] [157] [136] [83] [204] [180] [210] [84] [3] [35] [15] [92] [145] [182] [138] [108] [75] [190]
Comparação	[235] [137] [133] [243] [163] [169] [77] [150] [141] [41] [213] [238] [49] [197] [7] [107] [76]
Objetivos	[36] [26] [160] [121] [231] [246] [186] [54] [44] [152] [70] [247] [97] [183]
Alertas	[225] [209] [11] [148] [21]
Simulação	[226]
Artigos sem analises (apenas teóricos)	[102] [196] [37] [189] [96] [168] [80]

2.5.3 QP2 – Quais são as áreas de conhecimento da Engenharia de Software que têm maior predominância no uso da inteligência analítica?

A maior parte da utilização da inteligência analítica na Engenharia de Software se concentra na área de conhecimento da Manutenção de Software, seguido pela Construção de Software e Fundamentos da Computação, referenciados na Seção 2.2 deste artigo. A utilização da inteligência analítica na área da Manutenção de Software é bastante expressiva quando comparada a outras áreas.

Observa-se uma demanda das pesquisas nas áreas de Gestão de Engenharia de Software e de Práticas Profissionais em Engenharia de Software, que vem crescendo desde 2008, conforme mostra a Figura 2.5. Essas pesquisas contribuem para o gerenciamento de projetos de software e permitem analisar a dinâmica de equipes de desenvolvimento de software. Dessa maneira, a inteligência analítica tem sido utilizada para gerenciar incidentes de serviços online [148], planejar o projeto e otimizar as atividades de correção de defeitos [167], obter conhecimento sobre o risco do projeto de software [44], estimar esforço em projetos de software [66, 191], investigar redes de conhecimento em equipes de desenvolvimento de software [60] e reconhecer as habilidades e conhecimentos dos desenvolvedores de software [19, 162], entre outras atividades.

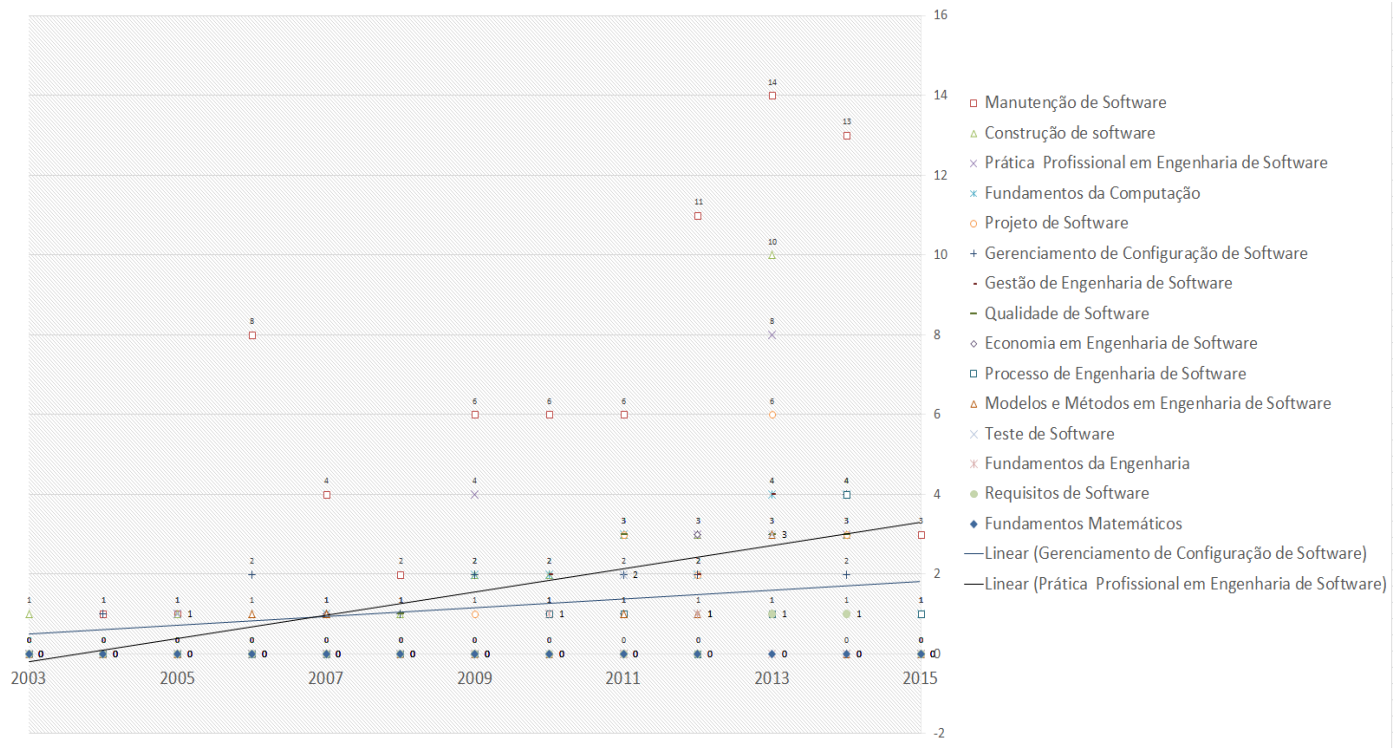


Figura 2.5. Áreas da Engenharia de Software versus Ano.

2.5.4 QP3 – As pesquisas em inteligência analítica têm suprido as necessidades de informações dos engenheiros de software?

A maioria das pesquisas sobre inteligência analítica na Engenharia de Software presentes neste mapeamento procurou suprir as necessidades de práticas de desenvolvimento. Conforme a pesquisa de Begel e Zimmermman [27], em que os autores realizaram um *survey* com os profissionais da Microsoft para descobrir as questões dos engenheiros de software que podem ser respondidas com a inteligência analítica. A categoria de maior destaque foi Prática de Desenvolvimento, descrita na fundamentação teórica deste artigo. Em contrapartida, as questões sobre Práticas de Teste, Processos de Desenvolvimento de Software e Ciclo de Vida de Desenvolvimento, bastante apontadas no *survey* de Begel e Zimmermman [27], não foram totalmente refletidas nas pesquisas analisadas neste mapeamento, como aponta a Figura 2.6. O mapeamento mostra que as pesquisas ainda não estão totalmente alinhadas com as necessidades práticas dos engenheiros de software descritas por Begel e Zimmermman [27].

Ressalta-se que a pesquisa realizada por Begel e Zimmermman [27] focou os engenheiros de software da Microsoft. Os próprios autores apontam esse fator como ameaça à validade da pesquisa. Para reforçar os achados, é encorajada a replicação dessa pesquisa em outras instituições. Contudo, o presente mapeamento apoia-se nos resultados deste trabalho por estar bem embasado, contando com a participação de 203 engenheiros de software no primeiro *survey* e de 607 no segundo *survey*. Contudo, espera-se que o presente mapeamento possa inspirar novas pesquisas em inteligência analítica para suprir as necessidades dos engenheiros de software que apresentaram maior carência de pesquisa.

Os resultados deste mapeamento e do *survey* [27] podem ser visualizados na Figura 2.6.

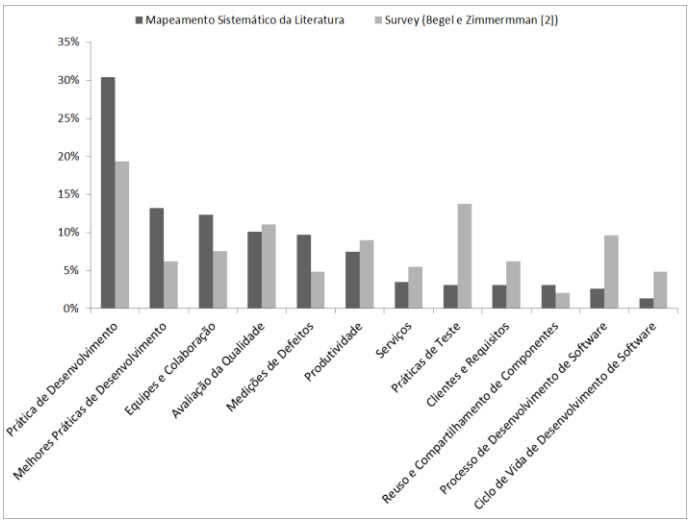


Figura 2.6 - Quantidade de artigos classificados no mapeamento sistemático da literatura versus necessidades dos engenheiros de software com inteligência analítica [27].

2.5.5 Interações das Categorias

As interações das categorias analisadas neste mapeamento entre as áreas do conhecimento da Engenharia de Software, os tipos de análises e as necessidades dos engenheiros de software podem ser verificadas na Figura 2.7.

Medições de Defeitos	Prática de Desenvolvimento	Melhores Práticas de Desenv.	Práticas de Teste	Avaliação da Qualidade	Serviços	Clientes e Requisitos	Ciclo de Vida de Desenvolv. de Soft.	Processo de Desenvol. de Soft.	Equipes e Colaboração	Reuso e Compartilh. de Compon.	Produtividade		Requisitos de Software	Projeto de Software	Construção de software	Teste de Software	Manutenção de Software	Gerenciamento de Config. De Soft.	Gestão de Eng. de Soft.	Processo de Enge. de Soft.	Modelos e Métodos em Eng. de Soft.	Qualidade de Software	Prática Profissional em Eng. de Soft.	Economia em Eng. de Soft.	Fundamentos da Computação	Fundamentos Matemáticos	Fundamentos da Engenharia
4	21	3	4	5	1	3	0	1	2	1	4	Tendências	0	3	3	4	17	2	1	1	0	4	3	6	2	0	1
5	8	7	0	6	3	2	0	1	2	4	2	Sobreposição	0	1	8	1	13	2	1	1	3	4	2	1	3	0	0
4	11	7	0	1	1	2	1	1	15	1	4	Modelagem	1	2	6	0	9	9	2	3	0	1	9	1	5	0	0
7	12	3	1	2	0	0	1	0	1	1	2	Previsão	0	1	1	1	16	0	4	1	2	2	0	0	1	0	1
2	8	1	0	4	0	1	0	2	8	1	1	Sumarização	1	2	4	0	8	1	1	1	0	1	4	1	2	0	0
0	5	0	0	2	0	0	1	0	1	0	5	Objetivos	0	1	1	0	4	0	4	1	2	0	1	0	0	0	0
1	2	8	1	3	2	0	0	0	0	0	0	Comparação	0	2	3	1	3	1	0	0	0	0	1	1	4	0	1
1	1	0	1	0	1	0	0	1	0	0	0	Alertas	0	2	0	1	1	0	1	0	0	0	0	0	1	0	0
0	1	0	0	0	0	0	0	0	0	0	0	Simulação	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0

Figura 2.7 Necessidades dos engenheiros de software versus tipos de análises versus áreas do conhecimento da Engenharia de software.

Os tipos de análises de tendências, sobreposição, previsão, sumarização estiveram presentes nas pesquisas com foco na área de Manutenção de Software, atendendo às necessidades de prática de desenvolvimento. Além da Manutenção de Software, o tipo de análise de modelagem se mostrou focado a atender às necessidades de equipe e colaboração utilizadas nas áreas de Gerenciamento de Configuração de Software e na Prática Profissional de Engenharia de Software. O tipo de análise objetivos, além de estar presente na área de manutenção de software e da prática de desenvolvimento está fortemente ligado às pesquisas na área de Gestão de Engenharia de Software, atendendo às necessidades ligadas a produtividade. O tipo de análise de comparação tem sido pesquisado na área de Fundamentos da Computação, objetivando produzir melhores práticas de desenvolvimento. Nota-se que análises de comparação estão sendo usadas na área de Fundamentos da Computação, além das áreas de Manutenção e Construção. Em Fundamentos da Computação, as pesquisas estão interessadas em melhorar a performance [137, 197], recuperação da informação [238] e o aumentar a interoperabilidade em repositórios de software [77].

2.6 AMEAÇAS À VALIDADE DA PESQUISA

Alguns fatores podem comprometer os resultados deste mapeamento, tal como os descritores de busca, que, apesar de bem embasados, podem não ter englobado todos os trabalhos relevantes sobre inteligência analítica. Este mapeamento apenas referenciou artigos escritos em língua inglesa. Entende-se que, por se tratar de uma forma interpretativa, a categorização dos artigos pode ser considerada subjetiva. Apontam-se, ainda, os critérios de seleção dos artigos e a quantidade de bases verificadas, que poderiam ter sido ampliadas para aumentar as chances de encontrar trabalhos relevantes.

2.7 CONCLUSÃO E TRABALHOS FUTUROS

Este trabalho apresentou um mapeamento sistemático da literatura sobre o uso de inteligência analítica na Engenharia de Software. Constatou-se que a maioria das pesquisas procurou generalizar ou caracterizar elementos no desenvolvimento de sistemas, bem como desenvolver métodos para a utilização da inteligência analítica. Dos 235 artigos analisados, houve apenas uma referência de simulação para analisar dados em repositórios de software. A inteligência analítica é muito utilizada nas atividades de manutenção de software, seja para tratar defeitos ou erros, realizar a análise de impacto de mudanças ou analisar a evolução de software entre outros tópicos relativos à manutenção. Observou-se um interesse cada vez maior nos estudos sobre as práticas profissionais e gestão de projetos. O presente estudo indica que as necessidades reais dos engenheiros de software não estão sendo totalmente pesquisadas, carecendo de maiores investimentos na prática de teste e nos processos e ciclo de vida do desenvolvimento de software.

Para facilitar a reprodução e a análise dos dados deste mapeamento, os artigos analisados podem ser obtidos pelo link: http://migre.me/pDQXm.

Durante a leitura dos artigos outros termos que tratam sobre inteligência analítica puderam ser observados como: s*oftware mining, software repository mining, mining software, source repository mining, mining software engineering data e engineering intelligence analytics*. Desta maneira, como sugestão para a realização de trabalhos futuros, é interessante investigar os artigos retornados com esses descritores com a finalidade de ampliar os achados deste mapeamento. É importante frisar a dificuldade para encontrar termos que se encaixem nesse contexto, devido à utilização de diversos termos ou, até mesmo, à não utilização de

termos específicos para se tratar de inteligência analítica na Engenharia de Software.

Com este trabalho, espera-se que novas pesquisas sejam incentivadas para explorar a inteligência analítica na utilização da Engenharia de Software.

2.8 REFERÊNCIAS

[1] B. Adams, Zhen Ming Jiang, and A.E. Hassan. 2010. Identifying crosscutting concerns using historical code changes. *2010 ACM/IEEE 32nd International Conference on Software Engineering*, 305–314. DOI= http://doi.org/10.1145/1806799.1806846

[2] A. Alali, H. Kagdi, and J.I. Maletic. 2008. What's a Typical Commit? A Characterization of Open Source Software Repositories. *The 16th IEEE International Conference on Program Comprehension, 2008. ICPC 2008*, 182–191. DOI= http://doi.org/10.1109/ICPC.2008.24

[3] Omar Alam, Bram Adams, and Ahmed E. Hassan. 2012. Preserving knowledge in software projects. *Journal of Systems and Software* 85, 10: 2318–2330. DOI= http://doi.org/10.1016/j.jss.2012.03.028

[4] N. Ali, Y. Gueneuc, and G. Antoniol. 2013. Trustrace: Mining Software Repositories to Improve the Accuracy of Requirement Traceability Links. *IEEE Transactions on Software Engineering* 39, 5: 725–741. DOI= http://doi.org/10.1109/TSE.2012.71

[5] M. Allamanis and C. Sutton. 2013. Mining source code repositories at massive scale using language modeling. *2013 10th IEEE Working Conference on Mining Software Repositories (MSR)*, 207–216. DOI= http://doi.org/10.1109/MSR.2013.6624029

[6] Izzat Alsmadi and Hassan Najadat. 2011. Evaluating the change of software fault behavior with dataset attributes based on categorical correlation. *Advances in Engineering Software* 42, 8: 535–546. DOI= http://doi.org/10.1016/j.advengsoft.2011.03.010

[7] J. H. Andrews, L. C. Briand, and Y. Labiche. 2005. Is mutation an appropriate tool for testing experiments? [software testing]. *Proceedings. 27th International Conference on Software Engineering, 2005. ICSE 2005.*, 402–411. DOI= http://doi.org/10.1109/ICSE.2005.1553583

[8] M.F. Aniche, G.A. Oliva, and M.A. Gerosa. 2013. What Do the Asserts in a Unit Test Tell Us about Code Quality? A Study on Open Source and Industrial Projects. *2013 17th European Conference on Software Maintenance and Reengineering (CSMR)*, 111–120. DOI= http://doi.org/10.1109/CSMR.2013.21

[9] Tom Arbuckle. 2011. Studying software evolution using artefacts' shared information content. *Science of Computer Programming* 76, 12: 1078–1097. DOI= http://doi.org/10.1016/j.scico.2010.11.005

[10] L. Arjona Reina and G. Robles. 2012. Mining for localization in Android. *2012 9th IEEE Working Conference on Mining Software Repositories (MSR)*, 136–139. DOI= http://doi.org/10.1109/MSR.2012.6224272

[11] V. Arnaoudova, L.M. Eshkevari, M. Di Penta, R. Oliveto, G. Antoniol, and Y.-G. Gueheneuc. 2014. REPENT: Analyzing the Nature of Identifier Renamings. *IEEE Transactions on Software Engineering* 40, 5: 502–532. DOI= http://doi.org/10.1109/TSE.2014.2312942

[12] A. Bachmann and A. Bernstein. 2010. When process data quality affects the number of bugs: Correlations in software engineering datasets. *2010 7th IEEE Working Conference on Mining Software Repositories (MSR)*, 62–71. DOI= http://doi.org/10.1109/MSR.2010.5463286

[13] R. Bahsoon and W. Emmerich. 2007. Economics-Driven Software Mining. *Economics of Software and Computation, 2007. ESC '07. First International Workshop on the*, 3–3. DOI= http://doi.org/10.1109/ESC.2007.5

[14] Sushil Bajracharya and C. Lopes. 2009. Mining search topics from a code search engine usage log. *6th IEEE International Working Conference on Mining Software Repositories, 2009. MSR '09*, 111–120. DOI= http://doi.org/10.1109/MSR.2009.5069489

[15] Normi Sham Awang Abu Bakar. 2011. Using Language-Based Search in Mining Large Software Repositories. *Procedia - Social and Behavioral Sciences* 27: 160–168. DOI= http://doi.org/10.1016/j.sbspro.2011.10.594

[16] Steve Bannerman and Andrew Martin. 2011. A multiple comparative study of test-with development product changes and their effects on team speed and product quality. *Empirical Software Engineering* 16, 2: 177–210. DOI= http://doi.org/10.1007/s10664-010-9137-5

[17] F. Bantelay, M.B. Zanjani, and H. Kagdi. 2013. Comparing and combining evolutionary couplings from interactions and commits. *2013 20th Working Conference on Reverse Engineering (WCRE)*, 311–320. DOI= http://doi.org/10.1109/WCRE.2013.6671306

[18] Cezary Bartoszuk, Robert Dąbrowski, Krzysztof Stencel, and Grzegorz Timoszuk. 2013. On Quick Comprehension and Assessment of Software. *Proceedings of the 14th International Conference on Computer Systems and Technologies*, ACM, 161–168. DOI= http://doi.org/10.1145/2516775.2516806

[19] Anton Barua, Stephen W. Thomas, and Ahmed E. Hassan. 2014. What are developers talking about? An analysis of topics and trends in Stack Overflow. *Empir Software Eng* 19, 3: 619–654. DOI= http://doi.org/10.1007/s10664-012-9231-y

[20] Gabriele Bavota, Gerardo Canfora, Massimiliano Di Penta, Rocco Oliveto, and Sebastiano Panichella. 2014. How the Apache community upgrades dependencies: an evolutionary study. *Empirical Software Engineering*: 1–43. DOI= http://doi.org/10.1007/s10664-014-9325-9

[21] Gabriele Bavota, Abdallah Qusef, Rocco Oliveto, Andrea De Lucia, and Dave Binkley. 2014. Are test smells really harmful? An empirical study. *Empirical Software Engineering*: 1–43. DOI= http://doi.org/10.1007/s10664-014-9313-0

[22] G. Bavota, M. Linares-Vasquez, C. Bernal-Cardenas, M. Di Penta, R. Oliveto, and D. Poshyvanyk. 2014. The Impact of API Change- and Fault-Proneness on the User Ratings of Android Apps. *IEEE Transactions on Software Engineering* PP, 99: 1–1. DOI= http://doi.org/10.1109/TSE.2014.2367027

[23] G. Bavota, A. Qusef, R. Oliveto, A. De Lucia, and D. Binkley. 2012. An empirical analysis of the distribution of unit test smells and their impact on software maintenance. *2012 28th IEEE International Conference on Software*

Maintenance (ICSM), 56–65. DOI= http://doi.org/10.1109/ICSM.2012.6405253

[24] O. Baysal. 2013. Informing development decisions: From data to information. *2013 35th International Conference on Software Engineering (ICSE)*, 1407–1410. DOI= http://doi.org/10.1109/ICSE.2013.6606729

[25] O. Baysal, R. Holmes, and M.W. Godfrey. 2012. Mining usage data and development artifacts. *2012 9th IEEE Working Conference on Mining Software Repositories (MSR)*, 98–107. DOI= http://doi.org/10.1109/MSR.2012.6224305

[26] Andrew Begel, Yit Phang Khoo, and Thomas Zimmermann. 2010. Codebook: Discovering and Exploiting Relationships in Software Repositories. *Proceedings of the 32Nd ACM/IEEE International Conference on Software Engineering - Volume 1*, ACM, 125–134. DOI= http://doi.org/10.1145/1806799.1806821

[27] Andrew Begel and Thomas Zimmermann. 2014. Analyze This! 145 Questions for Data Scientists in Software Engineering. *Proceedings of the 36th International Conference on Software Engineering*, ACM, 12–23. DOI= http://doi.org/10.1145/2568225.2568233

[28] Thorsten Berger, Rolf-Helge Pfeiffer, Reinhard Tartler, et al. 2014. Variability mechanisms in software ecosystems. *Information and Software Technology* 56, 11: 1520–1535. DOI= http://doi.org/10.1016/j.infsof.2014.05.005

[29] N. Bettenburg and A. Begel. 2013. Deciphering the story of software development through frequent pattern mining. *2013 35th International Conference on Software Engineering (ICSE)*, 1197–1200. DOI= http://doi.org/10.1109/ICSE.2013.6606677

[30] Nicolas Bettenburg, M. Nagappan, and A.E. Hassan. 2012. Think locally, act globally: Improving defect and effort prediction models. *2012 9th IEEE Working Conference on Mining Software Repositories (MSR)*, 60–69. DOI= http://doi.org/10.1109/MSR.2012.6224300

[31] Pamela Bhattacharya. 2011. Using Software Evolution History to Facilitate Development and Maintenance. *Proceedings of the 33rd International Conference on Software Engineering*, ACM, 1122–1123. DOI= http://doi.org/10.1145/1985793.1986012

[32] J. Boaz Lee, A. Ihara, A. Monden, and K.-I. Matsumoto. 2013. Patch Reviewer Recommendation in OSS Projects. *Software Engineering Conference (APSEC, 2013 20th Asia-Pacific*, 1–6. DOI= http://doi.org/10.1109/APSEC.2013.103

[33] G. Bougie, C. Treude, D.M. German, and M. Storey. 2010. A comparative exploration of FreeBSD bug lifetimes. *2010 7th IEEE Working Conference on Mining Software Repositories (MSR)*, 106–109. DOI= http://doi.org/10.1109/MSR.2010.5463291

[34] João Brunet, Gail C. Murphy, Ricardo Terra, Jorge Figueiredo, and Dalton Serey. 2014. Do Developers Discuss Design? *Proceedings of the 11th Working Conference on Mining Software Repositories*, ACM, 340–343. DOI= http://doi.org/10.1145/2597073.2597115

[35] Magiel Bruntink. 2015. Towards base rates in software analytics: Early results and challenges from studying Ohloh. *Science of Computer Programming* 97, Part 1: 135–142. DOI= http://doi.org/10.1016/j.scico.2013.11.023

[36] Raymond P.L. Buse and Thomas Zimmermann. 2010. Analytics for Software Development. *Proceedings of the FSE/SDP Workshop on Future of Software Engineering Research*, ACM, 77–80. DOI= http://doi.org/10.1145/1882362.1882379

[37] R.P.L. Buse and T. Zimmermann. 2012. Information needs for software development analytics. *2012 34th International Conference on Software Engineering (ICSE)*, 987–996. DOI= http://doi.org/10.1109/ICSE.2012.6227122

[38] Oscar Callaú, Romain Robbes, Éric Tanter, and David Röthlisberger. 2013. How (and why) developers use the dynamic features of programming languages: the case of smalltalk. *Empirical Software Engineering* 18, 6: 1156–1194. DOI= http://doi.org/10.1007/s10664-012-9203-2

