Visualização nas Atividades de Evolução de Software no Processo de Manutenção Colaborativa

Revisão Sistemática de Literatura

Marcos Miguel
Programa de Pós Graduação em Ciência da Computação – Universidade Federal de Juiz de Fora (UFJF)
marcos.miguel@ice.ufjf.br

José Maria N. David
Programa de Pós Graduação em Ciência da Computação – Universidade Federal de Juiz de Fora (UFJF)
jose.david@ufjf.edu.br

Marco Antônio P. Araújo
Programa de Pós Graduação em Ciência da Computação – Universidade Federal de Juiz de Fora (UFJF)
maraujo@acessa.com

Jacimar Tavares
Programa de Pós Graduação em Ciência da Computação – Universidade Federal de Juiz de Fora (UFJF)
jacimar.tavares@ice.ufjf.br

Claudio Lelis
Programa de Pós Graduação em Ciência da Computação – Universidade Federal de Juiz de Fora (UFJF)
claudio.lelis@ice.ufjf.br

Abstract. *The Evolution of software has emerged as one of the major topics in software engineering and maintenance. The visualization software aims to meet the visuals necessary for a correct understanding of the software. Within this context, the relevant article proposes a systematic literature review, with the goal of identifying and classifying relevant publications on the subject, identifying its shortcomings and suggesting points of research and discussion..*

Resumo. *A Evolução de software tem se destacado como um dos principais tópicos na engenharia de software e manutenção. A visualização de software visa atender aos recursos visuais necessários para a correta compreensão do software. Dentro deste contexto, o respectivo artigo propõe uma revisão sistemática da literatura, com o objetivo de identificar e classificar publicações relevantes sobre o assunto, identificando suas deficiências e sugerindo pontos de pesquisa e discussão.*

1 – Introdução

Evolução de software tem se destacado como um dos principais tópicos na engenharia de software e manutenção. A evolução de software lida, em geral, com uma elevada quantidade de dados, que provém de fontes heterogêneas, tais como repositório de gerenciamento de Código Fonte (SCM), sistemas de *BugTraces* (BTS), entre outras fontes. Um dos aspectos-chave da evolução de software é a construção de teorias e modelos que permitem a compreensão do passado e do presente, bem como a previsão

de propriedades futuras relacionadas à atividades de manutenção de software e, portanto, apoio as tarefas de manutenção[10].

Visualização de Software é o campo da Engenharia de Software que visa ajudar as pessoas a entender o software por meio de recursos visuais [1]. Pode ser efetivamente usado para analisar e compreender a grande quantidade de dados produzidos durante a evolução do mesmo.

Devido aos relatos acima citados, diversos pesquisadores têm proposto ferramentas de visualização e evolução do software (SEV). Geralmente, essas ferramentas têm como objetivo analisar a evolução do software no que tange a um conjunto de manutenção produzidas no mesmo ao longo do tempo. Visualização de Informação e a própria colaboração são campos de pesquisa que preocupam-se com o problema de apoio ao trabalho colaborativo. Em torno da informação visual, existem muitas pesquisas a serem realizadas [9].

Diante deste cenário de constante mutação de requisitos, códigos fontes, desenvolvedores nas equipes, entre outros, é necessário um mecanismo eficiente para a visualização das mudanças, possibilitando ao desenvolvedor a tomada de decisões de forma colaborativa, baseando-se no histórico de mudanças armazenado nas bases de dados.

Várias ferramentas e técnicas foram desenvolvidas pela comunidade de pesquisa de engenharia de software, com o foco em técnicas de visualização que possibilitam a identificação de atividades humanas combinadas com artefatos de softwares [4]. Diante desta diversidade, faz-se necessário um levantamento das técnicas, modelos, metodologias, ferramentas e frameworks existentes, para uma correta compreensão da área.

O presente trabalho contém uma estudo baseado em revisão sistemática, com intuito de identificar e classificar a literatura relevante sobre o assunto, suas deficiências e sugerindo pontos de pesquisa e discussão, bem como fazer um levantamento das técnicas, modelos, metodologias, ferramentas e frameworks existentes. Como resultado final, pretende-se identificar, quais fontes de dados e ou repositórios estão sendo utilizadas para visualização colaborativa no contexto de manutenção e ou evolução colaborativa de software.