[39] Felivel Camilo, Andrew Meneely, and Meiyappan Nagappan. 2015. Do Bugs Foreshadow Vulnerabilities?: A Study of the Chromium Project. *Proceedings of the 12th Working Conference on Mining Software Repositories*, IEEE Press, 269–279. Retrieved May 26, 2016 from http://dl.acm.org/citation.cfm?id=2820518.2820551

[40] G. Canfora, L. Cerulo, and M. Di Penta. 2007. Identifying Changed Source Code Lines from Version Repositories. *Fourth International Workshop on Mining Software Repositories, 2007. ICSE Workshops MSR '07*, 14–14. DOI= http://doi.org/10.1109/MSR.2007.14

[41] G. Canfora, L. Cerulo, and M. Di Penta. 2009. Ldiff: An enhanced line differencing tool. *IEEE 31st International Conference on Software Engineering, 2009. ICSE 2009*, 595–598. DOI= http://doi.org/10.1109/ICSE.2009.5070564

[42] Andrea Capiluppi and Daniel Izquierdo-Cortázar. 2013. Effort estimation of FLOSS projects: a study of the Linux kernel. *Empir Software Eng* 18, 1: 60–88. DOI= http://doi.org/10.1007/s10664-011-9191-7

[43] M. Ceccarelli, L. Cerulo, G. Canfora, and M. Di Penta. 2010. An eclectic approach for change impact analysis. *2010 ACM/IEEE 32nd International Conference on Software Engineering*, 163–166. DOI= http://doi.org/10.1145/1810295.1810320

[44] Ching-Pao Chang. 2013. Mining software repositories to acquire software risk knowledge. *2013 IEEE 14th International Conference on Information Reuse and Integration (IRI)*, 489–496. DOI= http://doi.org/10.1109/IRI.2013.6642510

[45] M. Choetkiertikul, H. K. Dam, T. Tran, and A. Ghose. 2015. Characterization and Prediction of Issue-Related Risks in Software Projects. *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*, 280–291. DOI= http://doi.org/10.1109/MSR.2015.33

[46] Shaiful Alam Chowdhury and Abram Hindle. 2015. Mining StackOverflow to Filter out Off-topic IRC Discussion. *Proceedings of the 12th Working Conference on Mining Software Repositories*, IEEE Press, 422–425. Retrieved May 26, 2016 from http://dl.acm.org/citation.cfm?id=2820518.2820577

[47] M. Colaço, M. Mendonca, and F. Rodrigues. 2009. Data Warehousing in an Industrial Software Development Environment. *2009 33rd Annual IEEE Software Engineering Workshop (SEW)*, 131–135. DOI= http://doi.org/10.1109/SEW.2009.7

[48] M. Colaco, M. Mendonca, and F. Rodrigues. 2009. Mining Software Change History in an Industrial Environment. *XXIII Brazilian Symposium on Software Engineering, 2009. SBES '09*, 54–61. DOI= http://doi.org/10.1109/SBES.2009.8

[49] James R. Cordy and Chanchal K. Roy. 2014. Tuning research tools for scalability and performance: The NiCad experience. *Science of Computer Programming* 79: 158–171. DOI= http://doi.org/10.1016/j.scico.2011.11.002

[50] C.S. Corley, E.A. Kammer, and N.A. Kraft. 2012. Modeling the ownership of source code topics. *2012 IEEE 20th International Conference on Program Comprehension (ICPC)*, 173–182. DOI= http://doi.org/10.1109/ICPC.2012.6240485

[51] D. Correa, S. Lal, A. Saini, and A. Sureka. 2013. Samekana: A Browser Extension for Including Relevant Web Links in Issue Tracking System Discussion Forum. *Software Engineering Conference (APSEC, 2013 20th Asia-Pacific*, 25–33. DOI= http://doi.org/10.1109/APSEC.2013.15

[52] D. Correa and A. Sureka. 2013. Integrating Issue Tracking Systems with Community-Based Question and Answering Websites. *Software Engineering Conference (ASWEC), 2013 22nd Australian*, 88–96. DOI= http://doi.org/10.1109/ASWEC.2013.20

[53] D. Cubranic and G. C. Murphy. 2003. Hipikat: recommending pertinent software development artifacts. *25th International Conference on Software Engineering, 2003. Proceedings.*, 408–418. DOI= http://doi.org/10.1109/ICSE.2003.1201219

[54] J. Czerwonka, N. Nagappan, W. Schulte, and B. Murphy. 2013. CODEMINE: Building a Software Development Data Analytics Platform at Microsoft. *IEEE Software* 30, 4: 64–71. DOI= http://doi.org/10.1109/MS.2013.68

[55] B. Dagenais and M.P. Robillard. 2008. Recommending adaptive changes for framework evolution. *ACM/IEEE 30th International Conference on Software Engineering, 2008. ICSE '08*, 481–490. DOI= http://doi.org/10.1145/1368088.1368154

[56] Hoa Khanh Dam and Aditya Ghose. 2013. Supporting change impact analysis for intelligent agent systems. *Science of Computer Programming* 78, 9: 1728–1750. DOI= http://doi.org/10.1016/j.scico.2013.04.008

[57] Julius Davies, Daniel M. German, Michael W. Godfrey, and Abram Hindle. 2013. Software Bertillonage. *Empirical Software Engineering* 18, 6: 1195–1237. DOI= http://doi.org/10.1007/s10664-012-9199-7

[58] S. Davies and M. Roper. 2013. Bug localisation through diverse sources of information. *2013 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, 126–131. DOI= http://doi.org/10.1109/ISSREW.2013.6688891

[59] Steven Davies, Marc Roper, and Murray Wood. 2014. Comparing text-based and dependence-based approaches for determining the origins of bugs. *Journal of Software: Evolution and Process* 26, 1: 107–139. DOI= http://doi.org/10.1002/smr.1619

[60] Christian Del Rosso. 2009. Comprehend and analyze knowledge networks to improve software evolution. *J. Softw. Maint. Evol.: Res. Pract.* 21, 3: 189–215. DOI= http://doi.org/10.1002/smr.408

[61] S. Demeyer, A. Murgia, K. Wyckmans, and A. Lamkanfi. 2013. Happy Birthday! A trend analysis on past MSR papers. *2013 10th IEEE Working Conference on Mining Software Repositories (MSR)*, 353–362. DOI= http://doi.org/10.1109/MSR.2013.6624049

[62] M. Di Penta and D.M. German. 2009. Who are Source Code Contributors and How do they Change? *16th Working Conference on Reverse Engineering, 2009. WCRE '09*, 11–20. DOI= http://doi.org/10.1109/WCRE.2009.41

[63] Bogdan Dit, Michael Wagner, Shasha Wen, et al. 2014. ImpactMiner: A Tool for Change Impact Analysis. *Companion Proceedings of the 36th International Conference on Software Engineering*, ACM, 540–543. DOI= http://doi.org/10.1145/2591062.2591064

[64] J. Ekanayake, J. Tappolet, H.C. Gall, and A. Bernstein. 2009. Tracking concept drift of software projects using defect prediction quality. *6th IEEE International Working Conference on Mining Software Repositories, 2009. MSR '09*, 51–60. DOI= http://doi.org/10.1109/MSR.2009.5069480

[65] Jayalath Ekanayake, Jonas Tappolet, Harald C. Gall, and Abraham Bernstein. 2012. Time variance and defect prediction in software projects. *Empirical Software Engineering* 17, 4–5: 348–389. DOI= http://doi.org/10.1007/s10664-011-9180-x

[66] Emad A. El-Sebakhy. 2011. Functional networks as a novel data mining paradigm in forecasting software development efforts. *Expert Systems with Applications* 38, 3: 2187–2194. DOI= http://doi.org/10.1016/j.eswa.2010.08.005

[67] M. Erfani, I. Keivanloo, and J. Rilling. 2013. Opportunities for Clone Detection in Test Case Recommendation. *Computer Software and Applications Conference Workshops (COMPSACW), 2013 IEEE 37th Annual*, 65–70. DOI= http://doi.org/10.1109/COMPSACW.2013.11

[68] Jacqui Finlay, Russel Pears, and Andy M. Connor. 2014. Data stream mining for predicting software build outcomes using source code metrics. *Information and Software Technology* 56, 2: 183–198. DOI= http://doi.org/10.1016/j.infsof.2013.09.001

[69] C. Forbes, I. Keivanloo, and J. Rilling. 2012. Doppel-Code: A Clone Visualization Tool for Prioritizing Global and Local Clone Impacts. *Computer Software and Applications Conference (COMPSAC), 2012 IEEE 36th Annual*, 366–367. DOI= http://doi.org/10.1109/COMPSAC.2012.58

[70] Ying Fu, Meng Yan, Xiaohong Zhang, Ling Xu, Dan Yang, and Jeffrey D. Kymer. Automated classification of software change messages by semi-supervised Latent Dirichlet Allocation. *Information and Software Technology*. DOI= http://doi.org/10.1016/j.infsof.2014.05.017

[71] Kenji Fujiwara, Hideaki Hata, Erina Makihara, et al. 2014. Kataribe: A Hosting Service of Historage Repositories. *Proceedings of the 11th Working Conference on Mining Software Repositories*, ACM, 380–383. DOI= http://doi.org/10.1145/2597073.2597125

[72] S. Gala-Perez, G. Robles, J.M. Gonzalez-Barahona, and I. Herraiz. 2013. Intensive metrics for the study of the evolution of open source projects: Case studies from Apache Software Foundation projects. *2013 10th IEEE Working Conference on Mining Software Repositories (MSR)*, 159–168. DOI= http://doi.org/10.1109/MSR.2013.6624023

[73] Vahid Garousi and James Leitch. 2010. IssuePlayer: An extensible framework for visual assessment of issue management in software development projects. *Journal of Visual Languages & Computing* 21, 3: 121–135. DOI= http://doi.org/10.1016/j.jvlc.2010.03.001

[74] C. Gerlec, A. Krajnc, M. Heričko, and J. Boznik. 2011. Mining source code changes from software repositories. *Software Engineering Conference in Russia (CEE-SECR)*,

2011 7th Central and Eastern European, 1–5. DOI= http://doi.org/10.1109/CEE-SECR.2011.6188468

[75] Daniel M. German. 2006. A Study of the Contributors of PostgreSQL. *Proceedings of the 2006 International Workshop on Mining Software Repositories*, ACM, 163–164. DOI= http://doi.org/10.1145/1137983.1138022

[76] Daniel M. German and Julius Davies. 2011. Apples vs. Oranges?: An Exploration of the Challenges of Comparing the Source Code of Two Software Systems. *Proceedings of the 8th Working Conference on Mining Software Repositories*, ACM, 246–249. DOI= http://doi.org/10.1145/1985441.1985483

[77] G. Ghezzi and H.C. Gall. 2013. Replicating mining studies with SOFAS. *2013 10th IEEE Working Conference on Mining Software Repositories (MSR)*, 363–372. DOI= http://doi.org/10.1109/MSR.2013.6624050

[78] G. Ghezzi, M. Wursch, E. Giger, and H.C. Gall. 2012. An architectural blueprint for a pluggable version control system for software (evolution) analysis. *2012 2nd Workshop on Developing Tools as Plug-ins (TOPI)*, 13–18. DOI= http://doi.org/10.1109/TOPI.2012.6229803

[79] Michael W. Godfrey. 2015. Understanding software artifact provenance. *Science of Computer Programming* 97, Part 1: 86–90. DOI= http://doi.org/10.1016/j.scico.2013.11.021

[80] Jesús M. González-Barahona and Gregorio Robles. 2012. On the reproducibility of empirical software engineering studies based on data retrieved from development repositories. *Empirical Software Engineering* 17, 1–2: 75–89. DOI= http://doi.org/10.1007/s10664-011-9181-9

[81] Jesus M. Gonzalez-Barahona, Gregorio Robles, Israel Herraiz, and Felipe Ortega. 2014. Studying the laws of software evolution in a long-lived FLOSS project. *Journal of Software: Evolution and Process* 26, 7: 589–612. DOI= http://doi.org/10.1002/smr.1615

[82] Jesus M. Gonzalez-Barahona, Gregorio Robles, Martin Michlmayr, Juan José Amor, and Daniel M. German. 2009. Macro-level software evolution: a case study of a large software compilation. *Empirical Software Engineering* 14, 3: 262–285. DOI= http://doi.org/10.1007/s10664-008-9100-x

[83] J.M. Gonzalez-Barahona, D. Izquierdo-Cortazar, S. Maffulli, and G. Robles. 2013. Understanding How Companies Interact with Free Software Communities. *IEEE Software* 30, 5: 38–45. DOI= http://doi.org/10.1109/MS.2013.95

[84] Antonio González-Torres, Francisco J. García-Peñalvo, and Roberto Therón. 2013. Human–computer interaction in evolutionary visual software analytics. *Computers in Human Behavior* 29, 2: 486–495. DOI= http://doi.org/10.1016/j.chb.2012.01.013

[85] G. Gousios. 2013. The GHTorent dataset and tool suite. *2013 10th IEEE Working Conference on Mining Software Repositories (MSR)*, 233–236. DOI= http://doi.org/10.1109/MSR.2013.6624034

[86] G. Gousios and D. Spinellis. 2009. A platform for software engineering research. *6th IEEE International Working Conference on Mining Software Repositories, 2009. MSR '09*, 31–40. DOI= http://doi.org/10.1109/MSR.2009.5069478

[87] Georgios Gousios and Andy Zaidman. 2014. A Dataset for Pull-based Development Research. *Proceedings of the 11th Working Conference on Mining Software Repositories*, ACM, 368–371. DOI= http://doi.org/10.1145/2597073.2597122

[88] S. Grant and B. Betts. 2013. Encouraging user behaviour with achievements: An empirical study. *2013 10th IEEE Working Conference on Mining Software Repositories (MSR)*, 65–68. DOI= http://doi.org/10.1109/MSR.2013.6624007

[89] V. Guana, F. Rocha, A. Hindle, and E. Stroulia. 2012. Do the stars align? Multidimensional analysis of Android's layered architecture. *2012 9th IEEE Working Conference on Mining Software Repositories (MSR)*, 124–127. DOI= http://doi.org/10.1109/MSR.2012.6224269

[90] Monika Gupta. 2014. Nirikshan: Process Mining Software Repositories to Identify Inefficiencies, Imperfections, and Enhance Existing Process Capabilities. *Companion Proceedings of the 36th International Conference on Software Engineering*, ACM, 658–661. DOI= http://doi.org/10.1145/2591062.2591080

[91] Emitza Guzman, David Azócar, and Yang Li. 2014. Sentiment Analysis of Commit Comments in GitHub: An Empirical Study. *Proceedings of the 11th Working Conference on Mining Software Repositories*, ACM, 352–355. DOI= http://doi.org/10.1145/2597073.2597118

[92] Jongdae Han and Woosung Jung. 2014. Extracting communication structure of a development organization from a software repository. *Personal and Ubiquitous Computing* 18, 6: 1413–1421. DOI= http://doi.org/10.1007/s00779-013-0742-3

[93] M. Harman, Yue Jia, and Yuanyuan Zhang. 2012. App store mining and analysis: MSR for app stores. *2012 9th IEEE Working Conference on Mining Software Repositories (MSR)*, 108–111. DOI= http://doi.org/10.1109/MSR.2012.6224306

[94] A.E. Hassan. 2006. Mining Software Repositories to Assist Developers and Support Managers. *22nd IEEE International Conference on Software Maintenance, 2006. ICSM '06*, 339–342. DOI= http://doi.org/10.1109/ICSM.2006.38

[95] A.E. Hassan. 2006. Mining Software Repositories to Assist Developers and Support Managers. *22nd IEEE International Conference on Software Maintenance, 2006. ICSM '06*, 339–342. DOI= http://doi.org/10.1109/ICSM.2006.38

[96] A.E. Hassan. 2008. The road ahead for Mining Software Repositories. *Frontiers of Software Maintenance, 2008. FoSM 2008.*, 48–57. DOI= http://doi.org/10.1109/FOSM.2008.4659248

[97] Ahmed E. Hassan and Richard C. Holt. 2006. Replaying development history to assess the effectiveness of change propagation tools. *Empirical Software Engineering* 11, 3: 335–367. DOI= http://doi.org/10.1007/s10664-006-9006-4

[98] Ahmed E. Hassan and Tao Xie. 2010. Software Intelligence: The Future of Mining Software Engineering Data. *Proceedings of the FSE/SDP Workshop on Future of Software Engineering Research*, ACM, 161–166. DOI= http://doi.org/10.1145/1882362.1882397

[99] Lile Hattori, Marco D'Ambros, Michele Lanza, and Mircea Lungu. 2013. Answering software evolution questions: An empirical evaluation. *Information and Software Technology* 55, 4: 755–775. DOI= http://doi.org/10.1016/j.infsof.2012.09.001

[100] Lile Palma Hattori, Michele Lanza, and Romain Robbes. 2012. Refining code ownership with synchronous changes. *Empirical Software Engineering* 17, 4–5: 467–499. DOI= http://doi.org/10.1007/s10664-010-9145-5

[101] Sarah Heckman and Laurie Williams. 2011. A Systematic Literature Review of Actionable Alert Identification Techniques for Automated Static Code Analysis. *Inf. Softw. Technol.* 53, 4: 363–387. DOI= http://doi.org/10.1016/j.infsof.2010.12.007

[102] H. Hemmati, S. Nadi, O. Baysal, et al. 2013. The MSR Cookbook: Mining a decade of research. *2013 10th IEEE Working Conference on Mining Software Repositories (MSR)*, 343–352. DOI= http://doi.org/10.1109/MSR.2013.6624048

[103] K. Herzig, S. Just, and A. Zeller. 2013. It's not a bug, it's a feature: How misclassification impacts bug prediction. *2013 35th International Conference on Software Engineering (ICSE)*, 392–401. DOI= http://doi.org/10.1109/ICSE.2013.6606585

[104] K. Herzig and A. Zeller. 2009. Mining the Jazz repository: Challenges and opportunities. *6th IEEE International Working Conference on Mining Software Repositories, 2009. MSR '09*, 159–162. DOI= http://doi.org/10.1109/MSR.2009.5069495

[105] Emily Hill, Zachary P. Fry, Haley Boyd, et al. 2008. AMAP: Automatically Mining Abbreviation Expansions in Programs to Enhance Software Maintenance Tools. *Proceedings of the 2008 International Working Conference on Mining Software Repositories*, ACM, 79–88. DOI= http://doi.org/10.1145/1370750.1370771

[106] A. Hindle. 2010. Software Process Recovery: Recovering Process from Artifacts. *2010 17th Working Conference on Reverse Engineering (WCRE)*, 305–308. DOI= http://doi.org/10.1109/WCRE.2010.46

[107] A. Hindle. 2012. Green mining: A methodology of relating software change to power consumption. *2012 9th IEEE Working Conference on Mining Software Repositories (MSR)*, 78–87. DOI= http://doi.org/10.1109/MSR.2012.6224303

[108] Reid Holmes and Robert J. Walker. 2008. A Newbie's Guide to Eclipse APIs. *Proceedings of the 2008 International Working Conference on Mining Software Repositories*, ACM, 149–152. DOI= http://doi.org/10.1145/1370750.1370787

[109] M.J. Howard, S. Gupta, L. Pollock, and K. Vijay-Shanker. 2013. Automatically mining software-based, semantically-similar words from comment-code mappings. *2013 10th IEEE Working Conference on Mining Software Repositories (MSR)*, 377–386. DOI= http://doi.org/10.1109/MSR.2013.6624052

[110] Changyun Huang, Yasutaka Kamei, Kazuhiro Yamashita, and Naoyasu Ubayashi. 2013. Using Alloy to Support Feature-based DSL Construction for Mining Software Repositories. *Proceedings of the 17th International Software Product Line Conference Co-located Workshops*, ACM, 86–89. DOI= http://doi.org/10.1145/2499777.2500714

[111] Changyun Huang, K. Yamashita, Y. Kamei, K. Hisazumi, and N. Ubayashi. 2013. Domain analysis for mining software repositories: Towards feature-based DSL construction. *2013 4th International Workshop on Product Line Approaches in Software Engineering (PLEASE)*, 41–44. DOI= http://doi.org/10.1109/PLEASE.2013.6608663

[112] K. Hullett, N. Nagappan, E. Schuh, and J. Hopson. 2011. Data analytics for game development: NIER track. *2011 33rd International Conference on Software Engineering (ICSE)*, 940–943. DOI= http://doi.org/10.1145/1985793.1985952

[113] F. Jaafar, Y.-G. Gueheneuc, S. Hamel, and F. Khomh. 2013. Mining the relationship between anti-patterns dependencies and fault-proneness. *2013 20th Working Conference on Reverse Engineering (WCRE)*, 351–360. DOI= http://doi.org/10.1109/WCRE.2013.6671310

[114] Fehmi Jaafar, Yann-Gaël Guéhéneuc, Sylvie Hamel, and Giuliano Antoniol. 2014. Detecting asynchrony and dephase change patterns by mining software repositories. *Journal of Software: Evolution and Process* 26, 1: 77–106. DOI= http://doi.org/10.1002/smr.1635

[115] P.M. Johnson. 2013. Searching under the Streetlight for Useful Software Analytics. *IEEE Software* 30, 4: 57–63. DOI= http://doi.org/10.1109/MS.2013.69

[116] Yungbum Jung, Hakjoo Oh, and Kwangkeun Yi. 2009. Identifying Static Analysis Techniques for Finding Non-fix Hunks in Fix Revisions. *Proceedings of the ACM First International Workshop on Data-intensive Software Management and Mining*, ACM, 13–18. DOI= http://doi.org/10.1145/1651309.1651313

[117] H. Kagdi, M. Gethers, and D. Poshyvanyk. 2011. SE2 model to support software evolution. *2011 27th IEEE International Conference on Software Maintenance (ICSM)*, 512–515. DOI= http://doi.org/10.1109/ICSM.2011.6080820

[118] H. Kagdi and J.I. Maletic. 2006. Software-Change Prediction: Estimated+Actual. *Second International IEEE Workshop on Software Evolvability, 2006. SE '06*, 38–43. DOI= http://doi.org/10.1109/SOFTWARE-EVOLVABILITY.2006.14

[119] H. Kagdi, J.I. Maletic, and B. Sharif. 2007. Mining software repositories for traceability links. *15th IEEE International Conference on Program Comprehension, 2007. ICPC '07*, 145–154. DOI= http://doi.org/10.1109/ICPC.2007.28

[120] Huzefa Kagdi, Michael L. Collard, and Jonathan I. Maletic. 2007. A survey and taxonomy of approaches for mining software repositories in the context of software evolution. *J. Softw. Maint. Evol.: Res. Pract.* 19, 2: 77–131. DOI= http://doi.org/10.1002/smr.344

[121] Huzefa Kagdi, Malcom Gethers, Denys Poshyvanyk, and Maen Hammad. 2012. Assigning change requests to software developers. *Journal of Software: Evolution and Process* 24, 1: 3–33. DOI= http://doi.org/10.1002/smr.530

[122] Y. Kamei, E. Shihab, B. Adams, et al. 2013. A large-scale empirical study of just-in-time quality assurance. *IEEE Transactions on Software Engineering* 39, 6: 757–773. DOI= http://doi.org/10.1109/TSE.2012.70

[123] D. Kawrykow and M.P. Robillard. 2011. Non-essential changes in version histories. *2011 33rd International Conference on Software Engineering (ICSE)*, 351–360. DOI= http://doi.org/10.1145/1985793.1985842

[124] I. Keivanloo. 2012. Online sharing and integration of results from mining software repositories. *2012 34th International Conference on Software Engineering (ICSE)*, 1644–1646. DOI= http://doi.org/10.1109/ICSE.2012.6227215

[125] I. Keivanloo, C. Forbes, A. Hmood, et al. 2012. A Linked Data platform for mining software repositories. *2012 9th*

IEEE Working Conference on Mining Software Repositories (MSR), 32–35. DOI= http://doi.org/10.1109/MSR.2012.6224296

[126] Iman Keivanloo and Juergen Rilling. 2014. Software trustworthiness 2.0—A semantic web enabled global source code analysis approach. *Journal of Systems and Software* 89: 33–50. DOI= http://doi.org/10.1016/j.jss.2013.08.030

[127] F. Khomh, M. Di Penta, and Y. Guéhéneuc. 2009. An Exploratory Study of the Impact of Code Smells on Software Change-proneness. *16th Working Conference on Reverse Engineering, 2009. WCRE '09*, 75–84. DOI= http://doi.org/10.1109/WCRE.2009.28

[128] Foutse Khomh, Massimiliano Di Penta, Yann-Gaël Guéhéneuc, and Giuliano Antoniol. 2012. An exploratory study of the impact of antipatterns on class change- and fault-proneness. *Empirical Software Engineering* 17, 3: 243–275. DOI= http://doi.org/10.1007/s10664-011-9171-y

[129] C. Kiefer, A. Bernstein, and J. Tappolet. 2007. Mining Software Repositories with iSPAROL and a Software Evolution Ontology. *Fourth International Workshop on Mining Software Repositories, 2007. ICSE Workshops MSR '07*, 10–10. DOI= http://doi.org/10.1109/MSR.2007.21

[130] Jungil Kim and Eunjoo Lee. 2014. The Effect of IMPORT Change in Software Change History. *Proceedings of the 29th Annual ACM Symposium on Applied Computing*, ACM, 1753–1754. DOI= http://doi.org/10.1145/2554850.2559925

[131] Sunghun Kim, S. Shivaji, and E.J. Whitehead. 2009. Kenyon-web: Reconfigurable web-based feature extractor. *IEEE 17th International Conference on Program Comprehension, 2009. ICPC '09*, 287–288. DOI= http://doi.org/10.1109/ICPC.2009.5090061

[132] Sunghun Kim, Hongyu Zhang, Rongxin Wu, and Liang Gong. 2011. Dealing with Noise in Defect Prediction. *Proceedings of the 33rd International Conference on Software Engineering*, ACM, 481–490. DOI= http://doi.org/10.1145/1985793.1985859

[133] Sunghun Kim, Thomas Zimmermann, Miryung Kim, et al. 2006. TA-RE: An Exchange Language for Mining Software Repositories. *Proceedings of the 2006 International Workshop on Mining Software Repositories*, ACM, 22–25. DOI= http://doi.org/10.1145/1137983.1137990

[134] Barbara A. Kitchenham, David Budgen, and O. Pearl Brereton. 2011. Using Mapping Studies As the Basis for Further Research - A Participant-observer Case Study. *Inf. Softw. Technol.* 53, 6: 638–651. DOI= http://doi.org/10.1016/j.infsof.2010.12.011

[135] Kitchenham, Barbara. 2007. *Guidelines for performing systematic literature reviews in software engineering*. Technical report, EBSE Technical Report EBSE-2007-01. Retrieved April 25, 2015 from https://www.cs.auckland.ac.nz/~norsaremah/2007%20Guidelines%20for%20performing%20SLR%20in%20SE%20v2.3.pdf

[136] C. Klammer and J. Pichler. 2014. Towards tool support for analyzing legacy systems in technical domains. *2014 Software Evolution Week - IEEE Conference on Software Maintenance, Reengineering and Reverse Engineering (CSMR-WCRE)*, 371–374. DOI= http://doi.org/10.1109/CSMR-WCRE.2014.6747197

[137] Oleksii Kononenko, Olga Baysal, Reid Holmes, and Michael W. Godfrey. 2014. Mining Modern Repositories with Elasticsearch. *Proceedings of the 11th Working Conference on Mining Software Repositories*, ACM, 328–331. DOI= http://doi.org/10.1145/2597073.2597091

[138] J. Krinke, N. Gold, Yue Jia, and D. Binkley. 2010. Cloning and copying between GNOME projects. *2010 7th IEEE Working Conference on Mining Software Repositories (MSR)*, 98–101. DOI= http://doi.org/10.1109/MSR.2010.5463290

[139] Raula Gaikovina Kula, Kyohei Fushida, Norihiro Yoshida, and Hajimu Iida. 2013. Micro process analysis of maintenance effort: an open source software case study using metrics based on program slicing. *Journal of Software: Evolution and Process* 25, 9: 935–955. DOI= http://doi.org/10.1002/smr.1572

[140] N. Kusunoki, K. Hotta, Y. Higo, and S. Kusumoto. 2013. How Much Do Code Repositories Include Peripheral Modifications? *Software Engineering Conference (APSEC, 2013 20th Asia-Pacific*, 19–24. DOI= http://doi.org/10.1109/APSEC.2013.106

[141] S. Lal and A. Sureka. 2012. Comparison of Seven Bug Report Types: A Case-Study of Google Chrome Browser Project. *Software Engineering Conference (APSEC), 2012 19th Asia-Pacific*, 517–526. DOI= http://doi.org/10.1109/APSEC.2012.54

[142] Alina Lazar, Sarah Ritchey, and Bonita Sharif. 2014. Generating Duplicate Bug Datasets. *Proceedings of the 11th Working Conference on Mining Software Repositories*, ACM, 392–395. DOI= http://doi.org/10.1145/2597073.2597128

[143] Bixin Li, Xiaobing Sun, Hareton Leung, and Sai Zhang. 2013. A survey of code-based change impact analysis techniques. *Software Testing, Verification and Reliability* 23, 8: 613–646. DOI= http://doi.org/10.1002/stvr.1475

[144] M. Linares-Vasquez, B. Dit, and D. Poshyvanyk. 2013. An exploratory analysis of mobile development issues using stack overflow. *2013 10th IEEE Working Conference on Mining Software Repositories (MSR)*, 93–96. DOI= http://doi.org/10.1109/MSR.2013.6624014

[145] Mario Linares-Vásquez, Collin McMillan, Denys Poshyvanyk, and Mark Grechanik. 2014. On using machine learning to automatically classify software applications into domain categories. *Empirical Software Engineering* 19, 3: 582–618. DOI= http://doi.org/10.1007/s10664-012-9230-z

[146] E. Linstead, P. Rigor, Sushil Bajracharya, C. Lopes, and P. Baldi. 2007. Mining Eclipse Developer Contributions via Author-Topic Models. *Fourth International Workshop on Mining Software Repositories, 2007. ICSE Workshops MSR '07*, 30–30. DOI= http://doi.org/10.1109/MSR.2007.20

[147] Erik Linstead, Sushil Bajracharya, Trung Ngo, Paul Rigor, Cristina Lopes, and Pierre Baldi. 2009. Sourcerer: mining and searching internet-scale software repositories. *Data Mining and Knowledge Discovery* 18, 2: 300–336. DOI= http://doi.org/10.1007/s10618-008-0118-x

[148] Jian-Guang Lou, Qingwei Lin, Rui Ding, Qiang Fu, Dongmei Zhang, and Tao Xie. 2013. Software analytics for incident management of online services: An experience report. *2013 IEEE/ACM 28th International Conference on Automated Software Engineering (ASE)*, 475–485. DOI= http://doi.org/10.1109/ASE.2013.6693105

[149] M. Lungu and M. Lanza. 2010. The small project observatory: a tool for reverse engineering software ecosystems. *2010 ACM/IEEE 32nd International*

Conference on Software Engineering, 289–292. DOI= http://doi.org/10.1145/1810295.1810356

[150] G. Maskeri, D. Karnam, S.A. Viswanathan, and S. Padmanabhuni. 2012. Version history based source code plagiarism detection in proprietary systems. *2012 28th IEEE International Conference on Software Maintenance (ICSM)*, 609–612. DOI= http://doi.org/10.1109/ICSM.2012.6405334

[151] G. Maskeri, D. Karnam, S.A. Viswanathan, and S. Padmanabhuni. 2012. Bug Prediction Metrics Based Decision Support for Preventive Software Maintenance. *Software Engineering Conference (APSEC), 2012 19th Asia-Pacific*, 260–269. DOI= http://doi.org/10.1109/APSEC.2012.43

[152] S. McIntosh. 2011. Build system maintenance. *2011 33rd International Conference on Software Engineering (ICSE)*, 1167–1169. DOI= http://doi.org/10.1145/1985793.1986031

[153] S. McIntosh, B. Adams, T.H.D. Nguyen, Y. Kamei, and A.E. Hassan. 2011. An empirical study of build maintenance effort. *2011 33rd International Conference on Software Engineering (ICSE)*, 141–150. DOI= http://doi.org/10.1145/1985793.1985813

[154] Shane McIntosh, Yasutaka Kamei, Bram Adams, and Ahmed E. Hassan. 2014. The Impact of Code Review Coverage and Code Review Participation on Software Quality: A Case Study of the Qt, VTK, and ITK Projects. *Proceedings of the 11th Working Conference on Mining Software Repositories*, ACM, 192–201. DOI= http://doi.org/10.1145/2597073.2597076

[155] Xiaozhu Meng, B.P. Miller, W.R. Williams, and A.R. Bernat. 2013. Mining Software Repositories for Accurate Authorship. *2013 29th IEEE International Conference on Software Maintenance (ICSM)*, 250–259. DOI= http://doi.org/10.1109/ICSM.2013.36

[156] T. Menzies and T. Zimmermann. 2013. Software Analytics: So What? *IEEE Software* 30, 4: 31–37. DOI= http://doi.org/10.1109/MS.2013.86

[157] R. Minelli and M. Lanza. 2013. SAMOA – A Visual Software Analytics Platform for Mobile Applications. *2013 29th IEEE International Conference on Software Maintenance (ICSM)*, 476–479. DOI= http://doi.org/10.1109/ICSM.2013.76

[158] S. Minto and G.C. Murphy. 2007. Recommending Emergent Teams. *Fourth International Workshop on Mining Software Repositories, 2007. ICSE Workshops MSR '07*, 5–5. DOI= http://doi.org/10.1109/MSR.2007.27

[159] A.T. Misirli, B. Caglayan, A. Bener, and B. Turhan. 2013. A Retrospective Study of Software Analytics Projects: In-Depth Interviews with Practitioners. *IEEE Software* 30, 5: 54–61. DOI= http://doi.org/10.1109/MS.2013.93

[160] Megha Mittal and Ashish Sureka. 2014. Process Mining Software Repositories from Student Projects in an Undergraduate Software Engineering Course. *Companion Proceedings of the 36th International Conference on Software Engineering*, ACM, 344–353. DOI= http://doi.org/10.1145/2591062.2591152

[161] O. Mizuno, S. Ikami, S. Nakaichi, and T. Kikuno. 2007. Spam Filter Based Approach for Finding Fault-Prone Software Modules. *Fourth International Workshop on Mining Software Repositories, 2007. ICSE Workshops MSR '07*, 4–4. DOI= http://doi.org/10.1109/MSR.2007.29

[162] P. Morrison and E. Murphy-Hill. 2013. Is programming knowledge related to age? An exploration of stack overflow. *2013 10th IEEE Working Conference on Mining Software Repositories (MSR)*, 69–72. DOI= http://doi.org/10.1109/MSR.2013.6624008

[163] R. Musson, J. Richards, D. Fisher, C. Bird, B. Bussone, and S. Ganguly. 2013. Leveraging the Crowd: How 48,000 Users Helped Improve Lync Performance. *IEEE Software* 30, 4: 38–45. DOI= http://doi.org/10.1109/MS.2013.67

[164] R. Nagano, H. Nakamura, Y. Kamei, et al. 2012. Using the GPGPU for scaling up Mining Software Repositories. *2012 34th International Conference on Software Engineering (ICSE)*, 1435–1436. DOI= http://doi.org/10.1109/ICSE.2012.6227077

[165] Anh Tuan Nguyen, Tung Thanh Nguyen, Hoan Anh Nguyen, and Tien N. Nguyen. 2012. Multi-layered Approach for Recovering Links Between Bug Reports and Fixes. *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*, ACM, 63:1–63:11. DOI= http://doi.org/10.1145/2393596.2393671

[166] Hoan Anh Nguyen, Anh Tuan Nguyen, and T.N. Nguyen. 2013. Filtering noise in mixed-purpose fixing commits to improve defect prediction and localization. *2013 IEEE 24th International Symposium on Software Reliability Engineering (ISSRE)*, 138–147. DOI= http://doi.org/10.1109/ISSRE.2013.6698913

[167] Tung Thanh Nguyen, T.N. Nguyen, E. Duesterwald, T. Klinger, and P. Santhanam. 2012. Inferring developer expertise through defect analysis. *2012 34th International Conference on Software Engineering (ICSE)*, 1297–1300. DOI= http://doi.org/10.1109/ICSE.2012.6227095

[168] Renato Lima Novais, André Torres, Thiago Souto Mendes, Manoel Mendonça, and Nico Zazworka. 2013. Software evolution visualization: A systematic mapping study. *Information and Software Technology* 55, 11: 1860–1883. DOI= http://doi.org/10.1016/j.infsof.2013.05.008

[169] G.A. Oliva and M.A. Gerosa. 2011. On the Interplay between Structural and Logical Dependencies in Open-Source Software. *2011 25th Brazilian Symposium on Software Engineering (SBES)*, 144–153. DOI= http://doi.org/10.1109/SBES.2011.39

[170] G.A. Oliva and M.A. Gerosa. 2012. A Method for the Identification of Logical Dependencies. *2012 IEEE Seventh International Conference on Global Software Engineering Workshops (ICGSEW)*, 70–72. DOI= http://doi.org/10.1109/ICGSEW.2012.19

[171] P. Bourque and R.E. Fairley. 2014. Guide to the Software Engineering Body of Knowledge, Version 3.0. Retrieved from www.swebok.org

[172] Rohan Padhye, Senthil Mani, and Vibha Singhal Sinha. 2014. A Study of External Community Contribution to Open-source Projects on GitHub. *Proceedings of the 11th Working Conference on Mining Software Repositories*, ACM, 332–335. DOI= http://doi.org/10.1145/2597073.2597113

[173] Jihun Park, Miryung Kim, and Doo-Hwan Bae. 2014. An Empirical Study on Reducing Omission Errors in Practice. *Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering*, ACM, 121–126. DOI= http://doi.org/10.1145/2642937.2642956

[174] Ken Peffers, Marcus Rothenberger, Tuure Tuunanen, and Reza Vaezi. 2012. Design Science Research Evaluation. In *Design Science Research in Information Systems. Advances*

in Theory and Practice, Ken Peffers, Marcus Rothenberger and Bill Kuechler (eds.). Springer Berlin Heidelberg, 398–410. Retrieved April 25, 2015 from http://link.springer.com/chapter/10.1007/978-3-642-29863-9_29

[175] Kai Petersen, Robert Feldt, Shahid Mujtaba, and Michael Mattsson. 2008. Systematic Mapping Studies in Software Engineering. *Proceedings of the 12th International Conference on Evaluation and Assessment in Software Engineering*, British Computer Society, 68–77. Retrieved April 14, 2016 from http://dl.acm.org/citation.cfm?id=2227115.2227123

[176] W. Poncin, A. Serebrenik, and M. van den Brand. 2011. Process Mining Software Repositories. *2011 15th European Conference on Software Maintenance and Reengineering (CSMR)*, 5–14. DOI= http://doi.org/10.1109/CSMR.2011.5

[177] Wouter Poncin, Alexander Serebrenik, and Mark van den Brand. 2011. Mining Student Capstone Projects with FRASR and ProM. *Proceedings of the ACM International Conference Companion on Object Oriented Programming Systems Languages and Applications Companion*, ACM, 87–96. DOI= http://doi.org/10.1145/2048147.2048181

[178] Lutz Prechelt and Alexander Pepper. 2014. Why software repositories are not used for defect-insertion circumstance analysis more often: A case study. *Information and Software Technology* 56, 10: 1377–1389. DOI= http://doi.org/10.1016/j.infsof.2014.05.001

[179] F. Rahman, C. Bird, and P. Devanbu. 2010. Clones: What is that smell? *2010 7th IEEE Working Conference on Mining Software Repositories (MSR)*, 72–81. DOI= http://doi.org/10.1109/MSR.2010.5463343

[180] A. Rastogi and A. Sureka. 2013. SamikshaUmbra: Contribution and Performance Assessment of Software Maintenance Professionals by Mining Software Repositories. *Software Engineering Conference (APSEC, 2013 20th Asia-Pacific*, 170–175. DOI= http://doi.org/10.1109/APSEC.2013.134

[181] Ayushi Rastogi and Ashish Sureka. 2014. SamikshaViz: A Panoramic View to Measure Contribution and Performance of Software Maintenance Professionals by Mining Bug Archives. *Proceedings of the 7th India Software Engineering Conference*, ACM, 2:1–2:10. DOI= http://doi.org/10.1145/2590748.2590750

[182] Sandeep Reddivari, Shirin Rad, Tanmay Bhowmik, Nisreen Cain, and Nan Niu. 2014. Visual requirements analytics: a framework and case study. *Requirements Engineering* 19, 3: 257–279. DOI= http://doi.org/10.1007/s00766-013-0194-3

[183] Meghan Revelle, Malcom Gethers, and Denys Poshyvanyk. 2011. Using structural and textual information to capture feature coupling in object-oriented software. *Empirical Software Engineering* 16, 6: 773–811. DOI= http://doi.org/10.1007/s10664-011-9159-7

[184] P.C. Rigby, D.M. German, and M. Storey. 2008. Open source software peer review practices. *ACM/IEEE 30th International Conference on Software Engineering, 2008. ICSE '08*, 541–550. DOI= http://doi.org/10.1145/1368088.1368162

[185] R. Robbes and M. Lungu. 2011. A study of ripple effects in software ecosystems: (NIER track). *2011 33rd International Conference on Software Engineering (ICSE)*, 904–907. DOI= http://doi.org/10.1145/1985793.1985940

[186] R. Robbes, R. Vidal, and M.C. Bastarrica. 2013. Are Software Analytics Efforts Worthwhile for Small Companies? The Case of Amisoft. *IEEE Software* 30, 5: 46–53. DOI= http://doi.org/10.1109/MS.2013.92

[187] Romain Robbes, David Röthlisberger, and Éric Tanter. 2014. Object-oriented software extensions in practice. *Empirical Software Engineering*: 1–38. DOI= http://doi.org/10.1007/s10664-013-9298-0

[188] Martin P. Robillard and Barthélémy Dagenais. 2010. Recommending change clusters to support software investigation: an empirical study. *Journal of Software Maintenance and Evolution: Research and Practice* 22, 3: 143–164. DOI= http://doi.org/10.1002/smr.413

[189] G. Robles. 2010. Replicating MSR: A study of the potential replicability of papers published in the Mining Software Repositories proceedings. *2010 7th IEEE Working Conference on Mining Software Repositories (MSR)*, 171–180. DOI= http://doi.org/10.1109/MSR.2010.5463348

[190] Gregorio Robles and Jesus M. Gonzalez-Barahona. 2005. Developer Identification Methods for Integrated Data from Various Sources. *Proceedings of the 2005 International Workshop on Mining Software Repositories*, ACM, 1–5. DOI= http://doi.org/10.1145/1082983.1083162

[191] Gregorio Robles, Jesús M. González-Barahona, Carlos Cervigón, Andrea Capiluppi, and Daniel Izquierdo-Cortázar. 2014. Estimating Development Effort in Free/Open Source Software Projects by Mining Software Repositories: A Case Study of OpenStack. *Proceedings of the 11th Working Conference on Mining Software Repositories*, ACM, 222–231. DOI= http://doi.org/10.1145/2597073.2597107

[192] Gregorio Robles, Jesus M. Gonzalez-Barahona, and Juan Julian Merelo. 2006. Beyond source code: The importance of other artifacts in software development (a case study). *Journal of Systems and Software* 79, 9: 1233–1248. DOI= http://doi.org/10.1016/j.jss.2006.02.048

[193] Gregorio Robles, Jesus M. Gonzalez-Barahona, Martin Michlmayr, and Juan Jose Amor. 2006. Mining Large Software Compilations over Time: Another Perspective of Software Evolution. *Proceedings of the 2006 International Workshop on Mining Software Repositories*, ACM, 3–9. DOI= http://doi.org/10.1145/1137983.1137986

[194] Tobias Sager, Abraham Bernstein, Martin Pinzger, and Christoph Kiefer. 2006. Detecting Similar Java Classes Using Tree Algorithms. *Proceedings of the 2006 International Workshop on Mining Software Repositories*, ACM, 65–71. DOI= http://doi.org/10.1145/1137983.1138000

[195] R.W. Selby. 2005. Enabling reuse-based software development of large-scale systems. *IEEE Transactions on Software Engineering* 31, 6: 495–510. DOI= http://doi.org/10.1109/TSE.2005.69

[196] Weiyi Shang, Bram Adams, and Ahmed E. Hassan. 2010. An Experience Report on Scaling Tools for Mining Software Repositories Using MapReduce. *Proceedings of the IEEE/ACM International Conference on Automated Software Engineering*, ACM, 275–284. DOI= http://doi.org/10.1145/1858996.1859050

[197] Weiyi Shang, Bram Adams, and Ahmed E. Hassan. 2012. Using Pig as a data preparation language for large-scale mining software repositories studies: An experience report.

Journal of Systems and Software 85, 10: 2195–2204. DOI= http://doi.org/10.1016/j.jss.2011.07.034

[198] Weiyi Shang, Zhen Ming Jiang, B. Adams, and A.E. Hassan. 2009. MapReduce as a general framework to support research in Mining Software Repositories (MSR). *6th IEEE International Working Conference on Mining Software Repositories, 2009. MSR '09*, 21–30. DOI= http://doi.org/10.1109/MSR.2009.5069477

[199] Weiyi Shang, Meiyappan Nagappan, and Ahmed E. Hassan. 2013. Studying the relationship between logging characteristics and the code quality of platform software. *Empirical Software Engineering*: 1–27. DOI= http://doi.org/10.1007/s10664-013-9274-8

[200] Mary Shaw. 2002. What makes good research in software engineering? *International Journal on Software Tools for Technology Transfer* 4, 1: 1–7. DOI= http://doi.org/10.1007/s10009-002-0083-4

[201] Mary Shaw. 2003. Writing Good Software Engineering Research Papers: Minitutorial. *Proceedings of the 25th International Conference on Software Engineering*, IEEE Computer Society, 726–736. Retrieved April 25, 2015 from http://dl.acm.org/citation.cfm?id=776816.776925

[202] E. Shihab, Zhen Ming Jiang, and A.E. Hassan. 2009. On the use of Internet Relay Chat (IRC) meetings by developers of the GNOME GTK+ project. *6th IEEE International Working Conference on Mining Software Repositories, 2009. MSR '09*, 107–110. DOI= http://doi.org/10.1109/MSR.2009.5069488

[203] Jacek Śliwerski, Thomas Zimmermann, and Andreas Zeller. 2005. When Do Changes Induce Fixes? *Proceedings of the 2005 International Workshop on Mining Software Repositories*, ACM, 1–5. DOI= http://doi.org/10.1145/1082983.1083147

[204] F.Z. Sokol, M. Finavaro Aniche, and M.A. Gerosa. 2013. MetricMiner: Supporting researchers in mining software repositories. *2013 IEEE 13th International Working Conference on Source Code Analysis and Manipulation (SCAM)*, 142–146. DOI= http://doi.org/10.1109/SCAM.2013.6648195

[205] R. Souza and C. Chavez. 2012. Characterizing verification of bug fixes in two open source IDEs. *2012 9th IEEE Working Conference on Mining Software Repositories (MSR)*, 70–73. DOI= http://doi.org/10.1109/MSR.2012.6224301

[206] R. Souza, C. Chavez, and R. Bittencourt. 2013. Patterns for extracting high level information from bug reports. *2013 1st International Workshop on Data Analysis Patterns in Software Engineering (DAPSE)*, 29–31. DOI= http://doi.org/10.1109/DAPSE.2013.6603807

[207] R. Souza, C. Chavez, and R. Bittencourt. 2013. Patterns for cleaning up bug data. *2013 1st International Workshop on Data Analysis Patterns in Software Engineering (DAPSE)*, 26–28. DOI= http://doi.org/10.1109/DAPSE.2013.6603806

[208] Rodrigo Souza, Christina Chavez, and Roberto A. Bittencourt. 2014. Do Rapid Releases Affect Bug Reopening? A Case Study of Firefox. *2014 Brazilian Symposium on Software Engineering (SBES)*, 31–40. DOI= http://doi.org/10.1109/SBES.2014.10

[209] H. Sözer. 2014. Integrated static code analysis and runtime verification. *Software: Practice and Experience*: n/a-n/a. DOI= http://doi.org/10.1002/spe.2287

[210] Pieter van der Spek and Steven Klusener. 2011. Applying a dynamic threshold to improve cluster detection of LSI. *Science of Computer Programming* 76, 12: 1261–1274. DOI= http://doi.org/10.1016/j.scico.2010.12.004

[211] Diomidis Spinellis. 2015. A Repository with 44 Years of Unix Evolution. *Proceedings of the 12th Working Conference on Mining Software Repositories*, IEEE Press, 462–465. Retrieved May 26, 2016 from http://dl.acm.org/citation.cfm?id=2820518.2820588

[212] M. Staron, W. Meding, C. Hoglund, P. Eriksson, J. Nilsson, and J. Hansson. 2013. Identifying Implicit Architectural Dependencies Using Measures of Source Code Change Waves. *2013 39th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA)*, 325–332. DOI= http://doi.org/10.1109/SEAA.2013.9

[213] Reinout Stevens, Coen De Roover, Carlos Noguera, Andy Kellens, and Viviane Jonckers. 2014. A logic foundation for a general-purpose history querying tool. *Science of Computer Programming* 96, Part 1: 107–120. DOI= http://doi.org/10.1016/j.scico.2014.02.014