O artigo está organizado da seguinte forma: Seção 2 descreve o processo da revisão sistemática realizado neste estudo. Seção 3 apresenta as principais conclusões do estudo e analisa-as. Seção 4 considera as ameaças à validade desta pesquisa. Por fim, a Seção 5 apresenta uma conclusão do trabalho.

2. Processo de mapeamento sistemático

Kitchenham et al. [11] define Revisão Sistemática de Literatura como um método para identificar, avaliar e interpretar todos os trabalhos disponíveis e relacionadas a uma questão, área de conhecimento ou fenômeno de interesse de estudo. Aplica-se, geralmente, na busca de evidências relacionadas a algum tratamento ou tecnologia para a identificação de arestas nos trabalhos existentes, promovendo novas possibilidades de pesquisa.

O Mapeamento sistemático e suas etapas são definidas por Kitchenham et al [11] e são compostas pelas seguintes partes: **planejamento da revisão, condução da revisão e**

apresentação dos resultados. Nesta seção, são apresentadas as etapas de planejamento e condução da revisão, e na seção seguinte, são apresentados os resultados obtidos.

2.1 Planejamento

O planejamento de uma revisão sistemática incluía criação do protocolo de revisão, que deve ser definido a priori, e revisado por diferentes pesquisadores para evitar viés Kitchenham et al [11]. Esta seção está organizada conforme o protocolo utilizado.

O objetivo deste trabalho é identificar e caracterizar o corpo de conhecimento da literatura relevante e discutir o uso da visualização na evolução do software, com base na questão primária de pesquisa a seguir: *quais são as tarefas de visualização colaborativa de software que estão sendo utilizadas no contexto de manutenção ou evolução colaborativa de software*?

2.1.1 Questões de pesquisa

A partir da questão da pesquisa supramencionada, foram derivadas 5 outras questões específicas que nortearam o estudo. Estas perguntas definem as faces de classificação do estudo e abordarão temas relacionados à Visualização de software e a utilização de metodologias, ferramentas e frameworks voltados para esta área.

• *Q1: Quais técnicas de visualização colaborativa de software estão sendo utilizadas no contexto de manutenção/evolução colaborativa de software?*

Esta questão tem como objetivo identificar as técnicas atuais utilizadas para a visualização colaborativa de software e geralmente são usadas para resolver problemas específicos da área de engenharia de software. Com exemplo de técnicas, encontrada nos artigos selecionados pelo estudo, pode-se citar às Técnicas entre componentes de software criar dependências sociais [12].

Em [77][84] é apresentada uma técnica que permite a construção de mecanismos para explorar a evolução de projetos de código fonte disponíveis em bases de dados de código fonte CVS. Na publicação é apresentado o conceito de *Visual Code Navigator* (VCN), que define meios com os quais é possível navegar, de forma visual entre artefatos de código. Este conceito é utilizado na técnica apresentada em forma de visualizações. Dentre elas estão: *Syntactic View* e *Evolution View*. *Syntactic View* pemite que o usuário visualize em miniatura, um conjunto selecionado de arquivos de código fonte disponíveis em um projeto. Por exemplo, pode-se selecionar 5 classes pertencentes a um software e analisá-las, ao mesmo tempo (por ficarem pareadas e em formato reduzido) permitindo analisar parte por parte da classes (seus métodos e atributos) apenas utilizando o recurso de *zoom*. *Evolution View* fornece uma linha do tempo que permite acompanhar as mudanças sofridas em uma classe ao longo das versões de um software, historicamente. Através dela, é possível visualizar, por exemplo, como uma dada classe evoluiu, em termos de número de linhas de código, ao longo das versões do software.

Em [80] é apresentada uma técnica desenvolvida para apoiar, por exemplo em atividades de manutenção, a descoberta da rastreabilidade entre entidades de código fonte orientado a objetos, como métodos de uma classe. Neste sentido, se está interessado em analisar o grau de dependência entre as entidades objetivando aprender sobre os acoplamentos existentes. A técnica desenvolvida acompanha visualizações que permitem explorar graficamente a rastreabilidade descoberta. Um exemplo é a matrix de

visualização que exibe em forma de matrix matemática a rastreabilidade calculada para um conjunto de entidades de código.