[214] A. Sureka, S. Lal, and L. Agarwal. 2011. Applying Fellegi-Sunter (FS) Model for Traceability Link Recovery between Bug Databases and Version Archives. *Software Engineering Conference (APSEC), 2011 18th Asia Pacific*, 146–153. DOI= http://doi.org/10.1109/APSEC.2011.12

[215] Mark D. Syer, Meiyappan Nagappan, Bram Adams, and Ahmed E. Hassan. 2014. Studying the relationship between source code quality and mobile platform dependence. *Software Quality Journal*: 1–24. DOI= http://doi.org/10.1007/s11219-014-9238-2

[216] T. Taipale, M. Qvist, and B. Turhan. 2013. Constructing Defect Predictors and Communicating the Outcomes to Practitioners. *2013 ACM / IEEE International Symposium on Empirical Software Engineering and Measurement*, 357–362. DOI= http://doi.org/10.1109/ESEM.2013.45

[217] Jonas Tappolet, Christoph Kiefer, and Abraham Bernstein. 2010. Semantic web enabled software analysis. *Web Semantics: Science, Services and Agents on the World Wide Web* 8, 2–3: 225–240. DOI= http://doi.org/10.1016/j.websem.2010.04.009

[218] Stephen W. Thomas. 2011. Mining Software Repositories Using Topic Models. *Proceedings of the 33rd International Conference on Software Engineering*, ACM, 1138–1139. DOI= http://doi.org/10.1145/1985793.1986020

[219] Stephen W. Thomas, Bram Adams, Ahmed E. Hassan, and Dorothea Blostein. 2014. Studying software evolution using topic models. *Science of Computer Programming* 80, Part B: 457–479. DOI= http://doi.org/10.1016/j.scico.2012.08.003

[220] Suresh Thummalapenta, Luigi Cerulo, Lerina Aversano, and Massimiliano Di Penta. 2010. An empirical study on the maintenance of source code clones. *Empirical Software Engineering* 15, 1: 1–34. DOI= http://doi.org/10.1007/s10664-009-9108-x

[221] Suresh Thummalapenta and Tao Xie. 2011. Alattin: mining alternative patterns for defect detection. *Automated Software Engineering* 18, 3–4: 293–323. DOI= http://doi.org/10.1007/s10515-011-0086-z

[222] G. Uddin, B. Dagenais, and M.P. Robillard. 2011. Analyzing temporal API usage patterns. *2011 26th IEEE/ACM International Conference on Automated*

Software Engineering (ASE), 456–459. DOI= http://doi.org/10.1109/ASE.2011.6100098

[223] G. Uddin, B. Dagenais, and M.P. Robillard. 2012. Temporal analysis of API usage concepts. *2012 34th International Conference on Software Engineering (ICSE)*, 804–814. DOI= http://doi.org/10.1109/ICSE.2012.6227138

[224] Olivier Vandecruys, David Martens, Bart Baesens, Christophe Mues, Manu De Backer, and Raf Haesen. 2008. Mining software repositories for comprehensible software fault prediction models. *Journal of Systems and Software* 81, 5: 823–839. DOI= http://doi.org/10.1016/j.jss.2007.07.034

[225] Adam Vanya, Steven Klusener, Rahul Premraj, and Hans van Vliet. 2013. Supporting software architects to improve their software system's decomposition – lessons learned. *Journal of Software: Evolution and Process* 25, 3: 219–232. DOI= http://doi.org/10.1002/smr.574

[226] Adam Vanya, Steven Klusener, Nico van Rooijen, and Hans van Vliet. 2013. Multidimensional characterization of evolutionary clusters: An experience report. *Information and Software Technology* 55, 9: 1625–1639. DOI= http://doi.org/10.1016/j.infsof.2013.02.016

[227] B. Vasilescu, A. Serebrenik, and T. Mens. 2013. A historical dataset of software engineering conferences. *2013 10th IEEE Working Conference on Mining Software Repositories (MSR)*, 373–376. DOI= http://doi.org/10.1109/MSR.2013.6624051

[228] Bogdan Vasilescu, Alexander Serebrenik, Mathieu Goeminne, and Tom Mens. 2014. On the variation and specialisation of workload—A case study of the Gnome ecosystem community. *Empirical Software Engineering* 19, 4: 955–1008. DOI= http://doi.org/10.1007/s10664-013-9244-1

[229] Lucian Voinea and Alexandru Telea. 2006. Mining Software Repositories with CVSgrab. *Proceedings of the 2006 International Workshop on Mining Software Repositories*, ACM, 167–168. DOI= http://doi.org/10.1145/1137983.1138024

[230] Lucian Voinea and Alexandru Telea. 2009. Visual querying and analysis of large software repositories. *Empirical Software Engineering* 14, 3: 316–340. DOI= http://doi.org/10.1007/s10664-008-9068-6

[231] Jinshui Wang, Xin Peng, Zhenchang Xing, and Wenyun Zhao. 2013. How developers perform feature location tasks: a human-centric and process-oriented exploratory study. *Journal of Software: Evolution and Process* 25, 11: 1193–1224. DOI= http://doi.org/10.1002/smr.1593

[232] Jue Wang, Yingnong Dang, HongYu Zhang, Kai Chen, Tao Xie, and Dongmei Zhang. 2013. Mining succinct and high-coverage API usage patterns from source code. *2013 10th IEEE Working Conference on Mining Software Repositories (MSR)*, 319–328. DOI= http://doi.org/10.1109/MSR.2013.6624045

[233] Claes Wohlin, Per Runeson, Martin Höst, Magnus C. Ohlsson, Björn Regnell, and Anders Wesslén. 2012. *Experimentation in Software Engineering*. Springer Berlin Heidelberg, Berlin, Heidelberg. Retrieved April 25, 2015 from http://link.springer.com/10.1007/978-3-642-29044-2

[234] Michael Würsch, Giacomo Ghezzi, Matthias Hert, Gerald Reif, and Harald C. Gall. 2012. SEON: a pyramid of ontologies for software evolution and its applications. *Computing* 94, 11: 857–885. DOI= http://doi.org/10.1007/s00607-012-0204-1

[235] Michael Würsch, Gerald Reif, Serge Demeyer, and Harald C. Gall. 2010. Fostering Synergies: How Semantic Web Technology Could Influence Software Repositories. *Proceedings of 2010 ICSE Workshop on Search-driven Development: Users, Infrastructure, Tools and Evaluation*, ACM, 45–48. DOI= http://doi.org/10.1145/1809175.1809187

[236] Xinrong Xie, D. Poshyvanyk, and A. Marcus. 2006. Visualization of CVS Repository Information. *13th Working Conference on Reverse Engineering, 2006. WCRE '06*, 231–242. DOI= http://doi.org/10.1109/WCRE.2006.55

[237] Andy Zaidman, Bart Van Rompaey, Arie van Deursen, and Serge Demeyer. 2011. Studying the co-evolution of production and test code in open source and industrial developer test processes through repository mining. *Empirical Software Engineering* 16, 3: 325–364. DOI= http://doi.org/10.1007/s10664-010-9143-7

[238] Sima Zamani, Sai Peck Lee, Ramin Shokripour, and John Anvik. 2014. A noun-based approach to feature location using time-aware term-weighting. *Information and Software Technology* 56, 8: 991–1011. DOI= http://doi.org/10.1016/j.infsof.2014.03.007

[239] M.S. Zanetti. 2012. The co-evolution of socio-technical structures in sustainable software development: Lessons from the open source software communities. *2012 34th International Conference on Software Engineering (ICSE)*, 1587–1590. DOI= http://doi.org/10.1109/ICSE.2012.6227030

[240] Dongmei Zhang. 2012. Software Analytics in Practice: Approaches and Experiences. *Proceedings of the 9th IEEE Working Conference on Mining Software Repositories*, IEEE Press, 1–1. Retrieved November 15, 2014 from http://dl.acm.org/citation.cfm?id=2664446.2664447

[241] Dongmei Zhang, Yingnong Dang, Jian-Guang Lou, Shi Han, Haidong Zhang, and Tao Xie. 2011. Software Analytics As a Learning Case in Practice: Approaches and Experiences. *Proceedings of the International Workshop on Machine Learning Technologies in Software Engineering*, ACM, 55–58. DOI= http://doi.org/10.1145/2070821.2070829

[242] Dongmei Zhang, Shi Han, Yingnong Dang, Jian-Guang Lou, Haidong Zhang, and Tao Xie. 2013. Software Analytics in Practice. *IEEE Software* 30, 5: 30–37. DOI= http://doi.org/10.1109/MS.2013.94

[243] Feng Zhang, A. Mockus, Ying Zou, F. Khomh, and A.E. Hassan. 2013. How Does Context Affect the Distribution of Software Maintainability Metrics? *2013 29th IEEE International Conference on Software Maintenance (ICSM)*, 350–359. DOI= http://doi.org/10.1109/ICSM.2013.46

[244] Feng Zhang, Audris Mockus, Iman Keivanloo, and Ying Zou. 2014. Towards Building a Universal Defect Prediction Model. *Proceedings of the 11th Working Conference on Mining Software Repositories*, ACM, 182–191. DOI= http://doi.org/10.1145/2597073.2597078

[245] Hongyu Zhang. 2008. Exploring Regularity in Source Code: Software Science and Zipf's Law. *15th Working Conference on Reverse Engineering, 2008. WCRE '08*, 101–110. DOI= http://doi.org/10.1109/WCRE.2008.37

[246] Y. Zhang and D. Sheth. 2006. Mining software repositories for model-driven development. *IEEE Software* 23, 1: 82–90. DOI= http://doi.org/10.1109/MS.2006.23

[247] Liming Zhao and Jane Huffman Hayes. 2011. Rank-based refactoring decision support: two studies. *Innovations in Systems and Software Engineering* 7, 3: 171–189. DOI= http://doi.org/10.1007/s11334-011-0154-3

[248] T. Zimmermann, A. Zeller, P. Weissgerber, and S. Diehl. 2005. Mining version histories to guide software changes. *IEEE Transactions on Software Engineering* 31, 6: 429–445. DOI= http://doi.org/10.1109/TSE.2005.72

[249] Thomas Zimmermann and Peter Wei\s sgerber. 2004. Preprocessing CVS data for fine-grained analysis. *Proceedings of the First International Workshop on Mining Software Repositories*, sn, 2–6. Retrieved May 26, 2016 from http://www.st.uni-trier.de/refactoring/papers/MSR04-PreprocessingVersionArchives.pdf

3 CAPÍTULO 2

Proposta de um Arcabouço Conceitual de Inteligência Analítica para a Engenharia de Software

RESUMO

A inteligência analítica na Engenharia de Software permite analisar os dados contidos nos diversos repositórios, com a finalidade de promover decisões bem embasadas aos engenheiros de software. Com a intenção de promover a utilização da inteligência analítica na Engenharia de Software este artigo responde à questão: Quais são os elementos tecnológicos de um arcabouço conceitual que permitem aos engenheiros de software obterem informações sobre o projeto de software? O arcabouço apresentado neste trabalho conta com quatro etapas: Extração; Tipos de análise; Visualização dos dados e Necessidades dos engenheiros de software. Com base nos conceitos elencados pelo arcabouço, foi construído um protótipo que atende à necessidade de prática de desenvolvimento dos engenheiros de software. Este protótipo analisou os dados do repositório de código fonte de um sistema de software livre, o Jenkins, que fornece uma visão das alterações realizadas no sistema. O protótipo foi avaliado por um grupo focal formado por profissionais da área de desenvolvimento de sistemas. Apura-se que as técnicas e as tecnologias apresentadas neste trabalho permite identificar quais artefatos do sistema estão sendo alterados e por que, facilitando o mapeamento dos arquivos do sistema por assunto, o planejamento de refatorações, os builds e o entendimento da estabilidade e evolução do software.

Categorias:

- **Sistemas de informação → Aplicações de sistemas de informação → Sistemas de apoio à decisão → Análise de dados**
- **Software e suas engenharias -> Notações e ferramentas de software -> Ferramentas de manutenção de software**

Palavras-Chaves

Inteligência Analítica; Engenharia de Software; Mineração de Repositórios de Software.

3.1 INTRODUÇÃO

A prática de desenvolvimento de software produz diversos artefatos, como: documentações de requisitos, teste de código, monitoramento de defeitos e o próprio código-fonte. Esses artefatos contêm informações sobre o projeto de software e são úteis durante todas as fases do desenvolvimento de software [9]. Apesar de úteis, esses dados, não são simples de serem interpretados. Diante desse problema, verifica-se a dificuldade para analisar esses dados e utilizá-los para embasar decisões em projetos de software.

A inteligência analítica é descrita como uma técnica que faz uso de análises, dados e raciocínio sistemático para tomar decisões [14]. Ela demonstrou ser uma técnica valiosa para a análise de processos de negócios [49]. Na Engenharia de Software, seu uso tem crescido de forma substancial na identificação de padrões e no apoio à tomada de decisão em todas as fases do desenvolvimento de software [9, 32]. Contudo, é comum que decisões tomadas no desenvolvimento de software sejam baseadas somente na intuição, sem a utilização de ferramentas ou indicadores para analisar o projeto [31].

Com a intenção de auxiliar os engenheiros de software a explorarem os dados contidos em seus repositórios, este trabalho responde à seguinte questão: Quais são os elementos tecnológicos de um arcabouço conceitual que permitem aos engenheiros de software obterem informações sobre o projeto de software? Para responder a essa questão, este trabalho apresenta um arcabouço conceitual de inteligência analítica aplicada à Engenharia de Software e o implementa, desenvolvendo um protótipo. Para avaliar o arcabouço, foi realizado um grupo focal com desenvolvedores e gestores de projeto de uma grande empresa do ramo da Tecnologia da Informação.

Este artigo está estruturado com base nos seguintes tópicos: Seção 3.2 Referencial teórico; a Seção 3.3 Arcabouço conceitual; a Seção 3.4 Implementação do arcabouço; a Seção 3.5 Configuração do grupo focal; a Seção 3.6 Resultados do grupo focal; Seção 3.7 Trabalhos relacionados e a Seção 3.8 Conclusão e trabalhos futuros.

3.2 REFERENCIAL TEÓRICO

3.2.1 Inteligência Analítica

A inteligência analítica permite aos engenheiros de software realizarem a exploração e análise de dados, a fim de obterem informações detalhadas e baseadas em dados do próprio software para a tomada de decisão sobre a condução de seu projeto e seus serviços [67]. Pesquisas têm relatado o uso da inteligência analítica para entender os sistemas de software, a propagação de mudanças, a predição e identificação de defeitos, a dinâmica de equipes de desenvolvimento, a melhorada da experiência do usuário, a reusabilidade de código e a automação de técnicas de mineração de repositórios [30]. Ela permite melhorar a qualidade do software [68] e capacitar os indivíduos e equipes de desenvolvimento [47], entre outras atividades.

Uma das funções da inteligência analítica na Engenharia de Software é permitir recomendações práticas para melhorar o projeto [47]. Ou seja, uma ferramenta de inteligência analítica pode recomendar aos programadores que realizem uma ação baseada no histórico de versões; por exemplo, “programadores que alteraram esta função também alteraram...” [71]. Dessa maneira, tem-se a intenção de reduzir erros por alterações incompletas no código e problemas de acoplamento.

Projetos de software tendem a continuar crescendo em tamanho e complexidade [13], tornando maior o esforço para analisar e interpretar os dados disponíveis em repositórios de software. A dificuldade de interpretar dados tem sido solucionada com a extração das métricas de software e a utilização de técnicas estatísticas [70].

Pesquisadores no campo da Engenharia de Software têm desenvolvido abordagens para extrair informações pertinentes e descobrir relações e tendências de repositórios no contexto da evolução de software [34]. A inteligência analítica na Engenharia de Software torna possível a utilização de indicadores para a tomada de decisão. Sem esses indicadores, desenvolvedores e gestores contam apenas com a intuição e a experiência para embasar suas decisões. Ferramentas online e ou integradas ao ambiente de desenvolvimento de software estão sendo utilizadas para essa finalidade [6]. Ou seja, oportunidades estão sendo criadas para serem exploradas nessa área.

3.3 O ARCABOUÇO CONCEITUAL

O arcabouço conceitual visa elencar os elementos tecnológicos necessários para utilização da inteligência analítica na engenharia de software. Esta Seção expõe esses elementos que foram identificados na literatura. Para facilitar sua utilização, o arcabouço foi dividido em quatro etapas: Extração, Tipos de Análises, Visualização dos Dados e as Necessidades dos Engenheiros de Software. Essas etapas e seus elementos estão representados na Figura 3.1. As etapas estão alinhadas em sequencia e permitem aos engenheiros de softwares de software maior facilidade durante a implementação dessa solução em seus projetos. Beneficiando-os com informações úteis sobre o projeto de software durante a sua condução.

3.3.1 Extração dos Dados

A extração dos dados do repositório de software é o primeiro passo para a utilização da inteligência analítica na Engenharia de Software. Visto que diversos repositórios podem ser utilizados como base para extrair informações, no Quadro 3.1 exibe os conceitos dos repositórios utilizados neste trabalho.

Quadro 3.1 Conceitos dos Repositórios de Software [30]

Repositórios	Descrição
Repositórios de Código Fonte	Rastreiam todas as alterações no código fonte junto com meta-dados sobre cada mudança. Exemplos: CVS, subversion, Perforce, ClearCas.
Repositórios de Defeitos	Rastreia o histórico de resolução dos relatos de defeitos ou pedidos de funcionalidades. Exemplos: Bugzilla e Jira.
Comunicações Arquivadas	Rastreia discussões sobre vários aspectos de um projeto de software em toda a seu tempo de vida. Exemplos: listas de discussão, e-mails e bate-papos.
Logs de Desenvolvimento	Registra informações sobre a execução de uma única implantação de um aplicativo de software ou diferentes implementações das mesmas aplicações. Exemplo: mensagem de erros da aplicação.
Repositórios de Código	Arquiva o código fonte de um grande número de projetos. Exemplos: Sourceforge.net e código do Google.

Diversas tecnologias têm sido propostas para extrair dados de repositórios de software, permitindo a automatização dessa tarefa. O C-Rex rastreia alterações para entidades de código fonte específicas, tais como, funções, variáveis ou definição de tipo de dados [29]. O CVSAnalY extrai dados de repositórios e os armazena em um banco de dados relacional, como o MySQL [52].

Para melhorar a performance no processo de extração dos dados recomendada-se a utilização de uma plataforma de computação distribuída, como o MapReduce [58]. O J-Rex, é uma ferramenta inspirada no C-Rex, para a linguagem Java, sendo executado sobre a plataforma Hadoop, uma implementação de código aberto do MapReduce. Assim como o Hadoop, a plataforma Pig tem se mostrado uma boa opção para trabalhar com dados em larga escala [57].

O Elasticsearch é um motor de pesquisa de texto escrito em Java e disponibilizado como software livre. Foi projetado para ser distribuído, escalável e com um tempo de execução quase instantânea. Podendo este ser utilizado para recuperar dados em diversos repositórios [38].

É comum a utilização de um banco de dados relacional que centraliza todas as informações para armazenar os dados extraídos dos repositórios de software. Contudo, a Web Semântica pode ser utilizada para resolver alguns problemas na mineração de repositórios de software como: formato de definição dos dados, armazenamento em diferentes localizações, rastreabilidade dos resultados da pesquisa e consultas complexas com muitos relacionamentos e a baixo custo de processamento [65]. A Web Semântica fornece uma estrutura padronizada e bem estabelecida, permitindo que dados sejam compartilhados e reutilizados entre as aplicações empresariais ou abertas [65] [61] [36]. Permite, ainda, reduzir muitas tarefas da mineração de repositórios de software para simples consultas na linguagem de consulta SPARQL [61]. O processo de bases de conhecimento distribuídas simplifica significativamente as análises entre muitos projetos de diferentes softwares [61] [36].

Por meio das ferramentas de extração de dados de repositórios, profissionais e pesquisadores podem realizar diversas análises sem terem de que dispensar tanto tempo no processo de extração dos dados, podendo elevar seus esforços para a análise dos dados.

Durante a extração dos dados as métricas podem ser obtidas. As métricas de software permitem que sejam construídos modelos que apoiem decisões sobre diferentes aspectos na Engenharia de Software [19]. As métricas extraídas de repositórios de software podem ser utilizadas para analisar o impacto de mudanças [42], premeditar defeitos [46], estimar esforço [52] e prover a reusabilidade [56], entre outras atividades da Engenharia de Software.

A análise de código com as métricas CK (Chidamber and Kemerer), métricas de complexidade, tamanho de linha de código e métricas de entropia, entre outras, fornecem informações do projeto aos gerentes de software [46]. As métricas *Number of Commits* (NoC) e *Commodification factor* (CF) permitem analisar a estrutura da comunicação no desenvolvimento de software distribuído geograficamente [28].

As métricas *Structural Feature Coupling* (SFC), *Textual Feature Coupling* (TFC), *Hybrid Feature Coupling Metric* (HFC), *Precision* e *Recall* medem o acoplamento, por meio das características de sistemas, de forma estruturada e não estruturada [51] [71].

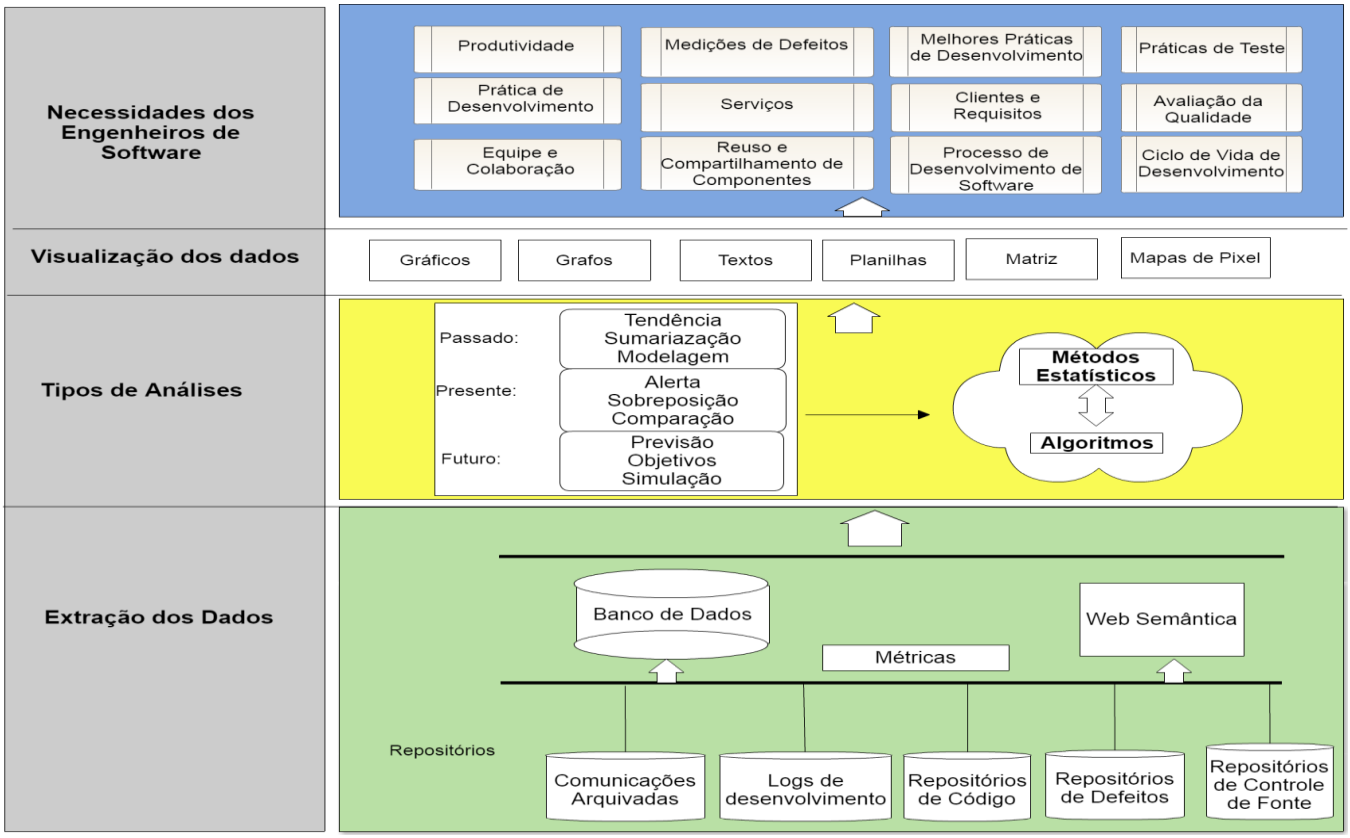


Figura 0.1 Arcabouço Conceitual de Inteligência Analítica Aplicada à Engenharia de Software

3.3.2 Tipos de Análise da Inteligência Analítica

Após a etapa de Extração dos Dados, diversas análises podem ser utilizadas para obter informações do projeto. Nesta Seção, apresentam-se os tipos de análise fornecidos pelas ferramentas de inteligência analítica aos engenheiros de software, com base no trabalho de Buse e Zimmermann [14].