Em [82] é apresentada uma técnica que permite a análise de *bugs* cadastrados em um sistema de *bug tracking* objetivando encontrar similaridades entre eles. Ela foi desenvolvida para ser utilizada de forma colaborativa, onde, desenvolvedores colaborando entre si serão capazes de encontrar similaridades entre bugs e assim conseguirem resolver um novo *bug*. Primeirmente, um novo *bug* é cadastrado e fica aguardando até que possa ser resolvido. Quando chega esse momento, um conjunto de desenvolvedores, através de uma ferramenta de software, irão analisar outros *bugs* que já tenham sido resolvidos anteriormente e selecionar um ou mais *bugs* que julgarem semelhantes ao *bug* a ser resolvido. Em seguida, um recurso de software é acionado tornando-se possível observar as mudanças ocorridas, a nível de código, nos *bugs* selecionados, indicando que, o *bug* a ser resolvido poderá impactar aqueles mesmos trechos de código, dado a similaridade entre eles.

Em [36] é descrita uma técnica para entender a dinâmica dos desenvolvedores e determinar quem possui propriedade do código. A propriedade de código informa a quantidade de conhecimento que cada desenvolvedor possui e indica qual desenvolvedor é dono de qual artefato de um sistema de software através da medição, de quem já acumulou mais conhecimento de cada artefato. Foi definido um refinamento na medição que conta a frequência com que cada desenvolvedor editou cada arquivo, mas considera a noção de perda de memória na definição de propriedade do código. Ou seja, um desenvolvedor que tem realizado a maioria das edições de código de um arquivo, mas não tocou nele por um longo período (quando o arquivo passou por mudanças significativas), começa a perder o conhecimento. Nesse meio tempo, o desenvolvedor que realiza mudança mais recentemente se torna mais experiente, mesmo que ele não tenha realizado tantas edições como o primeiro.

Uma técnica para quantificar e visualizar a estabilidade de código usando heatmaps, foi utilizada em [34] num estudo experimental. Heatmap visualiza as mudanças como uma tabela onde as linhas são componentes e as colunas a passagem do tempo. Em cada célula o número da célula mostra a soma das linhas de código adicionados, apagados e modificados no componente durante um período de uma semana e quanto maior o número, mais intensa a cor em uma célula. O estudo mostrado realizado em três empresas de desenvolvimento de software diferentes, mostrou que a visualização de como o código fonte muda usando heatmaps e, vincular essas visualizações aos perfis de defeito em curso fornecem indicadores de quão estável o produto em desenvolvimento é, e se os esforços de garantia da qualidade devem ser direcionadas para partes específicas do produto.

Em [42] foram aplicadas técnicas de visualização para os perfis de usuário e metadados de repositório de código-fonte do serviço de hospedagem GitHub. Três técnicas foram apresentadas e aplicadas: mapas ligados geo-dispersos, representando interações entre localidades, pequenas múltiplas telas que combinam múltiplas visões para permitir comparações visuais e diagramas de matriz que reduzem a desordem para revelar as relações de pares entre as áreas metropolitanas. O objetivo era identificar padrões que poderiam incluir o efeito da distância geográfica em relações de desenvolvimento, conectividade social e influência entre cidades, e variação de estilos de contribuição específica do projeto (por exemplo, centralizada ou distribuída). A análise utilizada examina grafos em que os nós representam as localizações geográficas

dos usuários e bordas podem representar (a) relações de seguidores, (b) commits sucessivas, ou (c) as contribuições para o mesmo projeto.

Em [29] é apresentada uma nova abordagem de análise de software para a recuperação eficiente de colaborações e papéis conceituais entre artefatos de código fonte, objetivando a compreensão desses sistemas. Durante todo o processo das interações, os módulos mostram fortes formas de cooperação: determinados módulos sempre aparecem juntos e cooperam para implementar um determinado tipo de tarefa. Já um papel conceitual é o estereótipo de comportamento de um módulo individual, onde módulo com papel de alto nível despacha tarefas para módulos com funções de nível inferior. Do ponto de vista do fluxo da informação, consumidor e fornecedor são os papéis mais comuns.