Buse e Zimmermann [14] realizaram um *survey* com desenvolvedores e gerentes da indústria de software para descobrir como a inteligência analítica pode contribuir para embasar decisões no desenvolvimento de sistemas. Eles categorizaram os tipos de análises da inteligência analítica descritos conforme o Quadro 3.2.

Os tipos de analises são caracterizados de acordo com tempo (passado, presente e futuro) e a técnica a aplicada (exploração, análise e experimentação). Com essa categorização é possível responder questões como: O que está acontecendo? Como e por que isso aconteceu? O que está acontecendo agora? Qual a melhor ação? O que acontecerá? O que de melhor ou pior pode acontecer? [14]

Quadro 3.2 Tipos de Análises da Inteligência Analítica [14]

	Passado	**Presente**	**Futuro**
Exploração	**Tendência** Quantifica como os artefatos estão mudando.	**Alerta** Informa mudanças incomuns nos artefatos quando elas acontecem.	**Previsão** Prevê eventos baseados em tendências.
Análise	**Sumarização** Características sucintas de artefatos-chaves ou grupos de artefatos.	**Sobreposição** Compara artefatos ou históricos de desenvolvimen-to de forma interativa.	**Objetivos** Descobre como os artefatos estão mudando com os respectivos objetivos.
Experimentação	**Modelagem** Caracteriza o comportamento normal do desenvolvimen-to.	**Comparação** Compara artefatos para obter as melhores práticas.	**Simulação** Testa condições antes de acontece-rem.

3.3.3 Métodos Estatísticos

Dentre as análises dos dados, alguns métodos estatísticos podem ser utilizados. Listam-se, a seguir, alguns dos métodos estatísticos utilizados pela inteligência analítica aplicados na Engenharia de Software.

- **A Causalidade de Granger -** é útil para analisar o impacto de mudanças, detectando alterações nos artefatos de software [16]. Por meio dos métodos estatísticos: teste exato de Fisher, teste de Mann-Whitney e modelo de regressão logística, verifica-se a propensão às mudanças das classes [35].
- **A Lei de Zipf -** tem sido utilizada para encontrar a regularidade subjacente em código fonte. A lei de Zipf prescreve que a frequência de um símbolo é inversamente proporcional à sua classificação em uma hipotética (ordenadas) tabela de frequência [5]. Por meio desse método estatístico, faz-se a análise léxica em código fonte. Zhang [69] acredita que por meio da Lei de Zipf é possível trabalhar para a recuperação da informação de software, otimização de código, compreensão de software e medida de complexidade de software.
- **Modelos de tópicos** – trata-se de um modelo probabilístico que procura encontrar palavras relevantes em um texto. Um tópico funciona como uma distribuição probabilística sobre as palavras [60], ou seja, uma coleção de palavras de co-ocorrência [62]. O LDA (*Latent Dirichlet Allocation*) é um modelo de tópicos probabilístico. A ideia básica é a de que os documentos são representados de forma aleatória ao longo dos tópicos, em que cada tópico é caracterizado por uma distribuição de palavras. Os modelos de tópicos têm se mostrado bastante útil em pesquisas de inteligência analítica na engenharia de software [4] [8] [21]. Tem sido utilizado para automatizar a estrutura de dados textuais encontrados em repositórios de software, analisar a evolução de software [63], descobrir tópicos sobre desenvolvimento em sites de discussão de perguntas e respostas técnicas sobre desenvolvimento de software, como o *StackOverflow* [44] [8] e promover um método para classificar automaticamente as alterações do software usando apenas as descrições textuais de mensagens de mudança em sua manutenção [21].
- **A correlação de Kendall -** é utilizada para correlacionar processo de qualidade de dados e o número de defeitos em sistemas [7].

Os métodos estatísticos, em geral, são utilizados juntamente com as métricas extraídas em repositórios de software, podendo ser utilizado para correlacionar, apresentar previsão ou encontrar padrões.

3.3.4 Algoritmos

Alguns algoritmos são frequentemente utilizados na etapa de análises dos dados dos repositórios de software. O uso de algoritmos na inteligência analítica permite automatizar e descobrir padrões para diversas atividades da Engenharia de Software. Por exemplo, para analisar impacto em projetos de software por meio de algoritmos de regras de associação [17]. As regras de associação permitem descobrir padrões, exibindo os valores que ocorrem juntos em determinado conjunto de dados [16] [1]. Assim, é possível verificar os artefatos que estão sendo alterados juntos [33]. O algoritmo de regra de associação *Apriori*, descobre padrões em grandes coleções de itens [2].

Algoritmos de aprendizado de máquina, como, SVM*, Naïve Bayes, Decision Tree, JRIP* e *IBK*, permitem categorizar sistemas de software [44] e classificar, por exemplo, os tipos de defeitos relatados em um sistema [41]. As ferramentas *Colt* e *Weka* podem compor a infraestrutura para execução dessas análises [45]. O algoritmo *regular language reachability* auxilia os engenheiros de software a coordenar equipes, partindo do princípio de grafos, em que todos os elementos estão relacionados (pessoas, código, teste, defeitos etc.) [10]. A rede funcional é uma generalização do padrão da rede neural. Este paradigma de inteligência computacional tem se mostrado eficiente para estimar o esforço de desenvolvimento [18].

Novos algoritmos vêm sendo propostos para minerar repositórios de software. O Alattin tem a proposta de ser utilizado para melhorar a reusabilidade em sistemas de software. O Atlattin analisa e detecta condições negligenciadas sobre APIs reutilizadas pelo aplicativo, detectando violações de regras de programação [64]. O *Closest List Noise Identification* (CLNI) foi criado para diminuir os ruídos na predição de defeitos de software [37].

3.3.5 Visualização

A etapa de Visualização dos Dados permite aos engenheiros de software obterem informações relevantes sobre o projeto de software de modo visual. Pode ser efetivamente usado para analisar e entender grande quantidade de dados produzidos durante a evolução do software [48].

Dependendo do objetivo do engenheiro de software em investigar o histórico de versões do sistema, é possível fornecer diferentes visões. Nota-se que as visualizações das alterações no sistema de software podem ser representadas por meio de gráficos de barra, mapas de pixel, grafos, texto, matriz e planilhas. Combinam cores e interações com o usuário, por meio de rotação, escala, *zoom*, filtragens etc [66], além da utilização de elementos animados [23].

Rastogi e Sureka [50] propuseram seis técnicas de visualização para auxiliar desenvolvedores durante a manutenção de software:

Trellis Plot - utilizado para descobrir relações de variáveis em um conjunto de dados multivariadas.

Treemap Plot- utiliza subpartições (retângulos) que possuem duas características, tamanho e cor dos retângulos. Ambas representam as propriedades específicas da informação.

Bertin's Hotel - estrutura homogênea que usa rearranjo de linhas e colunas para revelar informações de interesse.

Dart Chart - apresenta a informação em forma de círculo contendo setores.

Hybrid Plot - combina gráficos simples, para exibir informações de uma forma rica e complexa.

Nightingale Rose Plot (coxcomb) - gráfico radial que permite analisar a diferença entre o valor real e seu valor esperado correspondente.

As técnicas de visualização de inteligência analítica combinam as vantagens de máquinas com os pontos fortes dos seres humanos, tais como: análise, intuição, resolução de problemas e percepção visual [26].

As pesquisas sobre a visualização de software na inteligência analítica têm apresentado técnicas e formas de exibir as informações já processadas aos engenheiros de software. Por se tratar do elemento em que interage diretamente com o usuário final, são requeridos testes de usabilidade, para garantir as melhores exibições das informações extraídas dos repositórios de software.

3.3.6 Necessidades dos Engenheiros de Software

Cada etapa do arcabouço conceitual atende a uma ou mais necessidades dos engenheiros de software. Desta maneira, pode-se entender melhor as práticas de desenvolvimento e a produtividade da equipe de determinado sistema, por exemplo. Com o intuito de descobrir quais são os questionamentos sobre o projeto de software que a inteligência analítica pode ajudar a responder, Begel e Zimmermann [11] realizaram dois *surveys* com profissionais da Microsoft. Eles categorizaram 145 questões que os engenheiros de software gostariam que os cientistas de dados respondessem, as quais foram categorizadas em 12 grupos. O resultado desta categorização é apresentado no Quadro 3.3.

Quadro 3.3 Necessidades dos Engenheiros de Software [11]

Medições de defeitos (Bugs) Onde os defeitos são encontrados, os mais comuns, seu ciclo de vida e custo o para consertá-los.	**Prática de Desenvolvimento** Relacionado ao código, depuração, perfil de desempenho, refatoração, *branching* em repositório, revisão de código, comentários e documentação de código. Estimativa de esforço e riscos devido as mudanças no código.	**Melhores Práticas** Melhores conduções no projeto de software. Como técnicas adequadas de migrações entre as versões de software, a melhor maneira de controlar os itens de trabalho. O momento certo para usar métodos formais para a análise de software ou quais critérios devem influenciar a decisão de usar uma linguagem de programação específica ou API
Prática de Teste Cobre automação de teste, estratégia de teste, unidade de teste, desenvolvimento dirigido ao teste, processos de teste, escritas, infraestruturas, medidas e impactos nos defeitos.	**Avaliação da Qualidade** Trata sobre otimização de código, melhores métricas, código duplicado e problemas de qualidade de software.	**Serviços** Forte relação com desenvolvimento na nuvem, os efeitos sobre a retenção de clientes e monetização causadas nas versões do software.
Clientes e Requisitos Relacionado aos interesses do cliente, gosto do cliente, compatibilidade com versões anteriores. Investimento em especificações técnicas, custo com o aumento de clientes.	**Ciclo de Vida do Desenvolvimento de Software** Como o tempo de desenvolvimento deve ser alocado entre planejamento, projeto, codificação e teste, impacto dessas alocações no software.	**Processo de Desenvolvimento de Software** Funcionamento das metodologias de processos de software e seus impactos na empresa.
Produtividade Investigar a métricas de qualidade para medir produtividade dos desenvolvedores, como monitorar a produtividade da equipe ou do indivíduo.	**Equipe e Colaboração** Formação de equipes, funções de desenvolvimento, quantidade de recursos e práticas para gestão do conhecimento.	**Reuso e Compartilhamento de Componentes** Reuso de código, seus componentes, custo de reaproveitamento de código.

3.4 IMPLEMENTAÇÃO DO ARCABOUÇO CONCEITUAL

Para demonstrar a utilização das técnicas de inteligência analítica na Engenharia de Software, elencadas no arcabouço conceitual apresentado, este trabalho explorou o repositório de código fonte do Jenkins. O Jenkins é um servidor de integração contínua escrito na linguagem Java [72]. Seu conteúdo está inteiramente disponível ao público, além de possuir uma comunidade bastante ativa. Essas características fornecem as condições ideais para promover os estudos sobre inteligência analítica na Engenharia de Software. Os projetos de software livre têm contribuído para o aumento das pesquisas em inteligência analítica, na medida em que utilizam Gestão de Configuração e Versões (SCM) e seus repositórios contêm informações históricas disponíveis com confiabilidade e detalhes suficientes para permitir conclusões significativas [24].

Diante da preocupação dos engenheiros de software em saber o que tem acontecido ao projeto [14], este trabalho se propõe a utilizar o arcabouço conceitual proposto, com a finalidade de expor os arquivos alterados durante seu

desenvolvimento e indicar por que eles foram alterados. Dessa maneira, foi desenvolvido um protótipo que identifica os tópicos mais relevantes nos comentários dos *commits* e os relacionam com os arquivos mais alterados em cada tópico.

Os *logs* dos *commits* têm se mostrado um bom indicador para entender as mudanças do software durante sua manutenção [3] [22] [24]. Já as regras de associações auxiliam na descoberta de padrões dentro de um conjunto de dados. Esses dados são chamados de "itens". Cada item faz parte de uma transação [1]. Assim, cada transação é um registro no banco de dados. Depois de ler uma transação, o algoritmo determina quais itens foram mais encontrados em todas as transações.

A seguir, são relacionam-se os métodos e as tecnologias utilizadas para compor este protótipo, com base nos elementos presentes no arcabouço. Conforme a Figura 3.1 é possível identificar as etapas e elementos relacionados.

• **Extração -** o primeiro passo para se realizar a análise de um projeto de software, consiste em obter os dados de seu repositório. Para esse, caso foram obtidos os dados do repositório de código fonte da *branch master* do Jenkins, disponível pelo SCM Git [72]. O repositório de código fonte foi clonado, para se trabalhar de forma local. A extração dos dados do repositório contou com a utilização da ferramenta CVSAnaly2, apresentado na Seção 3.3.1. Esta ferramenta permite a extração dos dados do repositório de controle de fonte dos repositórios CVS, Subversion e Git, que, após extraídas, ficam armazenadas em um banco de dados relacional [54] [55], que por padrão, é o MySQL.

• **Tipos de Análise -** *n*este protótipo, foram utilizados os tipos de análises que permitem obter conhecimento sobre o passado do sistema. Dessa maneira, contou-se com tipos de análise de tendência, sumarização e modelagem, apresentados pelo arcabouço conceitual na Seção 3.3.2. Para implementar as análises propostas, foram utilizados o método estatístico modelo de tópicos LDA e a regra de associação *Apriori*.

A análise do modelo de tópicos utilizou a API Java Mallet, em sua versão 2.07 [6]. Essa API propicia as condições para analisar textos com o modelo de tópicos LDA. Após extrair os textos dos *logs* dos *commits* do Jenkins, a API foi configurada para 10000 iterações e 40 tópicos. Ta configuração permite obter um resultado médio desejado para a análise dos dados.

Com a intenção de reduzir os ruídos de termos encontrados nos *logs* dos *commits,* utilizou-se um arquivo de *stop words*, ou seja, um arquivo que contem palavras comuns da língua inglesa, como: *a, the, and ,is* a qual não ajuda a gerar tópicos significativos [8]. Portanto, essas palavras são ignoradas na análise. Nesse *stop words,* foram adicionadas as palavras apresentadas no trabalho *What's a Typical Commit? A Characterization of Open Source Software Repositories* [3]. Essas palavras são bastante utilizadas nos *commits.* Portanto, não geram tópicos significativos, consultar Quadro 3.4.

Para descobrir os arquivos constantemente alterados de acordo com os termos encontrados nos *commits*, foi utilizado o algoritmo de regra de associação *Apriori*. Neste trabalho, foi utilizada a biblioteca SPMF, uma biblioteca de mineração de mineração de dados de código aberto escrito em Java, especializada em mineração de padrões [20].

A partir dos termos obtidos do modelo de tópicos, foram procurados os arquivos alterados pelos autores destes comentários. Dessa maneira, os autores que se referiram a determinado termo alteraram os arquivos encontrados pela regra de associação. A execução desse algoritmo contou com o suporte mínimo inicial de 20%, baixando seus valores para 15% e 10%, que obteve resultados mais satisfatórios para essa pesquisa. Ou seja, o percentual mínimo de vezes que o termo aparece na transação. Os arquivos *changelog* e *pom.xml* obtiveram as maiores alterações. Contudo, por se tratar de arquivos de configuração, é esperado que todas as alterações realizadas no sistema também alterem esses arquivos. Assim, este trabalho julgou por bem retirá-los das análises.

Quadro 3.4 Palavras mais usadas nos commits [3]

fix, add, test, file, new, support, chang, change, bug, patch, code, remov, set, work, update, get, error, build, function, typo, call, message, includ, path , need, merge, use, doc

Para enriquecer as análises trabalho, foram analisados também: quantidade de tipos de ações realizados nos *commits* por ano (arquivos modificação, adicionados, renomeados e deletados); quantidade de *commits* por ano; quantidade de desenvolvedores por ano; arquivos modificados por autor; e quantidade de *commits* por dia da semana. Para analisar esses dados, foi utilizada a ferramenta MicroStrategy Analytics Desktop [73] na versão *free.*

• **Visualização -** em relação à visualização dos dados, foram exibidos textos e gráficos como o de barra e a de linha e mapa de calor.

• **Necessidades dos Engenheiros de Software -** o protótipo desenvolvido permite atender à necessidade dos engenheiros de software de prática de desenvolvimento, visto que foram analisadas as alterações dos arquivos, que possibilitam aos engenheiros de software tomarem decisões a respeito das mudanças no código fonte do sistema. Apêndice A podem ser vistas as telas do protótipo.

3.5 CONFIGURAÇÃO DO GRUPO FOCAL

Com a finalidade de validar o arcabouço conceitual deste trabalho, criou-se um grupo focal com desenvolvedores e gestores de projetos de software em uma empresa de tecnologia da informação de grande porte do estado de Minas Gerais. O grupo focal é um método qualitativo que auxilia na validação de trabalhos empíricos. Trata-se de uma discussão planejada com o proposito de obter a percepção de um determinado grupo sobre um assunto. Em geral, é realizada com 3 a 12 participantes e guiada por um moderador, que mantém o foco da discussão [39] [40]. A condução do grupo focal deste trabalho segue os seguintes passos [39] [40]:

a) **Definição do problema** - por meio do grupo focal, teve por objetivo principal avaliar a utilização do arcabouço conceitual de inteligência analítica na Engenharia de Software apresentado na Seção 3.3 e

implementado na Seção 3.4. Com a implementação de parte desse arcabouço em um protótipo, remete-se a questão: O protótipo permite aos engenheiros de software obterem informações sobre a prática de desenvolvimento em um projeto de software? Ou seja, expõe os arquivos alterados durante o seu desenvolvimento e indicar porque eles foram alterados?

Para isso, procurou-se entender a visão de desenvolvedores e gerentes sobre questionamentos recorrentes na prática de desenvolvimento inspirados pela literatura nos trabalhos: *Estimating Development Effort in Free/Open Source Software Projects by Mining Software Repositories: A Case Study of OpenStack* [53], *Effort estimation of FLOSS projects: a study of the Linux kernel* [15], *Analyze this! 145 questions for data scientists in software engineering* [12] e *Information needs for software development analytics* [14]. A apresentação do protótipo do arcabouço conceitual ao grupo focal teve a intenção identificar os pontos fortes e os pontos fracos do arcabouço.

b) **Seleção dos participantes -** o critério de seleção dos participantes contemplou sua participação efetiva no processo de desenvolvimento de software. Eles foram selecionados segundo as funções realizadas na empresa, e indicados pela área de pesquisa da empresa e por gerentes dos setores. O grupo focal contou com seis participantes caracterizados na Seção 3.6.

c) **Condução do grupo focal –** para a condução do grupo focal, os participantes foram convidados para uma reunião, sendo que todos conheciam os objetivos principais da pesquisa, pois já haviam sido comunicados individualmente pelo próprio pesquisador. Durante a reunião, foram apresentados os objetivos principais e específicos da reunião e o termo de Consentimento de Participação em Grupo Focal. O pesquisador se apresentou como moderador, ficando responsável pelas anotações dos comentários e da gravação em áudio concedidas pelos participantes.

A reunião foi dividida em dois momentos. No primeiro, foram discutidas questões sobre os problemas do desenvolvimento de sistemas baseados na literatura. Após a discussão, foi apresentado o protótipo, com a utilização do arcabouço conceitual desenvolvido neste trabalho. Em seguida, foram levantadas as opiniões dos participantes sobre a resolução dos problemas discutidos e as soluções apresentadas pelo protótipo. Essas discussões são descritas na Seção 3.6.

a) **Análises -** para promover as análises dos dados gerados durante a reunião, o áudio da reunião foi gravado. Os participantes preencheram um formulário a respeito de informações profissionais, o qual permitiu caracterizá-los (formulário disponível no Apêndice B), e um formulário de consentimento do grupo focal (disponível no Apêndice C). Um formulário com os questionamentos para discussão. Durante a apresentação do protótipo, solicitou-se aos participantes que fizessem anotações sobre esses questionamentos. Após a apresentação do protótipo, cada um comentou sobre as questões. Ao final, o moderador apresentou as conclusões ao grupo, com a finalidade de reduzir os problemas de interpretação.

3.6 Resultados do Grupo Focal

3.6.1 Caracterização dos Participantes

A média de idade dos participantes foi de 35 anos; o tempo de experiência variou entre 7 a 25 anos na área de desenvolvimento de sistemas; todos os participantes graduaram-se em Ciência da Computação ou Sistemas de Informação. Havia dois mestres, um doutorando e dois pós-graduandos. Entre as certificações dos participantes, havia um *Scrum Master*, um certificado *Java Programming*, um com as certificações PMP, ITIL e MPS.BR e um com certificação *Caché ObjectScript*.

3.6.2 Discussões do Grupo Focal

Nesta Seção são apresentados os questionamentos referentes aos problemas encontrados na engenharia de software e os comentários do grupo focal. As questões foram tratadas antes e depois da apresentação da ferramenta implementada neste trabalho.

3.6.2.1 Importância das Questões

As questões discutidas a seguir tiveram como objetivo entender como os problemas enfrentados na prática de desenvolvimento são enfrentados pelos desenvolvedores e gestores.

- *Q1*- Como você consegue uma visão sobre o que de fato está acontecendo com o projeto?

Os participantes responderam que costumam obter a visão do projeto por meio de conversas com os membros do projeto, revisão de código, da documentação e de ferramentas como *Rational Team Concert* da IBM. Apesar das ferramentas existentes para obter uma visão do projeto, segundo um participante, "nada melhor do que conversar com as pessoas para saber o que está acontecendo". "Tem coisa que vem da cabeça do desenvolvedor e do analista que está só lá", completa.

- *Q2* - Quais indicadores ou métricas você utiliza para analisar o projeto de software?

Foram listados: ponto de função, quantidade de pessoas alocadas no projeto, quantidade de defeitos ou funcionalidades, quantidade de casos de uso, tempo de disponibilização de uma versão para outra, conjunto de métricas CK e horas trabalhadas no projeto. Porém, um participante revela que "não tenho segurança em meus indicadores". Já que muitos dados são colhidos manualmente e nem sempre são atualizados.

- *Q3* - Em quais situações você utiliza sua experiência ou "intuição" para tomar decisões sobre o projeto de software?

Os participantes revelam que em mais de 90% de suas tarefas eles sempre utilizam a experiência ou a intuição para tomar decisões. Apesar dos dados obtidos nos projetos de software, eles não devem ser usados sozinhos, sendo que a experiência do profissional minimiza o risco de uma decisão malfeita, mesmo de posse de informações sobre o projeto. Ressaltou-se, ainda, a dificuldade para obter informações sobre o sistema durante seu desenvolvimento. Em suma, a experiência é, geralmente utilizada para planejar projetos, principalmente em projetos muito grandes, em que se têm dificuldade para

gerenciar o processo e requisitos, ou até mesmo, durante uma proposta comercial. Um dos participantes afirma que se tem “a certeza que vamos errar no planejamento". Porém, "se você diminui o escopo, tem-se a certeza que vai errar menos", aconselha.

- *Q4* - Como você analisa a evolução do sistema que está sendo desenvolvido ou mantido?

Os participantes obtêm essa visão de forma externa, de acordo com as solicitações feitas pelo cliente, o uso do sistema pelos clientes, a quantidade de incidentes e as melhorias solicitadas, assim como acumulação de módulos e a complexidade acidental.

- *Q5* - Como você mede o seu esforço ou da equipe de desenvolvimento?

Foi consenso entre os participantes a utilização de horas para medir o esforço da equipe, ou seja, quantas horas são gastas para cada entrega. Um dos participantes mencionou que utiliza dados mais qualitativos, como as siglas P-M-G (Pequeno, Médio e Grande) para medir os esforços. Então, uma tarefa com P equivale a tarefas que consomem em até quatro horas; M, até seis horas; e G, até nove horas.

- *Q6* - Quais ferramentas você utiliza para gerar conhecimento sobre o projeto de software?

Foram relacionados os programas típicos do desenvolvimento de sistemas, como: Enterprise Architec, Wiki, Microsoft Project e Ganter, assim como repositórios de documentos e o próprio manual do sistema.

3.6.2.2 Adequação da ferramenta a resolução das questões

Após o debate sobre os questionamentos elencados, foi apresentado o protótipo, que trazia como exemplo as informações do sistema de código aberto Jenkins. Em seguida, foi verificado com os participantes se o exemplo apresentado resolveria os questionamentos discutidos. Para isso, as questões discutidas foram adaptadas. A seguir, são exibidas as questões e os comentários dos participantes.

- *Q1* - Com o protótipo apresentado foi possível ter uma visão do projeto?

A maioria os participantes respondeu afirmativamente, porém algumas visões poderiam ser acrescentadas, como mais visões arquiteturais e gerenciais. Identifica-se que o *dashboard* apresentado permite analisar a evolução do software e a contribuição dos desenvolvedores.

- *Q2* - Permite analisar o projeto por meio de indicadores ou métricas?

Os participantes afirmaram que foi possível analisar indicadores e métricas de desenvolvimento de forma quantitativa e qualitativa. Sugere-se que o sistema seja parametrizado, com indicadores e métricas fornecidos pelo usuário, bem como a classificação das entregas como corretiva ou evolutiva.

- *Q3* - Permite embasamento para tomada de decisão?

Os participantes apontaram que poderia auxiliar nas decisões como: momentos para se fazer *builds* da aplicação, iniciar testes, quais arquivos merecem maiores atenção na refatoração e decisões sobre o gerenciamento da equipe.

- *Q4* - Permite analisar a evolução do sistema?

Os participantes comentaram que o protótipo exibido pode ser útil para avaliar a maturidade e estabilidade do software, por meio dos tipos de alterações.

- *Q5* - É possível verificar o esforço despendido pelo desenvolvimento?

Este ponto foi elencado como negativo, pois os *commits* apresentados somente exibem suas entregas, não sabendo exatamente as horas gastas com a mudança ou sua complexidade. O sistema somente exibiu as contribuições feitas numa visão macro, ou seja, somente as entregas dos *commits*, não se outras informações sobre o esforço e o tempo gasto para aquela funcionalidade ou manutenção.

- *Q6* - Agrega conhecimento ao projeto?

Neste quesito, o protótipo permite verificar quem trabalhou em cada item e o que tem sido trabalhado. Um dos participantes ressaltou que permite ver o projeto por dentro e conhecer como estão suas mudanças, sendo útil para quem está no projeto compreender melhor o sistema.

3.6.2.3 Discussões Gerais

O grupo focal teve como finalidade descobrir se o protótipo permite aos engenheiros de software obterem informações sobre a prática de desenvolvimento em um projeto de software. Por meio do protótipo construído com os elementos elencados no arcabouço conceitual, é possível ter uma visão da prática de desenvolvimento, avaliar o projeto por meio de indicadores e métricas, analisar a evolução do sistema e servir como ferramenta para gerar conhecimento sobre o projeto de software. Contudo, não foi possível medir o esforço da equipe de desenvolvimento em sua totalidade.

Ressalta-se que, para a realização de análises mais produtivas, é essencial que se tenha o contexto do projeto que vai ser analisado. A combinação de análise de dados do projeto com o uso de metodologias ágeis podem tornar projetos mais fáceis de serem conduzidos. Visto que os participantes do grupo focal consideraram que a conversa com a equipe é importante para compreender o projeto. Apesar de a inteligência analítica auxiliar a encontrar informações a respeito do projeto de software, o engenheiro de software não deve dispensar o uso de sua experiência e intuição na tomada de decisão. A inteligência analítica agregada a experiência devem ser utilizadas em conjunto para maior assertividade em suas decisões.

O protótipo implementado não forneceu informações mais gerenciais e não explorou as métricas de produto como, Loc, complexidade ciclomática e métricas CK, que poderiam agregar mais informações ao projeto. Apesar disso, entende-se que outros trabalhos têm suprido esse tipo de pesquisa [59]. A classificação dos *commits* como corretivas ou melhorias poderia facilitar as análises e revelar novos conhecimentos. Assim, como a análise dos *commits* por versão.

Entende-se que o protótipo implementado pode ser utilizado como ferramenta para se obter informações do

projeto de software e expor as alterações no sistema. Permite ainda, entender as mudanças realizadas no sistema por meio dos *commits*. Assim como o arcabouço conceitual proposto neste trabalho, é indicado que seja feitas adaptações do trabalho para sua implantação em outros projetos.

3.7 TRABALHOS RELACIONADOS

A inteligência analítica na Engenharia de Software tem sido recorrente em diversas pesquisas. Gupta [27] propôs um *framework* para minerar dados de repositórios de software de múltiplas perspectivas, o *Nirikshan*. Para isso, realizou um estudo de caso focando um processo de mineração, para identificar papéis organizacionais, padrão de interação e distribuição de trabalho entre os recursos e as atividades executadas por um recurso particular. Este trabalho também propõe um arcabouço para explorar dados em repositórios de software. Contudo, a proposta do arcabouço deste trabalho não foca no processo e no entendimento da dinâmica organizacional. Ele mostra como diversas tecnologias podem ser utilizadas para atender às necessidades dos engenheiros de software.

Poncin et al. [49] sugerem a aplicação de uma técnica de mineração de repositório que relaciona o papel dos desenvolvedores e o fluxo de defeitos. Para avaliar o *framework* criado, eles desenvolveram um protótipo (FRASR) e realizaram um estudo de caso em um software livre. Diferente do trabalho apresentado por Poncin et al. [49], o arcabouço deste estudo não é focado em processos, mas sim em tecnologias utilizadas. Foi também construído um protótipo, contudo este foi avaliado por um grupo focal.

Zhang e Sheth [70] utilizaram as técnicas de inteligência analítica para analisar um sistema de uma empresa de telecomunicações. Desse sistema foram extraídas as métricas de software. Com base em técnicas estatísticas aplicadas às métricas extraídas, conseguiram compreender melhor o projeto de software deste sistema. Consequentemente, conseguiu-se melhorar seu gerenciamento. Assim como no rabalho de Zhang e Sheth [70], este trabalho pretende clarear a compreensão de projeto de software por meio da inteligência analítica. Diferente da pesquisa de Zhang e Sheth [61] este trabalho elencou os elementos tecnológicos necessários à criação de um arcabouço conceitual da inteligência analítica na engenharia de software.

Gonzalez-Barahona et al. [25] estudaram a evolução de software de um software livre, o Glibc, que tem mais de vinte anos de projeto. Eles correlacionaram análises sobre as atividades dos desenvolvedores e a evolução do tamanho do código desse sistema com a Lei de Lehman. Foram identificadas cinco leis, a primeira, a segunda, a quarta, a quinta e a sexta lei. A primeira Lei de Lehman, a mudança contínua, foi claramente identificada. A segunda lei, o aumento da complexidade, durou alguns períodos. A quarta lei, a conservação da estabilidade organizacional, esteve até certo período. A quinta lei, a conservação de familiaridade, não se mostra durável durante todas as fases do projeto. E a sexta lei, o crescimento contínuo, não durou até a última fase do projeto. Nesse trabalho, Gonzalez-Barahona et al. [25] demonstraram a metodologia para realizar o estudo e possibilitar a compreensão de grandes e longos projetos de software. Por meio do presente trabalho, é possível analisar a evolução de software. Contudo, tem-se como ideia central a compreensão das alterações realizadas no projeto de software.

Utilizando o modelo de tópicos, Barua et al. [8] analisaram o fórum *Stack Overflow*, no qual se promove a troca de conhecimentos entre desenvolvedores de software. Por meio dessa pesquisa, é possível verificar as tendências e os conhecimentos das tecnologias, além de compreender os pensamentos e as necessidades dos desenvolvedores. Entre os achados desse trabalho, tem-se: aumento do uso de tecnologias móveis; a popularidade da linguagem Java e PHP; o declínio dot.Net, além do aumento das discussões sobre o repositório Git e o banco de dados MySQL. Este trabalho demonstra a utilização do arcabouço conceitual com a técnica de modelo de tópicos. Porém, ao contrário de Barua et al. [8], foram analisados os textos dos *logs* dos *commits*.

Outras técnicas de análise de dados são comumente usadas na Engenharia de Software. Zimmermann et al. [71] exploraram a técnica de regra de associação para criar a ferramenta Rose, por meio da qual é possível saber as alterações no código fonte feitas juntas. Dessa maneira, a ferramenta recomenda alterações dos arquivos aos programadores, para evitar erros de mudanças incompletas, além de detectar acoplamento. Este trabalho também explorou a técnica de regras de associação, contrapondo-se ao trabalho de Zimmermann et al. [71]. A intenção não é recomendar alterações no código, mas sim mapear as alterações de acordo com os tópicos encontrados.

3.8 CONCLUSÃO E TRABALHOS FUTUROS

Este trabalho teve por objetivo elencar os elementos tecnológicos de um arcabouço conceitual de inteligência analítica na Engenharia de Software. Para isso, foram apresentadas as tecnologias presentes nas etapas de extração, tipos de análise, visualização dos dados e as necessidades dos engenheiros de software. Para avaliar a utilização do arcabouço conceitual, foi implementado um protótipo, que utilizou como exemplo os dados do sistema de código aberto Jenkins. Por meio de um grupo focal, foi possível verificar que o protótipo apresentado permite que sejam entendidas as mudanças do software sem a necessidade de fazer uma investigação profunda no código. Essas informações permitem aos engenheiros de software tomarem decisões baseadas em dados do projeto em relação a: refatoração, *builds*, planejamento de testes e gerenciamento da equipe. Teve como ponto negativo do protótipo apresentado, o entendimento do esforço de desenvolvimento baseado no tempo gasto por tarefa.

Este trabalho contribuiu com uma proposta de um arcabouço conceitual de inteligência analítica na engenharia de software, que utiliza tecnologias encontradas na literatura com o objetivo de guiar o engenheiro de software em sua implementação. Sendo que o protótipo implementado neste trabalho permite compreender as alterações realizadas no sistema.

Entende-se que é necessário evoluir o arcabouço adicionando novas tecnologias e métodos. O protótipo apresentado supriu apenas uma parte das necessidades

dos engenheiros de software, espera-se que estudos futuros possam melhorar o arcabouço aqui proposto.

É indicado como trabalho futuro que este arcabouço seja implantado em uma empresa de software para melhor compreender seus benefícios e desafio.

Um exemplo do *dashboard* com os dados do repositório do Jenkins pode ser visto no link: http://goo.gl/OD8KAo e no Apêndice A.

3.9 REFERÊNCIAS

[1] Rakesh Agrawal, Tomasz Imieliński, and Arun Swami. 1993. Mining Association Rules Between Sets of Items in Large Databases. *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, ACM, 207–216. DOI= http://doi.org/10.1145/170035.170072

[2] Rakesh Agrawal, Ramakrishnan Srikant, and others. 1994. Fast algorithms for mining association rules. *Proc. 20th int. conf. very large data bases, VLDB*, 487–499. Retrieved April 9, 2016 from https://www.it.uu.se/edu/course/homepage/infoutv/ht08/vldb94_rj.pdf

[3] A. Alali, H. Kagdi, and J.I. Maletic. 2008. What's a Typical Commit? A Characterization of Open Source Software Repositories. *The 16th IEEE International Conference on Program Comprehension, 2008. ICPC 2008*, 182–191. DOI= http://doi.org/10.1109/ICPC.2008.24

[4] Anahita Alipour, Abram Hindle, and Eleni Stroulia. 2013. A Contextual Approach Towards More Accurate Duplicate Bug Report Detection. *Proceedings of the 10th Working Conference on Mining Software Repositories*, IEEE Press, 183–192. Retrieved April 9, 2016 from http://dl.acm.org/citation.cfm?id=2487085.2487123

[5] M. Allamanis and C. Sutton. 2013. Mining source code repositories at massive scale using language modeling. *2013 10th IEEE Working Conference on Mining Software Repositories (MSR)*, 207–216. DOI= http://doi.org/10.1109/MSR.2013.6624029

[6] Andrew Kachites McCallum. 2002. MALLET: A Machine Learning for Language Toolkit. Retrieved April 9, 2016 from http://mallet.cs.umass.edu

[7] A. Bachmann and A. Bernstein. 2010. When process data quality affects the number of bugs: Correlations in software engineering datasets. *2010 7th IEEE Working Conference on Mining Software Repositories (MSR)*, 62–71. DOI= http://doi.org/10.1109/MSR.2010.5463286

[8] Anton Barua, Stephen W. Thomas, and Ahmed E. Hassan. 2014. What are developers talking about? An analysis of topics and trends in Stack Overflow. *Empir Software Eng* 19, 3: 619–654. DOI= http://doi.org/10.1007/s10664-012-9231-y

[9] O. Baysal. 2013. Informing development decisions: From data to information. *2013 35th International Conference on Software Engineering (ICSE)*, 1407–1410. DOI= http://doi.org/10.1109/ICSE.2013.6606729

[10] Andrew Begel, Yit Phang Khoo, and Thomas Zimmermann. 2010. Codebook: Discovering and Exploiting Relationships in Software Repositories. *Proceedings of the 32Nd ACM/IEEE International Conference on Software Engineering - Volume 1*, ACM, 125–134. DOI= http://doi.org/10.1145/1806799.1806821

[11] Andrew Begel and Thomas Zimmermann. 2014. Analyze This! 145 Questions for Data Scientists in Software Engineering. *Proceedings of the 36th International Conference on Software Engineering*, ACM, 12–23. DOI= http://doi.org/10.1145/2568225.2568233

[12] Andrew Begel and Thomas Zimmermann. 2014. Analyze this! 145 questions for data scientists in software engineering. ACM Press, 12–23. DOI= http://doi.org/10.1145/2568225.2568233

[13] Raymond P.L. Buse and Thomas Zimmermann. 2010. Analytics for Software Development. *Proceedings of the FSE/SDP Workshop on Future of Software Engineering Research*, ACM, 77–80. DOI= http://doi.org/10.1145/1882362.1882379

[14] R.P.L. Buse and T. Zimmermann. 2012. Information needs for software development analytics. *2012 34th International Conference on Software Engineering (ICSE)*, 987–996. DOI= http://doi.org/10.1109/ICSE.2012.6227122

[15] Andrea Capiluppi and Daniel Izquierdo-Cortázar. 2013. Effort estimation of FLOSS projects: a study of the Linux kernel. *Empir Software Eng* 18, 1: 60–88. DOI= http://doi.org/10.1007/s10664-011-9191-7

[16] M. Ceccarelli, L. Cerulo, G. Canfora, and M. Di Penta. 2010. An eclectic approach for change impact analysis. *2010 ACM/IEEE 32nd International Conference on Software Engineering*, 163–166. DOI= http://doi.org/10.1145/1810295.1810320

[17] Bogdan Dit, Michael Wagner, Shasha Wen, et al. 2014. ImpactMiner: A Tool for Change Impact Analysis. *Companion Proceedings of the 36th International Conference on Software Engineering*, ACM, 540–543. DOI= http://doi.org/10.1145/2591062.2591064

[18] Emad A. El-Sebakhy. 2011. Functional networks as a novel data mining paradigm in forecasting software development efforts. *Expert Systems with Applications* 38, 3: 2187–2194. DOI= http://doi.org/10.1016/j.eswa.2010.08.005

[19] Jacqui Finlay, Russel Pears, and Andy M. Connor. 2014. Data stream mining for predicting software build outcomes using source code metrics. *Information and Software Technology* 56, 2: 183–198. DOI= http://doi.org/10.1016/j.infsof.2013.09.001

[20] Philippe Fournier-Viger, Antonio Gomariz, Ted Gueniche, Azadeh Soltani, Cheng-Wei Wu, and Vincent S. Tseng. 2014. SPMF: A Java Open-source Pattern Mining Library. *J. Mach. Learn. Res.* 15, 1: 3389–3393.

[21] Ying Fu, Meng Yan, Xiaohong Zhang, Ling Xu, Dan Yang, and Jeffrey D. Kymer. Automated classification of software change messages by semi-supervised Latent Dirichlet Allocation. *Information and Software Technology*. DOI= http://doi.org/10.1016/j.infsof.2014.05.017

[22] S. Gala-Perez, G. Robles, J.M. Gonzalez-Barahona, and I. Herraiz. 2013. Intensive metrics for the study of the evolution of open source

projects: Case studies from Apache Software Foundation projects. *2013 10th IEEE Working Conference on Mining Software Repositories (MSR)*, 159–168. DOI= http://doi.org/10.1109/MSR.2013.6624023

[23] Vahid Garousi and James Leitch. 2010. IssuePlayer: An extensible framework for visual assessment of issue management in software development projects. *Journal of Visual Languages & Computing* 21, 3: 121–135. DOI= http://doi.org/10.1016/j.jvlc.2010.03.001

[24] Jesus M. Gonzalez-Barahona, Gregorio Robles, Israel Herraiz, and Felipe Ortega. 2014. Studying the laws of software evolution in a long-lived FLOSS project. *J. Softw. Evol. and Proc.* 26, 7: 589–612. DOI= http://doi.org/10.1002/smr.1615

[25] Jesus M. Gonzalez-Barahona, Gregorio Robles, Israel Herraiz, and Felipe Ortega. 2014. Studying the laws of software evolution in a long-lived FLOSS project. *Journal of Software: Evolution and Process* 26, 7: 589–612. DOI= http://doi.org/10.1002/smr.1615

[26] Antonio González-Torres, Francisco J. García-Peñalvo, and Roberto Therón. 2013. Human–computer interaction in evolutionary visual software analytics. *Computers in Human Behavior* 29, 2: 486–495. DOI= http://doi.org/10.1016/j.chb.2012.01.013

[27] Monika Gupta. 2014. Nirikshan: Process Mining Software Repositories to Identify Inefficiencies, Imperfections, and Enhance Existing Process Capabilities. *Companion Proceedings of the 36th International Conference on Software Engineering*, ACM, 658–661. DOI= http://doi.org/10.1145/2591062.2591080

[28] Jongdae Han and Woosung Jung. 2014. Extracting communication structure of a development organization from a software repository. *Pers Ubiquit Comput* 18, 6: 1413–1421. DOI= http://doi.org/10.1007/s00779-013-0742-3

[29] A.E. Hassan. 2006. Mining Software Repositories to Assist Developers and Support Managers. *22nd IEEE International Conference on Software Maintenance, 2006. ICSM '06*, 339–342. DOI= http://doi.org/10.1109/ICSM.2006.38

[30] A.E. Hassan. 2008. The road ahead for Mining Software Repositories. *Frontiers of Software Maintenance, 2008. FoSM 2008.*, 48–57. DOI= http://doi.org/10.1109/FOSM.2008.4659248

[31] A.E. Hassan and Tao Xie. 2010. Mining software engineering data. *2010 ACM/IEEE 32nd International Conference on Software Engineering*, 503–504. DOI= http://doi.org/10.1145/1810295.1810451

[32] Ahmed E. Hassan and Tao Xie. 2010. Software Intelligence: The Future of Mining Software Engineering Data. *Proceedings of the FSE/SDP Workshop on Future of Software Engineering Research*, ACM, 161–166. DOI= http://doi.org/10.1145/1882362.1882397

[33] Fehmi Jaafar, Yann-Gaël Guéhéneuc, Sylvie Hamel, and Giuliano Antoniol. 2014. Detecting asynchrony and dephase change patterns by mining software repositories. *J. Softw. Evol. and Proc.* 26, 1: 77–106. DOI= http://doi.org/10.1002/smr.1635

[34] Huzefa Kagdi, Michael L. Collard, and Jonathan I. Maletic. 2007. A survey and taxonomy of approaches for mining software repositories in the context of software evolution. *J. Softw. Maint. Evol.: Res. Pract.* 19, 2: 77–131. DOI= http://doi.org/10.1002/smr.344

[35] F. Khomh, M. Di Penta, and Y. Guéhéneuc. 2009. An Exploratory Study of the Impact of Code Smells on Software Change-proneness. *16th Working Conference on Reverse Engineering, 2009. WCRE '09*, 75–84. DOI= http://doi.org/10.1109/WCRE.2009.28

[36] C. Kiefer, A. Bernstein, and J. Tappolet. 2007. Mining Software Repositories with iSPAROL and a Software Evolution Ontology. *Fourth International Workshop on Mining Software Repositories, 2007. ICSE Workshops MSR '07*, 10–10. DOI= http://doi.org/10.1109/MSR.2007.21

[37] Sunghun Kim, Hongyu Zhang, Rongxin Wu, and Liang Gong. 2011. Dealing with Noise in Defect Prediction. *Proceedings of the 33rd International Conference on Software Engineering*, ACM, 481–490. DOI= http://doi.org/10.1145/1985793.1985859

[38] Oleksii Kononenko, Olga Baysal, Reid Holmes, and Michael W. Godfrey. 2014. Mining Modern Repositories with Elasticsearch. *Proceedings of the 11th Working Conference on Mining Software Repositories*, ACM, 328–331. DOI= http://doi.org/10.1145/2597073.2597091

[39] J. Kontio, L. Lehtola, and J. Bragge. 2004. Using the focus group method in software engineering: obtaining practitioner and user experiences. *2004 International Symposium on Empirical Software Engineering, 2004. ISESE '04. Proceedings*, 271–280. DOI= http://doi.org/10.1109/ISESE.2004.1334914

[40] Jyrki Kontio, Johanna Bragge, and Laura Lehtola. 2008. The Focus Group Method as an Empirical Tool in Software Engineering. In *Guide to Advanced Empirical Software Engineering*, Forrest Shull, Janice Singer and Dag I. K. Sjøberg (eds.). Springer London, 93–116. Retrieved May 20, 2016 from http://link.springer.com/chapter/10.1007/978-1-84800-044-5_4

[41] S. Lal and A. Sureka. 2012. Comparison of Seven Bug Report Types: A Case-Study of Google Chrome Browser Project. *Software Engineering Conference (APSEC), 2012 19th Asia-Pacific*, 517–526. DOI= http://doi.org/10.1109/APSEC.2012.54

[42] Bixin Li, Xiaobing Sun, Hareton Leung, and Sai Zhang. 2013. A survey of code-based change impact analysis techniques. *Softw. Test. Verif. Reliab.* 23, 8: 613–646. DOI= http://doi.org/10.1002/stvr.1475

[43] M. Linares-Vasquez, B. Dit, and D. Poshyvanyk. 2013. An exploratory analysis of mobile development issues using stack overflow. *2013 10th IEEE Working Conference on Mining Software Repositories (MSR)*, 93–96. DOI= http://doi.org/10.1109/MSR.2013.6624014

[44] Mario Linares-Vásquez, Collin McMillan, Denys Poshyvanyk, and Mark Grechanik. 2014. On using machine learning to automatically classify

software applications into domain categories. *Empir Software Eng* 19, 3: 582–618. DOI= http://doi.org/10.1007/s10664-012-9230-z

[45] Erik Linstead, Sushil Bajracharya, Trung Ngo, Paul Rigor, Cristina Lopes, and Pierre Baldi. 2009. Sourcerer: mining and searching internet-scale software repositories. *Data Min Knowl Disc* 18, 2: 300–336. DOI= http://doi.org/10.1007/s10618-008-0118-x

[46] G. Maskeri, D. Karnam, S.A. Viswanathan, and S. Padmanabhuni. 2012. Bug Prediction Metrics Based Decision Support for Preventive Software Maintenance. *Software Engineering Conference (APSEC), 2012 19th Asia-Pacific*, 260–269. DOI= http://doi.org/10.1109/APSEC.2012.43

[47] T. Menzies and T. Zimmermann. 2013. Software Analytics: So What? *IEEE Software* 30, 4: 31–37. DOI= http://doi.org/10.1109/MS.2013.86

[48] Renato Lima Novais, André Torres, Thiago Souto Mendes, Manoel Mendonça, and Nico Zazworka. 2013. Software evolution visualization: A systematic mapping study. *Information and Software Technology* 55, 11: 1860–1883. DOI= http://doi.org/10.1016/j.infsof.2013.05.008

[49] W. Poncin, A. Serebrenik, and M. van den Brand. 2011. Process Mining Software Repositories. *2011 15th European Conference on Software Maintenance and Reengineering (CSMR)*, 5–14. DOI= http://doi.org/10.1109/CSMR.2011.5

[50] Ayushi Rastogi and Ashish Sureka. 2014. SamikshaViz: A Panoramic View to Measure Contribution and Performance of Software Maintenance Professionals by Mining Bug Archives. *Proceedings of the 7th India Software Engineering Conference*, ACM, 2:1–2:10. DOI= http://doi.org/10.1145/2590748.2590750

[51] Meghan Revelle, Malcom Gethers, and Denys Poshyvanyk. 2011. Using structural and textual information to capture feature coupling in object-oriented software. *Empir Software Eng* 16, 6: 773–811. DOI= http://doi.org/10.1007/s10664-011-9159-7

[52] Gregorio Robles, Jesús M. González-Barahona, Carlos Cervigón, Andrea Capiluppi, and Daniel Izquierdo-Cortázar. 2014. Estimating Development Effort in Free/Open Source Software Projects by Mining Software Repositories: A Case Study of OpenStack. *Proceedings of the 11th Working Conference on Mining Software Repositories*, ACM, 222–231. DOI= http://doi.org/10.1145/2597073.2597107

[53] Gregorio Robles, Jesús M. González-Barahona, Carlos Cervigón, Andrea Capiluppi, and Daniel Izquierdo-Cortázar. 2014. Estimating Development Effort in Free/Open Source Software Projects by Mining Software Repositories: A Case Study of OpenStack. *Proceedings of the 11th Working Conference on Mining Software Repositories*, ACM, 222–231. DOI= http://doi.org/10.1145/2597073.2597107

[54] Gregorio Robles, Jesus M. Gonzalez-Barahona, and Juan Julian Merelo. 2006. Beyond source code: The importance of other artifacts in software development (a case study). *Journal of Systems and Software* 79, 9: 1233–1248. DOI= http://doi.org/10.1016/j.jss.2006.02.048

[55] Gregorio Robles, Stefan Koch, Jesús M. GonZÁlEZ-BARAHonA, and Juan Carlos. 2004. Remote analysis and measurement of libre software systems by means of the CVSAnalY tool. *Proceedings of the 2nd ICSE Workshop on Remote Analysis and Measurement of Software Systems (RAMSS)*, 51–56. Retrieved May 18, 2016 from http://flosshub.org/system/files/robles-koch-barahona_cvsanaly.pdf

[56] R.W. Selby. 2005. Enabling reuse-based software development of large-scale systems. *IEEE Transactions on Software Engineering* 31, 6: 495–510. DOI= http://doi.org/10.1109/TSE.2005.69

[57] Weiyi Shang, Bram Adams, and Ahmed E. Hassan. 2012. Using Pig as a data preparation language for large-scale mining software repositories studies: An experience report. *Journal of Systems and Software* 85, 10: 2195–2204. DOI= http://doi.org/10.1016/j.jss.2011.07.034

[58] Weiyi Shang, Zhen Ming Jiang, B. Adams, and A.E. Hassan. 2009. MapReduce as a general framework to support research in Mining Software Repositories (MSR). *6th IEEE International Working Conference on Mining Software Repositories, 2009. MSR '09*, 21–30. DOI= http://doi.org/10.1109/MSR.2009.5069477

[59] F.Z. Sokol, M. Finavaro Aniche, and M.A. Gerosa. 2013. MetricMiner: Supporting researchers in mining software repositories. *2013 IEEE 13th International Working Conference on Source Code Analysis and Manipulation (SCAM)*, 142–146. DOI= http://doi.org/10.1109/SCAM.2013.6648195

[60] Mark Steyvers and Tom Griffiths. 2007. Probabilistic topic models. *Handbook of latent semantic analysis* 427, 7: 424–440.