Outra técnica que segue linha semelhante, analisando as colaborações feitas entre classes por exemplo para realização de tarefas, é apresentada em [35]. Neste trabalho é proposta uma abordagem para a recuperação de colaborações que usa informação dinâmica do tempo de execução do código, mas não dependem fortemente de técnicas de visualização. Enquanto a maioria das ferramentas de visualização exibem um rastreamento completo dando ao usuário uma visão do comportamento geral de uma aplicação, a abordagem apresentada se concentra em entender pedaços muito menores de interações e os papéis que as classes desempenham nestas interações. É geral o suficiente para ser usado em aplicações de software implementados em qualquer linguagem orientada a objetos baseado em classes. A abordagem começa com a execução de um rastreamento e condensa esta informação, representando o comportamento do programa em termos de padrões de colaboração. Ele apresenta essas informações para os desenvolvedores em termos de classes emissoras, classes receptoras, métodos invocados e padrões de colaboração e permite aos desenvolvedores consultar cada um desses itens relacionando-os entre si. Desta forma, permite que um desenvolvedor foque em aspectos da aplicação sem passar por um monte de informações do rastreamento.

Outra abordagem apresentada em [30] visa atender pessoas que são especialistas em algum domínio, mas não em ciência da computação, sendo diretamente envolvidas na criação de sistemas interativos visuais dedicados a si mesmos ou a quem pertence à mesma comunidade dos utilizadores finais. A abordagem apoia o desenvolvimento de sistemas que poderiam ser mais aceitáveis pelos usuários, uma vez que eles são baseados no conhecimento (muitas vezes tácito), linguagens e notações adequadas à comunidade interessada. Parte do princípio de oferecer ao usuário aquilo que ele realmente precisa ver e lidar, que atraia e envolva-os na atividade de trabalho.

Técnicas de transformação de grafos já foram utilizados com sucesso em [56] é defendida uma abordagem evolutiva que incorpora transformações no processo de desenvolvimento normal e fornece suporte a ferramenta para monitorar o progresso em curso. Neste artigo, é discutido como os grafos temporais de consulta baseados em diagramas pode fornecer um meio natural de expressar métricas orientadas a tendência e regras de consistência identificados em estudos de caso industriais. A abordagem apresentada é baseada num subconjunto de sintaxe e semânticas da linguagem SDM (Story Driven Modeling), para transformação de grafos. Uma interpretação lógica temporal de primeira ordem de um subconjunto bem definido de grafos temporais de consultas por meio de uma gramatica. Além disso foi definida uma função de mapeamento para transformar em estados da estrutura temporal subjacente. Equipado

com esses novos conceitos, é possível especificar regras visuais de consistência em uma seqüência temporal de instancias do grafo.

Outra técnica de visualização chamada Figura Fractal [57], fornecem um meio para entender o quanto um determinado arquivo foi alterado ao longo do tempo e por quem. Essa abordagem para visualizar arquivos usando figuras fractal: (i) transmitem o esforço global de desenvolvimento; (ii) ilustram a distribuição do esforço entre os vários desenvolvedores; e (iii) permitem que arquivos sejam categorizados em termos da distribuição do esforço. É possível descobrir arquivos com alto esforço de desenvolvimento em termos de tamanho da equipe e intensidade de esforço de desenvolvedores individuais. As visualizações permitem que um analista ou um gerente de projeto obtenha as primeiras noções sobre a estrutura da equipe e princípios de propriedade de código. A Figura Fractal dá uma visão imediata de como (em termos de esforço de desenvolvimento e distribuição entre os autores) um produto foi desenvolvido. Pode-se facilmente descobrir se o desenvolvimento foi feito, principalmente, por um autor ou se muitas pessoas contribuíram para isso e em que intensidade cada contribuição foi. A figura é constituída por um conjunto de retângulos com diferentes tamanhos e cores diferentes. Cada retângulo, e, assim, cada cor, é mapeado para um autor que trabalhou no produto. A área do retângulo é proporcional à percentagem de commits realizados pelo autor ao longo de todo o conjunto de commits. Com isso foi possível descobrir 4 padrões de desenvolvimento: um desenvolvedor; apenas alguns desenvolvedores e esforço equilibrado, muitos desenvolvedores, mas esforço desequilibrado, e muitos desenvolvedores e esforço equilibrado.

O fato do código e o design “cheirarem mal” (do ingles smell), indica soluções pobres empregadas para problemas recorrentes de implementação e de design. Eles podem impedir a evolução de um sistema, tornando-se difícil para os engenheiros de software realizar mudanças. Alguns sintomas são código duplicado, métodos longos, grandes classes, e longas listas de parâmetros constituindo oportunidades de refatorações. As técnicas apresentadas em [60] focam este problema. Em primeiro lugar, é proposto o método DECOR (DEtection & CORrection), um método que descreve todos os passos necessários para a especificação e detecção de “smells” no código e design. Em segundo lugar, a técnica de detecção chamado de DETection EXpert (DETEX). DETEX permite aos engenheiros de software especificar “smells” num alto nível de abstração usando um vocabulário unificado e linguagem específica de domínio, obtido a partir de uma análise de domínio em profundidade, e para gerar automaticamente algoritmos de detecção. Assim, DECOR representa um método concreto e genérico para a detecção de “smells” e DETEX é uma instância ou uma implementação concreta deste método na forma de uma técnica de detecção.

É descrita em [61] uma junção da técnica de “Slicing Program” para extrair atributos baseados na estrutura do programa e técnicas de renderização 3D baseadas em Metaball para ajudar a análise da estrutura de código-fonte baseada em visualização. Program Slicing (do inglês "corte do programa") é uma técnica de decomposição bem conhecido que transforma um grande programa em um menor que contém apenas instruções relevantes para o cálculo de uma função de programa selecionado (saída). Esta técnica em combinação com métricas e visualizações de analises parciais permite concentrar o cálculo de métricas apenas nas partes do software de interesse imediato. Metaballs, uma técnica de modelagem em 3D foi a base para o sistema 3D de visualização de software desenvolvido em Java, chamado MetaViz. A partir disso criaram gráficos 3D que são facilmente compreensíveis e comunicam informações sobre

a complexidade relativa a um componente e o acoplamento entre os componentes e, portanto, melhora a compreensão da estrutura do programa.

Em [62] é apresentada uma técnica de visualização, um Wordle, que permite compreender rapidamente o nível de cooperação da equipe num projeto. Cada palavra é o nome de uma classe, o esforço total gasto pela equipe sobre a classe é usada como métrica para o tamanho da letra, e o percentual da equipe de trabalho sobre a classe determina a cor da palavra. Técnicas de "Pre-Attentive Processing" foram usadas para projetar a Wordle, de modo que o observador pode descobrir rapidamente as classes com baixa cooperação. Essa técnica consiste em destacar o dado que é importante para o usuário com a premissa de que assim o dado fica mais fácil de ser localizado. Na base do Wordle, podemos obter uma interpretação das atividades de desenvolvimento. Em particular, descrevemos quatro casos possíveis. O acompanhamento periódico dessa visualização (por exemplo, a cada duas semanas) pode ajudar a equipe na mitigação da perda de conhecimento e o abrandamento devido ao "turnover". Encontrar essas classes que necessitam de alto esforço de uma pequena parte da equipe, de fato, permite tomar ações corretivas, como a re-alocação de recursos.

Outro trabalho [65] utiliza a técnica de análise de redes sociais para compreender a evolução. Uma rede social representa uma estrutura social e padrões estruturais basilares que podem ser usados para analisar e compreender como as pessoas se relacionam e seu comportamento como um grupo. Desenvolver software é fundamentalmente uma atividade humana. Os desenvolvedores cooperam e trocam conhecimento e informação criando na verdade uma forma particular de rede social que é chamada de rede de conhecimento. Pela análise estrutural da comunicação e padrões de coordenação é possível identificar a rede de conhecimento, potenciais gargalos de comunicação, e pontos de coordenação da equipe. Essa abordagem permite ao arquiteto de software analisar e manter a organização e a arquitetura do software alinhados durante a evolução do mesmo. A abordagem SNA foi originalmente proposta por sociólogos e tem como objetivo principal, detectar e interpretar padrões de laços sociais entre os envolvidos. Ao aplicar a analise SNA no contexto de desenvolvimento de software, os vértices são representados pelos desenvolvedores e seus relacionamentos sociais podem ser extraídos usando vários meios incluindo comunicações mediadas por computados usando questionários ou entrevistas e vários tipos de relação podem ser usados e analisados como: confiança, conhecimento de um domínio, até mesmo a distribuição geográfica da equipe.