[61] Jonas Tappolet, Christoph Kiefer, and Abraham Bernstein. 2010. Semantic web enabled software analysis. *Web Semantics: Science, Services and Agents on the World Wide Web* 8, 2–3: 225–240. DOI= http://doi.org/10.1016/j.websem.2010.04.009

[62] Stephen W. Thomas. 2011. Mining Software Repositories Using Topic Models. *Proceedings of the 33rd International Conference on Software Engineering*, ACM, 1138–1139. DOI= http://doi.org/10.1145/1985793.1986020

[63] Stephen W. Thomas, Bram Adams, Ahmed E. Hassan, and Dorothea Blostein. 2014. Studying software evolution using topic models. *Science of Computer Programming* 80, Part B: 457–479. DOI= http://doi.org/10.1016/j.scico.2012.08.003

[64] Suresh Thummalapenta and Tao Xie. 2011. Alattin: mining alternative patterns for defect detection. *Autom Softw Eng* 18, 3–4: 293–323. DOI= http://doi.org/10.1007/s10515-011-0086-z

[65] Michael Würsch, Gerald Reif, Serge Demeyer, and Harald C. Gall. 2010. Fostering Synergies: How Semantic Web Technology Could Influence Software Repositories. *Proceedings of 2010 ICSE Workshop on Search-driven Development: Users, Infrastructure, Tools and Evaluation*, ACM, 45–48. DOI= http://doi.org/10.1145/1809175.1809187

[66] Xinrong Xie, D. Poshyvanyk, and A. Marcus. 2006. Visualization of CVS Repository Information. *13th Working Conference on Reverse*

Engineering, 2006. WCRE '06, 231–242. DOI= http://doi.org/10.1109/WCRE.2006.55

[67] Dongmei Zhang, Yingnong Dang, Jian-Guang Lou, Shi Han, Haidong Zhang, and Tao Xie. 2011. Software Analytics As a Learning Case in Practice: Approaches and Experiences. *Proceedings of the International Workshop on Machine Learning Technologies in Software Engineering*, ACM, 55–58. DOI= http://doi.org/10.1145/2070821.2070829

[68] Dongmei Zhang, Shi Han, Yingnong Dang, Jian-Guang Lou, Haidong Zhang, and Tao Xie. 2013. Software Analytics in Practice. *IEEE Software* 30, 5: 30–37. DOI= http://doi.org/10.1109/MS.2013.94

[69] Hongyu Zhang. 2008. Exploring Regularity in Source Code: Software Science and Zipf's Law. *15th Working Conference on Reverse Engineering, 2008. WCRE '08*, 101–110. DOI= http://doi.org/10.1109/WCRE.2008.37

[70] Y. Zhang and D. Sheth. 2006. Mining software repositories for model-driven development. *IEEE Software* 23, 1: 82–90. DOI= http://doi.org/10.1109/MS.2006.23

[71] T. Zimmermann, A. Zeller, P. Weissgerber, and S. Diehl. 2005. Mining version histories to guide software changes. *IEEE Transactions on Software Engineering* 31, 6: 429–445. DOI= http://doi.org/10.1109/TSE.2005.72

[72] Jenkins. Retrieved April 9, 2016 from https://jenkins.io/

[73] Business Intelligence Software and Data Analytics. *MicroStrategy*. Retrieved May 18, 2016 from http://www.microstrategy.com/us

4 CONSIDERAÇÕES FINAIS

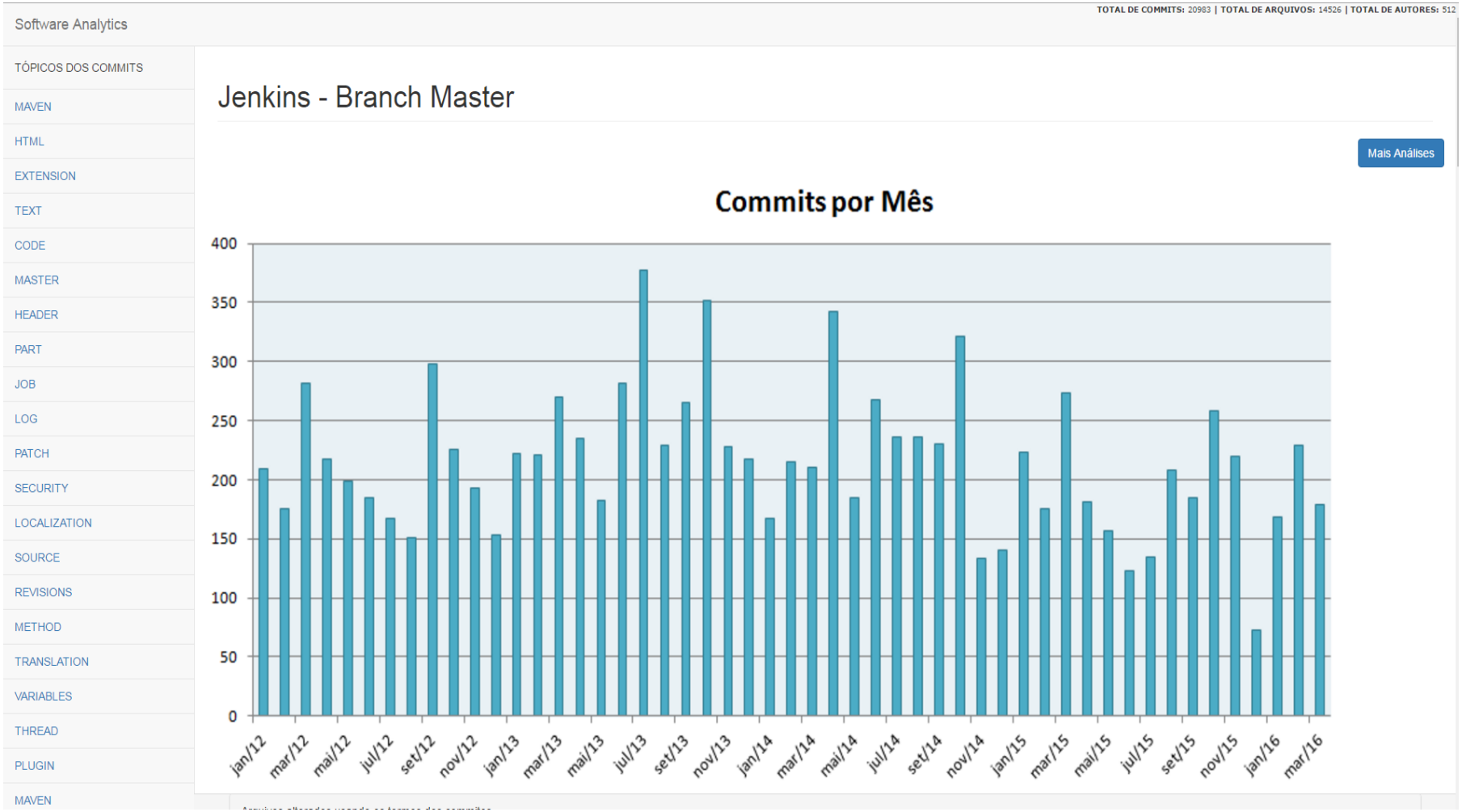
Este trabalho teve por objetivo principal propor um arcabouço conceitual de inteligência analítica aplicada à Engenharia de Software, com a finalidade de fornecer múltiplas visões do projeto, a partir das informações geradas pelos artefatos produzidos em seu desenvolvimento. Para alcançar este objetivo, identificou os elementos do arcabouço conceitual (OBJ1), por meio do mapeamento sistemático da literatura, e identificou os métodos e as tecnologias utilizadas na literatura (OBJ2). De posse desses dados, foi produzido um arcabouço conceitual de inteligência analítica (OBJ3) que visa fornecer a base necessária para sua utilização em projetos de software.

Visto que o mapeamento sistemático apresentado neste trabalho revela que a inteligência analítica tem sido fortemente utilizada durante a fase de manutenção e na prática de desenvolvimento, na qual engenheiros de software precisam entender o que foi alterado no sistema e o seu motivo, este trabalho desenvolveu um protótipo com base no arcabouço conceitual que atendesse a tal propósito. Este protótipo foi avaliado por um grupo focal composto por profissionais da área de desenvolvimento de sistemas (OBJ4) , o qual constatou que o protótipo permite entender às mudanças dos artefatos do sistema sem a necessidade de realizar grandes investigações em seu código, documentos e repositórios.

Este trabalho contribuiu para o aumento da compreensão da utilização da inteligência analítica na Engenharia de Software e sua aplicabilidade. Como trabalhos futuros, pretende-se aplicar as técnicas apresentadas no arcabouço conceitual em uma empresa de desenvolvimento de sistemas, que permitirá verificar os reais desafios para sua implantação, assim como a solução de outras necessidades dos engenheiros de software, como necessidades mais gerenciais.

5 APÊNDICES

Apêndice A – Protótipo Desenvolvido



CHANGE
ISSUE
CHANGE
FILES
REMOTE-TRACKING
SYMBOL
SLAVE
API
NULL
PAGE
TAG
NODE
THROWS
PULL
ICON
PREPARE
COMMIT
LINE
FORM

Quem usou o Termo: **maven** alterou os arquivos:

Maven.java, Maven3Builder.java, MavenBuild.java, MavenJob.java, MavenModuleSet.java, MavenModuleSetBuild.java, SurefireArchiver.java, MavenProcessFactory.java, MavenBuilder.java, RedeployPublisher.java

Quem usou o Termo: **html** alterou os arquivos:

Functions.java, style.css

Quem usou o Termo: **extension** alterou os arquivos:

Functions.java, Hudson.java, Node.java, Run.java, , ExtensionFinder.java, AbstractBuild.java, AbstractProject.java, MatrixProject.java

Quem usou o Termo: **text** alterou os arquivos:

Run.java, hudson-behavior.js, AbstractProject.java, Messages.properties

Quem usou o Termo: **code** alterou os arquivos:

Run.java

Quem usou o Termo: **master** alterou os arquivos:

FilePath.java, Computer.java, Executor.java, Hudson.java, Queue.java, Run.java, JDKInstaller.java, executors.jelly, SlaveComputer.java, MavenBuild.java, AbstractBuild.java

Quem usou o Termo: **header** alterou os arquivos:

Mailer.java, layout.jelly, hudson-behavior.js, MailSender.java, HistoryWidget.java, ParametersDefinitionProperty.java

Quem usou o Termo: **part** alterou os arquivos:

FilePath.java, Util.java, Hudson.java, Job.java, Fingerprinter.java, AbstractBuild.java, AbstractProject.java, MavenModuleSetBuild.java, MatrixBuild.java

Quem usou o Termo: **job** alterou os arquivos:

Job.java, AbstractProject.java, ListView.java

Quem usou o Termo: **log** alterou os arquivos:

Run.java, AbstractBuild.java

Quem usou o Termo: **patch** alterou os arquivos:

Queue.java, AbstractProject.java

Quem usou o Termo: **security** alterou os arquivos:

layout.jelly, AbstractProject.java, HudsonPrivateSecurityRealm.java

Quem usou o Termo: **localization** alterou os arquivos:

Messages.properties, layout_ja.properties, sidepanel_ja.properties, queue_ja.properties, Messages_ja.properties, Messages_ja.properties, Messages_ja.properties, sidepanel_fr.properties, queue_fr.properties, index_fr.properties, configure-common_fr.properties, sidepanel_fr.properties, configure_fr.properties, fingerprintCheck_fr.properties, manage_fr.properties, global_fr.properties, buildCaption_fr.properties, buildProgressBar_fr.properties, executors_fr.properties, rssBar_fr.properties, systemInfo_ja.properties, configure-entries_de.properties, configure-common_de.properties, sidepanel_de.properties, configure_de.properties, systemInfo_de.properties, Messages_de.properties, Messages_de.properties, index_fr.properties, index_de.properties, rssBar_tr.properties, body_fr.properties, installed_ja.properties, installed_fr.properties, table_ja.properties, table_fr.properties, index_fr.properties, index_fr.properties, index_fr.properties, configure_de.properties, index_de.properties, index_de.properties, tasks_ja.properties, tasks_de.properties, index_fi.properties, tasks_es_ES.properties, layout_fi.properties, sidepanel_es.properties, main_es.properties, description_es.properties, buildListTable_es.properties, editableDescription_es.properties, configure_zh_CN.properties, console_zh_CN.properties, columnHeader_fi.properties, advanced_zh_TW.properties

Quem usou o Termo: **source** alterou os arquivos:

ClassicPluginStrategy.java

Quem usou o Termo: **revisions** alterou os arquivos:

Descriptor.java, Project.java, Slave.java, AbstractProjectTest.java, Mailer.java, RobustCollectionConverter.java, XStream2.java, Cause.java MavenJob.java, AbstractProject.java, OldDataMonitor.java, GlobalMatrixAuthorizationStrategy.java, ParametersAction.java, AuthorizationMatrixProperty.java

Quem usou o Termo: **method** alterou os arquivos:

Run.java, AbstractProject.java

Quem usou o Termo: **translation** alterou os arquivos:

Messages_fr.properties

Quem usou o Termo: **variables** alterou os arquivos:

EnvVars.java, Launcher.java, Build.java, Ant.java, Maven.java, BuildWrapper.java, MavenBuild.java, AbstractBuild.java, MavenModuleSet.java, MavenModuleSetBuild.java

Quem usou o Termo: **thread** alterou os arquivos:

Quem usou o Termo: **plugin** alterou os arquivos:

PluginManager.java, PluginWrapper.java, ClassicPluginStrategy.java

Quem usou o Termo: **maven** alterou os arquivos:

Maven.java, Maven3Builder.java, MavenBuild.java, MavenJob.java, MavenModuleSet.java, MavenModuleSetBuild.java, SurefireArchiver.java, MavenProcessFactory.java, MavenBuilder.java, RedeployPublisher.java

Quem usou o Termo: **change** alterou os arquivos:

Functions.java, Hudson.java, Run.java, AbstractProject.java

Quem usou o Termo: **issue** alterou os arquivos:

AbstractProject.java

Quem usou o Termo: **change** alterou os arquivos:

Functions.java, Hudson.java, Run.java, AbstractProject.java

Quem usou o Termo: **files** alterou os arquivos:

FilePath.java

Quem usou o Termo: **remote-tracking** alterou os arquivos:

Quem usou o Termo: **symbol** alterou os arquivos:

VirtualFile.java, PeepholePermalink.java, MavenProcessFactory.java, FilePathRuleConfig.java, Main.java, PeepholePermalink.java, ArgumentListBuilder.java, PermalinkProjectAction.java, Kernel32.java, WinIOException.java, SocketInputStream.java, SocketOutputStream.java, Util.java, Kernel32Utils.java, RemoteInputStream.java, SecretRewriter.java, MavenProcessFactory.java, MavenProcessFactory.java, SocketInputStream.java, WinIOException.java, PeepholePermalink.java, SecretRewriter.java, MavenProcessFactory.java

Quem usou o Termo: **slave** alterou os arquivos:

Computer.java, Hudson.java, Node.java, Slave.java, index.jelly, ManagedWindowsServiceLauncher.java, AbstractBuild.java, AbstractProject.java, ComputerSet.java, SlaveStartMethod.java, SlaveComputer.java, Messages.properties

Quem usou o Termo: **api** alterou os arquivos:

Computer.java, Run.java, Api.java, _api.jelly

Quem usou o Termo: **null** alterou os arquivos:

FilePath.java, Functions.java, Run.java

Quem usou o Termo: **page** alterou os arquivos:

Functions.java, configure.jelly, layout.jelly, hudson-behavior.js

Quem usou o Termo: **tag** alterou os arquivos:

Run.java

Quem usou o Termo: **node** alterou os arquivos:

Computer.java, Hudson.java, Node.java, Slave.java

Quem usou o Termo: **throws** alterou os arquivos:

AbstractProject.java, FilePath.java

Quem usou o Termo: **pull** alterou os arquivos:

AbstractItem.java

Quem usou o Termo: **icon** alterou os arquivos:

Functions.java, style.css

Quem usou o Termo: **prepare** alterou os arquivos:

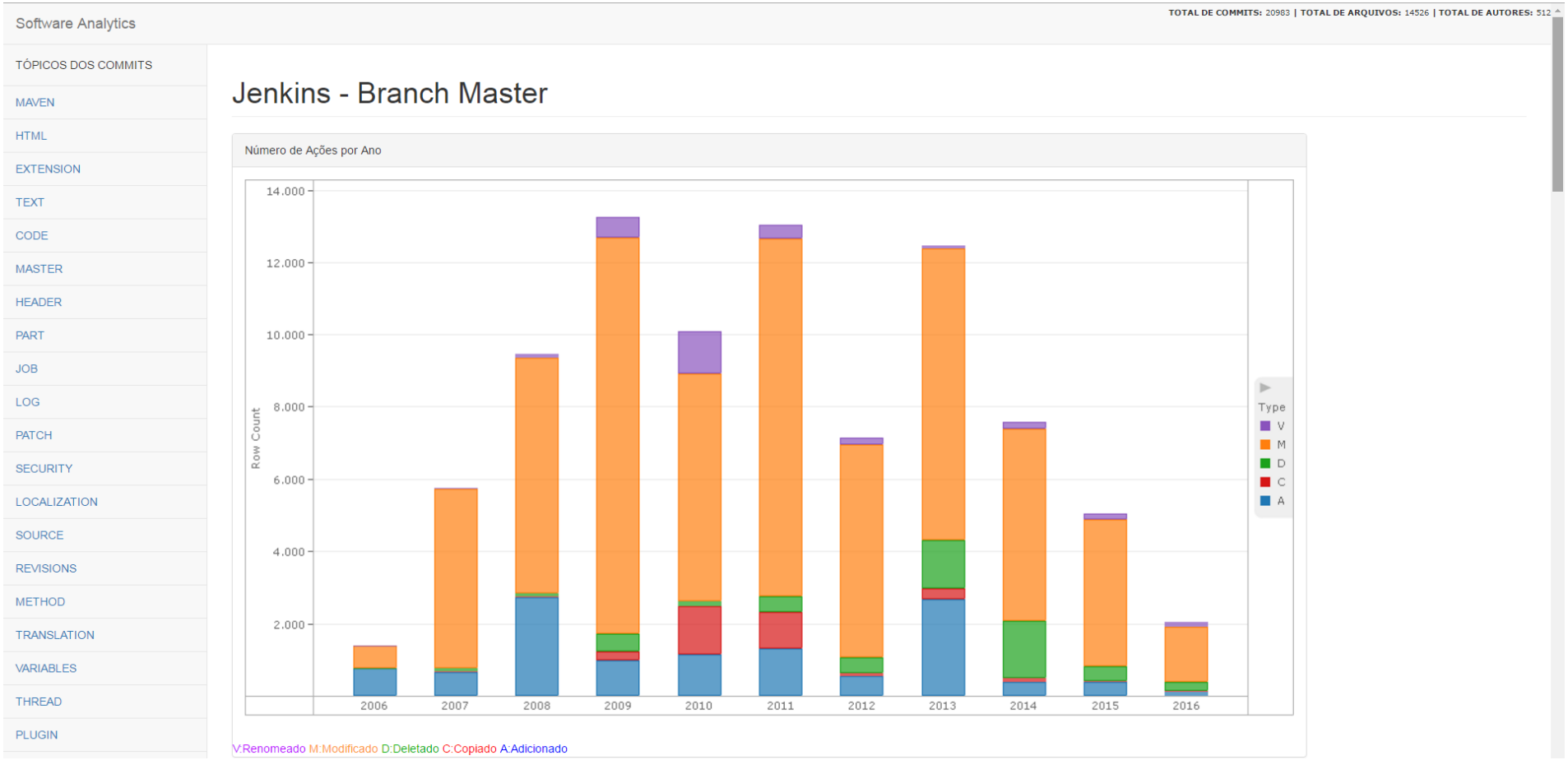
PluginManager.java, HudsonTestCase.java, hudson-behavior.js

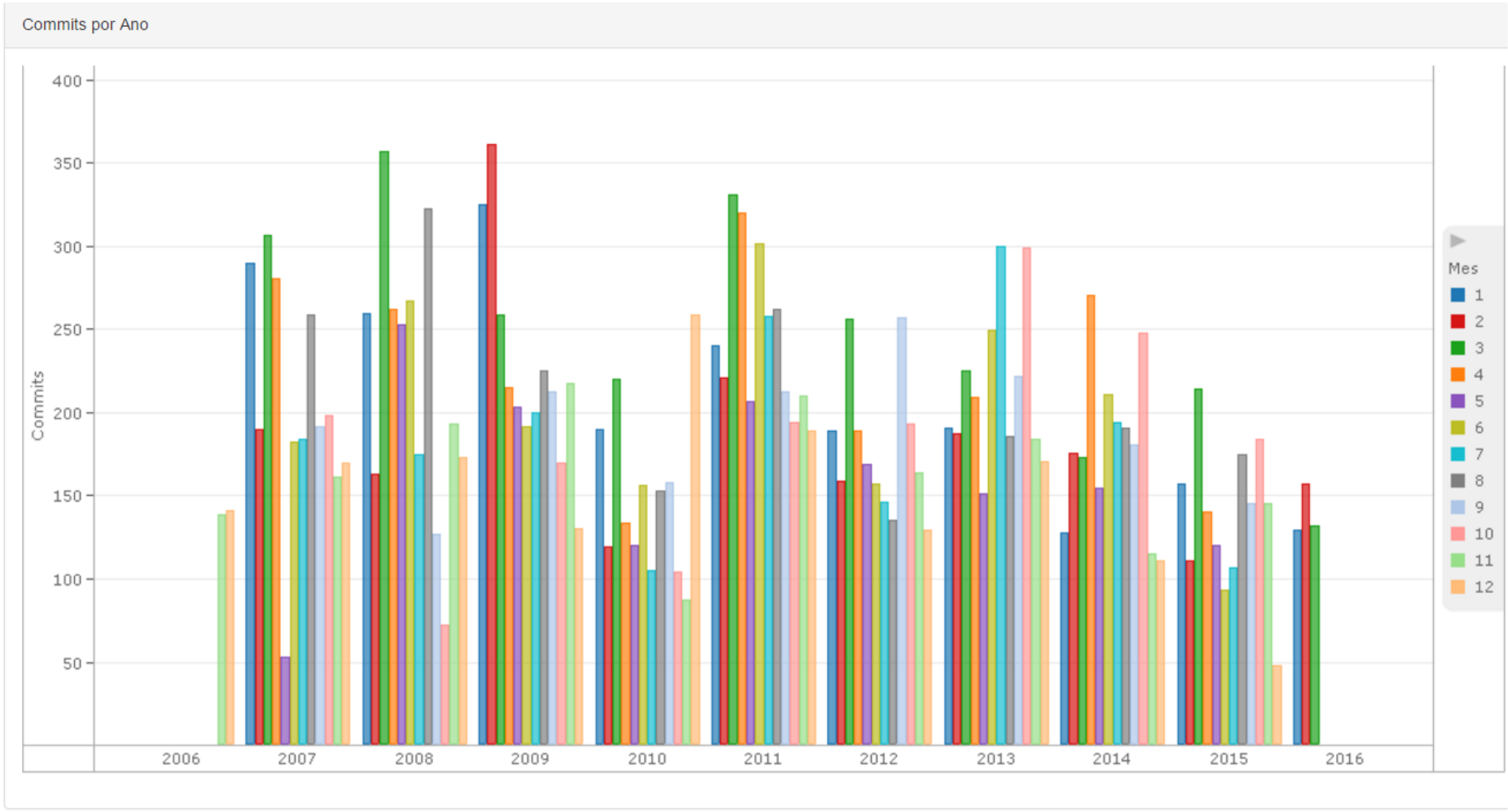
Quem usou o Termo: **commit** alterou os arquivos:

Jenkins.java

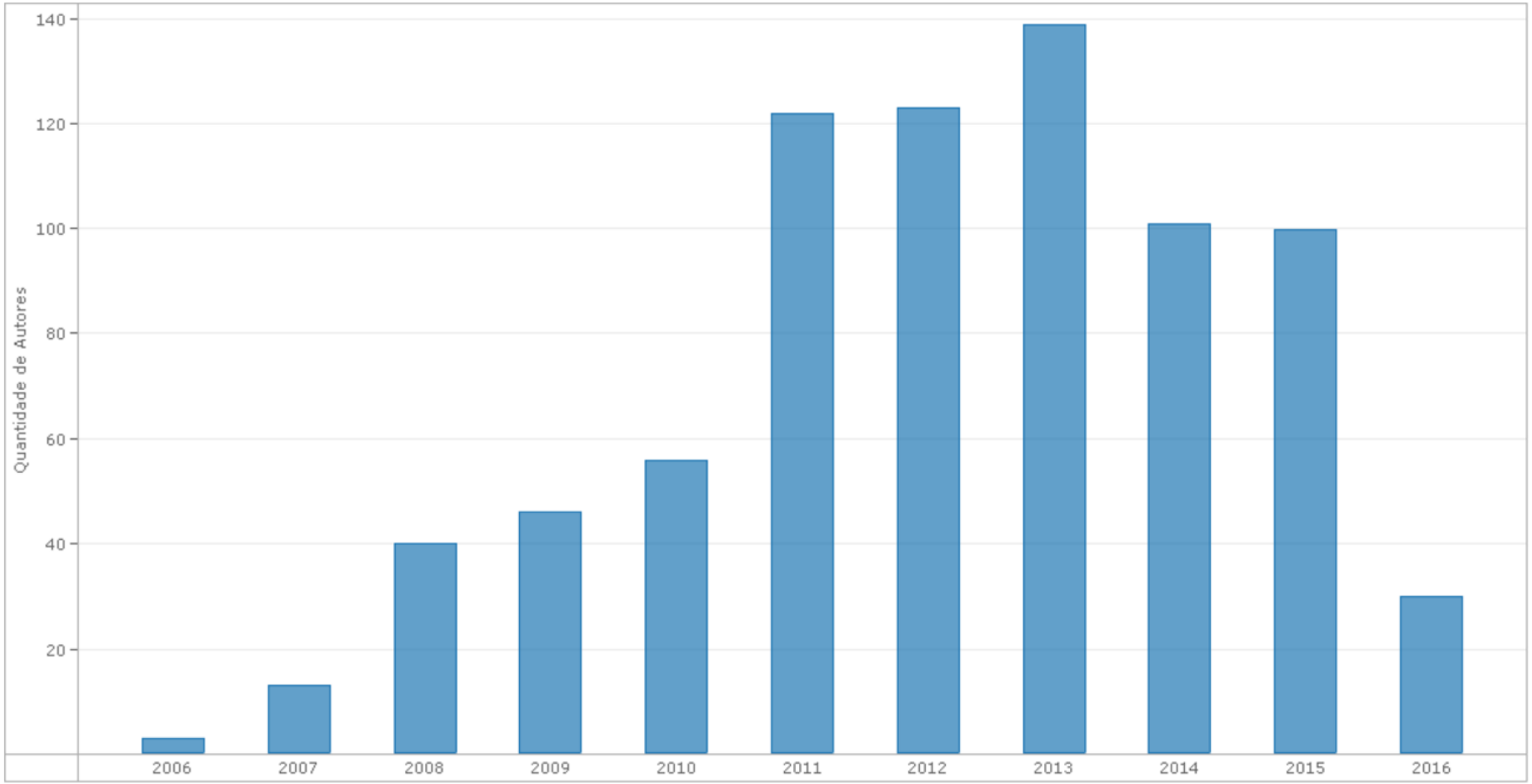
Quem usou o Termo: **line** alterou os arquivos:

AbstractProject.java

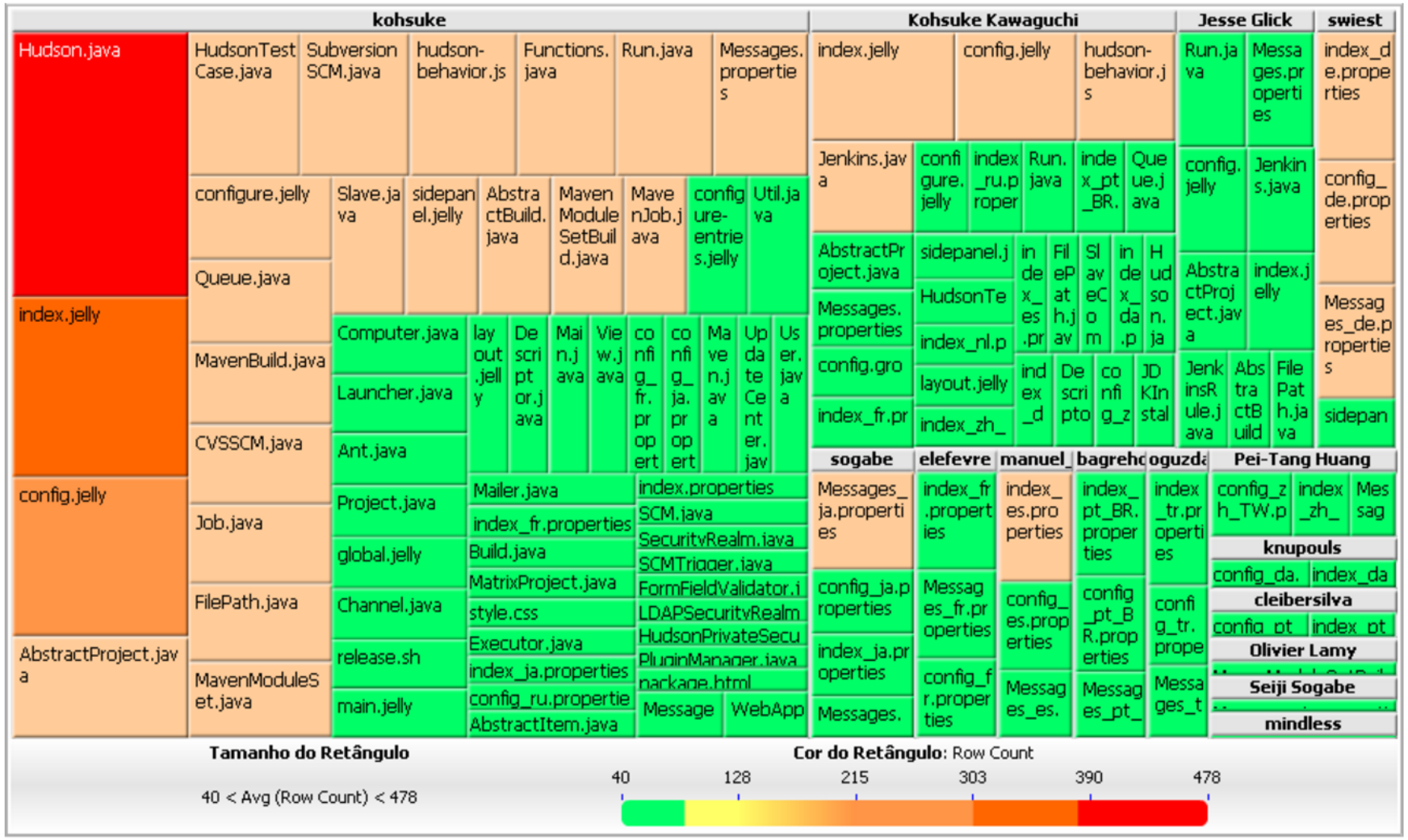

Software Analytics
TOTAL DE COMMITS: 20983 | TOTAL DE ARQUIVOS: 14526 | TOTAL DE AUTORES: 512
TÓPICOS DOS COMMITS
MAVEN
HTML
EXTENSION
TEXT
CODE
MASTER
HEADER
PART
JOB
LOG
PATCH
SECURITY
LOCALIZATION
SOURCE
REVISIONS
METHOD
TRANSLATION
VARIABLES
THREAD
PLUGIN
Jenkins - Branch Master
Número de Ações por Ano
Row Count
14.000
12.000
10.000
8.000
6.000
4.000
2.000
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
Type
V
M
D
C
A
V:Renomeado M:Modificado D:Deletado C:Copiado A:Adicionado


Commits por Ano
Commits
400
350
300
250
200
150
100
50
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
Mes
1
2
3
4
5
6
7
8
9
10
11
12

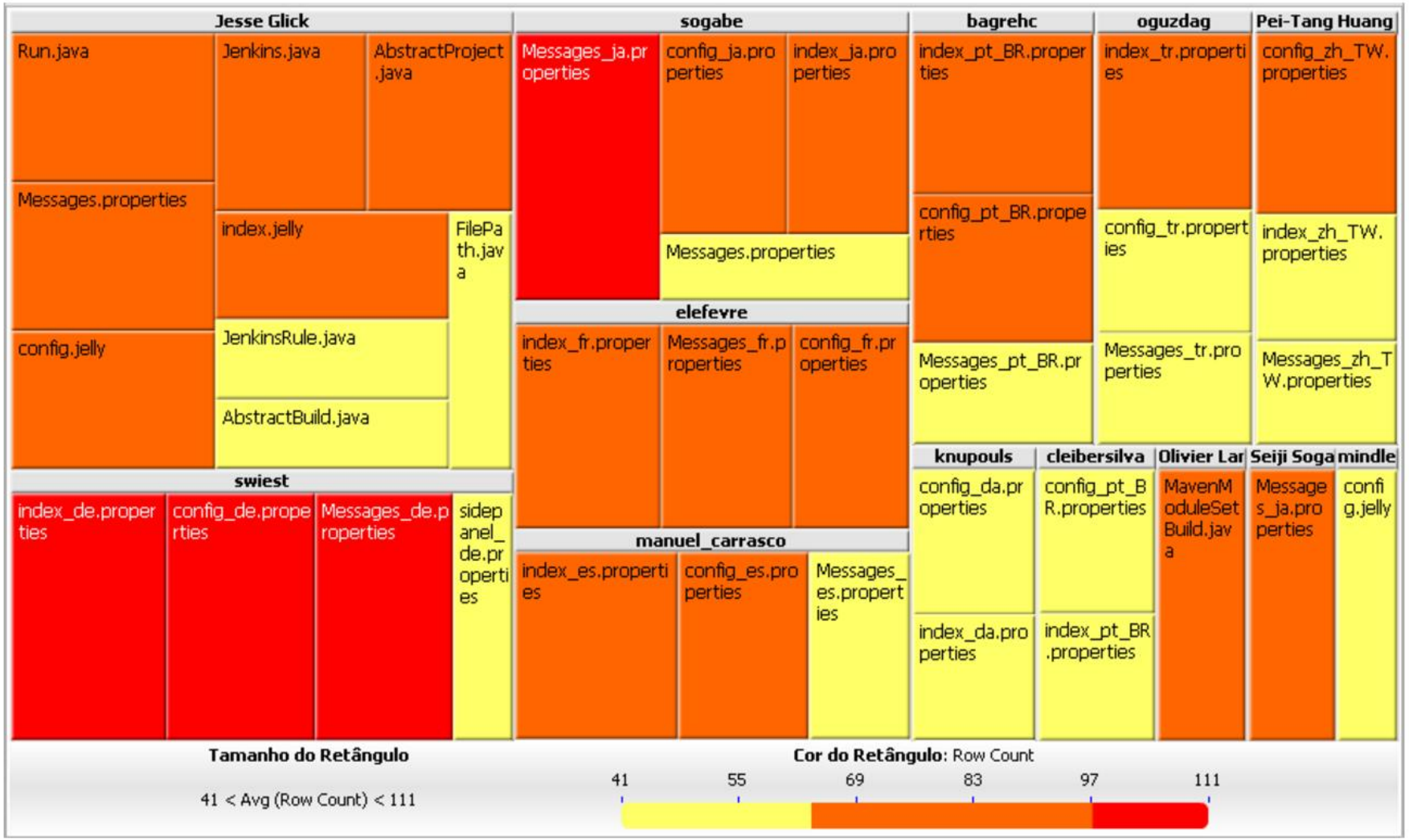
Quantidade de Autores por Ano



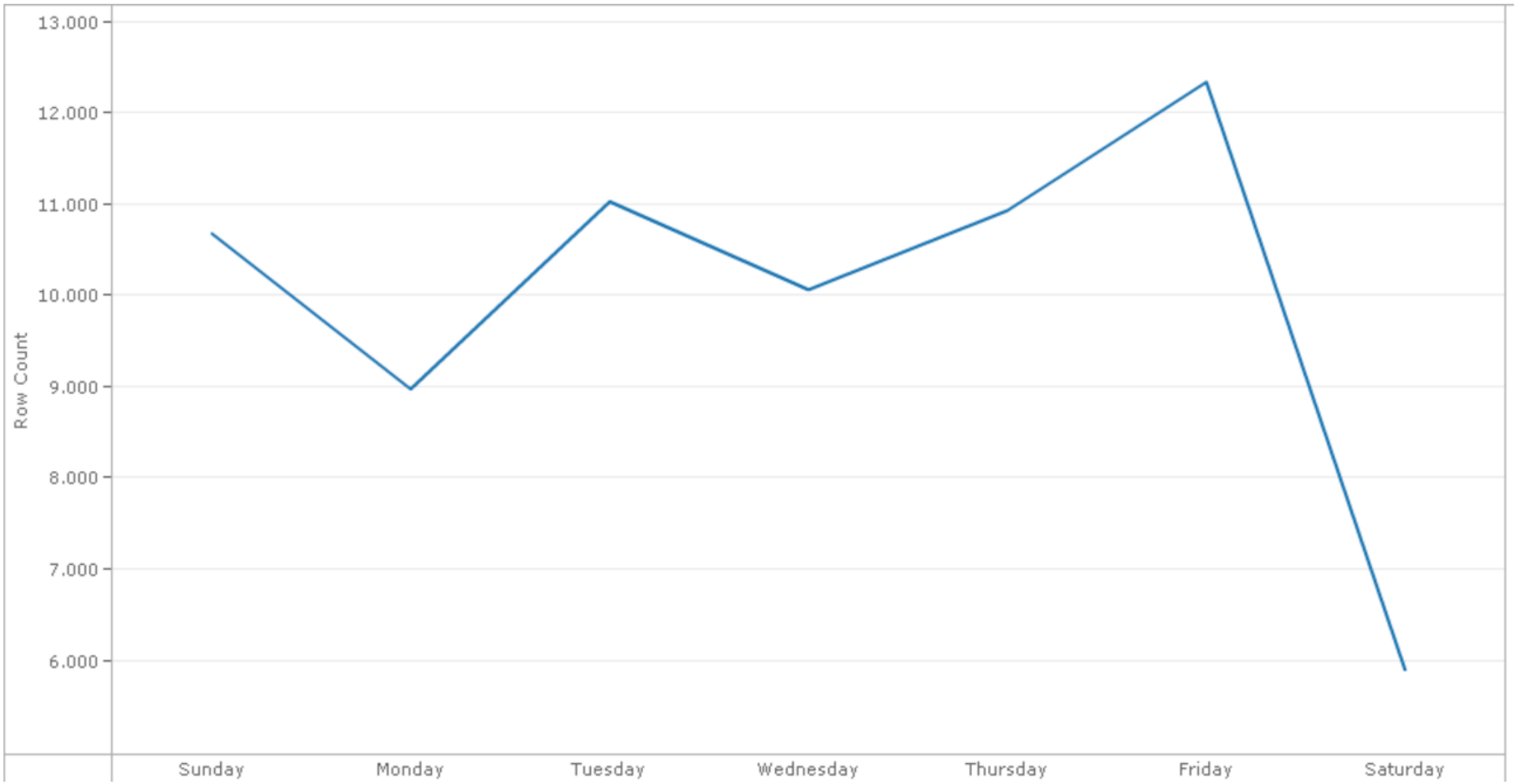
Arquivos Modificados por Autor

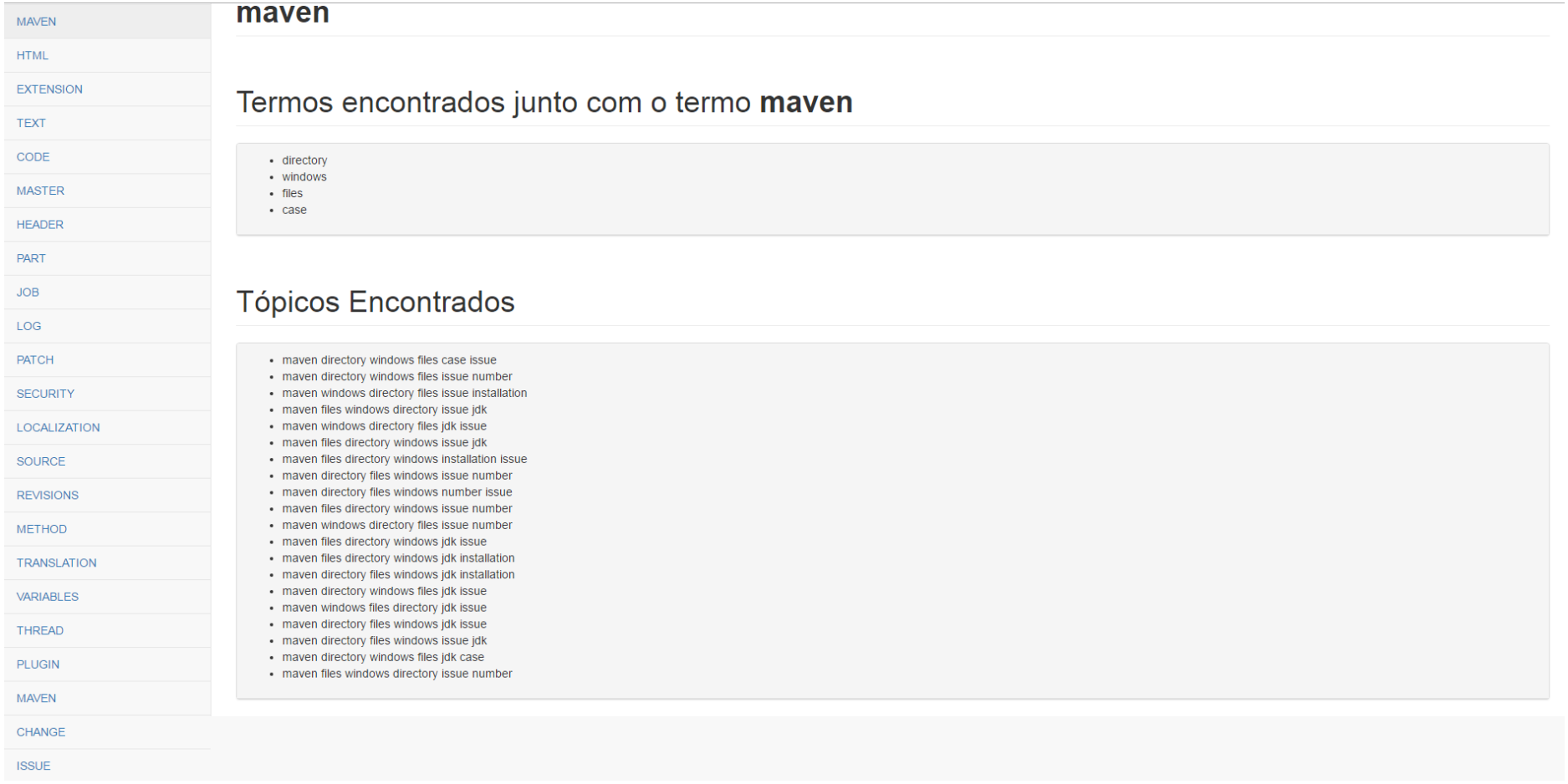


Arquivos Modificados por Autor



Commits por Dia da Semana




MAVEN
HTML
EXTENSION
TEXT
CODE
MASTER
HEADER
PART
JOB
LOG
PATCH
SECURITY
LOCALIZATION
SOURCE
REVISIONS
METHOD
TRANSLATION
VARIABLES
THREAD
PLUGIN
MAVEN
CHANGE
ISSUE
maven
Termos encontrados junto com o termo maven
directory
windows
files
case
Tópicos Encontrados
maven directory windows files case issue
maven directory windows files issue number
maven windows directory files issue installation
maven files windows directory issue jdk
maven windows directory files jdk issue
maven files directory windows issue jdk
maven files directory windows installation issue
maven directory files windows issue number
maven directory files windows number issue
maven files directory windows issue number
maven windows directory files issue number
maven files directory windows jdk issue
maven files directory windows jdk installation
maven directory files windows jdk installation
maven directory windows files jdk issue
maven windows files directory jdk issue
maven directory files windows jdk issue
maven directory files windows issue jdk
maven directory windows files jdk case
maven files windows directory issue number

html

Termos encontrados junto com o termo **html**

- href
- report
- html#a
- locked

Tópicos Encontrados

- html report href locked html#a patch
- html report href patch html#a locked
- html report href locked html#a reported
- html report href locked patch html#a
- html report href html#a patch locked
- html report href html#a locked patch
- html href report html#a patch http://n
- html report href html#a patch http://n
- html report href html#a http://n locked
- html report href html#a locked http://n
- html report href locked html#a http://n
- html href report locked html#a patch
- html href report html#a locked patch
- html href report html#a locked http://n
- html href report patch html#a locked
- html href report html#a http://n locked
- html report href html#a locked discussion

Apêndice B – Informações dos participantes do grupo focal

Participação de Grupo Focal

Este formulário é preenchido pelos participantes do grupo focal. As informações abaixo servirão de base para as análises dos dados após a reunião. É importante salientar que os nomes dos participantes não serão divulgados, se mantendo em pleno sigilo.

Nome:_____

Cargo/Função atual: _____

Idade: _____

Tempo de experiência na indústria de software: _____

Curso de Graduação:_____

Marque a opção da sua última qualificação:

( ) Graduação em curso

( ) Graduação completa

( ) Pós-Graduação Lato Sensu / MBA - **CURSANDO**

( ) Pós-Graduado Lato Sensu / MBA - **COMPLETO**

( ) Mestrado - **CURSANDO**

( ) Mestrado - **COMPLETO**

( ) Doutorado - **CURSANDO**

( ) Doutorado – **COMPLETO**

( ) Outro

Curso:_____

Certificação(ões) _____

Apêndice C – Formulário de Consentimento do Grupo Focal

UNIVERSIDADE FUMEC
DE MINAS GERAIS PARA O MUNDO

CONSENTIMENTO DE PARTICIPAÇÃO EM GRUPO FOCAL

Declaro que compreendi o propósito do grupo de discussão e a natureza das questões formuladas. Declaro ainda que concordo com a gravação do áudio desta reunião.

Declaro que concordo em relatar experiências e sugestões que possam proporcionar melhorias nos serviços e/ou recursos existentes para obtenção de dados utilizados para inteligência competitiva, reconhecendo que esta etapa é crucial para o planejamento estratégico.

Declaro estar ciente de que minha participação é voluntária. Compreendo que sou livre para deixar o grupo a qualquer momento. Compreendo que tenho o direito de não participar ou opinar em determinados momentos da discussão.

Declaro estar ciente de que as experiências e pensamentos relatados por mim durante a reunião poderão ser compartilhados com outras pessoas no âmbito da Pesquisa de Mestrado em Sistemas de Informação e Gestão do Conhecimento da Universidade FUMEC.

_____ _____

Nome do Participante Data

_____

Assinatura

Reitoria
AV. Afonso Pena, 3.880 – 4º Andar
Cruzeiro – 30.130.009
Belo Horizonte – MG
Tel. (31) 3269-5200
reitoria@fumec.br
www.fumec.br