A técnica Cognitive Assignment Technique [66] é uma abordagem baseada em recuperação da informação para ajudar na compreensão de software através de extração de dados de documentos que não constituem o código fonte do sistema. Estes artefatos podem ser documento de requisitos e de projeto além de banco de dados de sistemas gerenciadores de mudança, fóruns online e email por ter se tornado o meio principal para troca de preocupações e opiniões.

Em [68] é proposta uma técnica de análise que combina informações de execuções dinâmicas com informações numa estruturação hierárquica das unidades de implementação. Com múltiplas visualizações integradas os desenvolvedores podem explorar como o fluxo de controle passa pelas unidades de implementação durante a execução de uma função. As visões focam em diferentes aspectos da execução: (i) uma visão geral focada no tempo permite a detecção de diferentes fases durante a execução, (ii) uma visualização detalhada focada no tempo permite compreender sequências particulares de interação entre funções, (iii) uma visão focada em colaboração permite

entender aspectos estruturais da interação de funções e (iv) uma visão focada no código que permite ao desenvolvedor correlacionar as descobertas feitas nas visualizações gráficas com as linhas de código.

Outra técnica é apresentada em [69] trata-se de uma nova abordagem para a análise de banco de dados de defeitos através de duas visualizações, chamadas System Radiography e Bug Watch. Neste trabalho os defeitos são considerados como entidades que podem mudar e evoluir com o tempo, em particular usa-se o conceito de ciclo de vida do defeito que compreende o histórico do defeito e os estados nos quais ele pode atravessar. A partir da analise dos dados do Bugzilla do sistema Mozilla, as visualizações foram propostas: O System Radiography renderiza as informações de defeito no nível de sistema e indica quais partes são afetadas por quais defeitos, em que ponto no tempo, sendo um indicador, em alto nível, da saúde do sistema e serve de base para atividades de engenharia reversa, além de destacar partes críticas do sistema como componentes mais afetados por defeitos. E o Bug Watch suporta a análise de defeitos que afetam parte do sistema, como um componente ou alguns deles. Essa visualização facilita a caracterização dos *bugs* além da identificação dos mais críticos.

É apresentada em [71] uma abordagem para revelar quais features e como elas são afetadas por mudanças no código. Features encapsulam o conhecimento de domínio de um sistema de software e, portanto, são valiosas fontes de informação para a engenharia reversa. Ao analisar a evolução de um sistema, é preciso saber como e quais recursos foram modificadas para recuperar tanto a intenção da mudança quanto a extensão, ou seja, que artefatos fonte foram afetados. A implementação de uma feature passa por um número de artefatos de fonte. Para mapear as features com os artefatos de fonte é preciso exercitar as features e capturar os rastros da execução ou seja por onde a execução passou. No entanto isso resulta em longos rastros de execução que são difíceis de interpretar. Para lidar com este problema os rastros foram compactados em conjuntos simples de artefatos de fonte que participam no comportamento em tempo de execução de uma feature. Esses rastros compactados foram referenciados como visões de features. Com isso, é possível analisar as features ao longo de várias versões de um sistema e revelar como e quais features foram afetadas por alterações no código.

• *Q2: Quais metodologias de visualização colaborativa de software estão sendo utilizadas no contexto de manutenção/evolução colaborativa de software?*

Esta questão tem como objetivo identificar as metodologias atuais utilizadas para a visualização colaborativa de software e geralmente são usadas para resolver problemas específicos da área de engenharia de software. Exemplos identificados no estudo são a Análise dinâmica e análise de rede de sistemas de software de grande escala, com o objetivo de determinar os métodos de classes que são funcionalmente importante no que diz respeito a uma determinada funcionalidade do software [13].

Em [83] assume-se que código fonte versionado em sistemas de controle de versão como SVN e CVS podem dar algum tipo de indício sobre a evolução de software, pois armazenam alterações realizadas durante todo o ciclo de vida de um software. Com base nisto, foi desenvolvido um método, chamado *Code Flows*, para auxiliar na tarefa de analisar a evolução de um software em um contexto onde são observadas mudanças a que um projeto de código foi submetido ao longo dos anos.

Code Flows oferece um conjunto de visualizações criadas para explorar os dados sobre a evolução de software que foram processados. Em uma delas é possível analisar o contexto histórico de um bloco de código (seja ele um método, uma classe, um conjunto de linhas ou uma única linha de código). Na prática, isto quer dizer que *Code Flows* é capaz de indicar que um bloco de código que estava em uma classe A na versão X de um software, agora, na versão X + 1 encontra-se na classe B ou em outro ponto dentro da mesma classe A. Em outra visualização fornecida é possível rastrear o mesmo bloco de código e seu destino final, mas com uma representação visual em formato de árvore, onde, por exemplo, um método A1 que pertencia a classe A (classe A então é representada como sendo um nó pai na árvore e o método A1 como sendo um nó filho de A) em uma outra versão do software analisado passou a pertencer a classe B (portanto agora A1 é nó filho de B).

É apresentado em [74] uma abordagem chamada History Slicing que permite aos desenvolvedores gerar um histórico "fatiado" de mudanças ocorrido em qualquer trecho de código fonte (contíguo ou fragmentado, e em um único arquivo ou através de vários). Está fundamentada na ideia de que nem todas as revisões possuem alterações em linhas de interesse para o desenvolvedor que ele deseje analisar, ou mesmo alterações em arquivos de interesse, e assim a abordagem consumiria menos tempo de análise. Chronos, é o protótipo da implementação da abordagem History Slicing que inclui um modelo de geração, análise e uma interface de usuário que provê visualização de qualquer "fatia" do histórico que pode contar com vários arquivos ou vários revisões. A visualização ainda apoia os desenvolvedores a reconhecer visualmente padrões na evolução do código.

• *Q3: Quais ferramentas estão sendo utilizadas na visualização colaborativa de software no contexto de manutenção/evolução colaborativa?*

Esta questão tem como objetivo identificar as ferramentas que são frequentemente utilizadas para a visualização colaborativa de software e geralmente são usadas para resolver problemas específicos da área de engenharia de software. Exemplos identificados no estudo são: Syde, ferramenta para restabelecer conscientização da equipe, compartilhando mudanças e conflitos de informações entre os espaços de trabalho do desenvolvedor [14], Integrare, uma ferramenta com funções de capturar, visualizar e propagar a evolução de um ambiente de desenvolvimento colaborativo [15].

Em [18] é apresentada a ferramenta *Softwarenaut*, que atua na recuperação de arquiteturas de software com base na análise de projetos de código. A ideia central é mostrar, através de visualizações específicas (como *treemaps*, grafos e estruturas de árvores) as relações existentes entre as entidades presentes no software, isto é, a relação entre módulos, classes e métodos.

Em [79] é apresentado *SourceVis*, uma ferramenta que fornece visualizações sobre a evolução de software com base a análise de código fonte. O objetivo é fornecer várias visualizações colaborativas para apoiar o trabalho de equipes distribuídas nas tarefas de entendimento sobre a evolução do software. Tais tarefas podem ser de manutenção também, mas não é o foco exclusivo e nem principal de *SourceVis* [89].

Entre as informações que podem ser visualizadas estão métricas de código como numero de acoplamentos, linhas de código, número de classes, dentro outras.

Em [76] é apresentado *Storygraph*, uma ferramenta visual que analisa dados históricos de software, como *logs* provenientes de repositórios de código fonte, objetivando fornecer visualmente informações que podem ser usadas na gerência de projetos desenvolvidos colaborativamente. Ela fornece visualizações onde é possível que a gerência do projeto acompanhe quem mais está colaborando na equipe geograficamente distribuída, pois ela cria um gráfico com as coordenadas geográficas de cada membro da equipe e exibe em um mapa de quais regiões vem a maior parte da colaboração.

Em [31][33][3] é apresentada a ferramenta Replay que mostra as mudanças feitas no código fonte em projetos com multi-desenvolvedores. Analisa dados de um repositório próprio desenvolvido em um trabalho anterior do mesmo autor, chamado Syde. onde os desenvolvedores podem optar por ver as alterações feitas por um conjunto de desenvolvedores em um conjunto de artefatos (Unidade de compilação, pacote ou classe ou método, lembrando que estas são informações oferecidas por Syde) durante um determinado período de tempo, preservando a ordem em que foram realizadas. Não há suporte a outros repositórios de código fonte. Permite que os desenvolvedores investiguem como as mudanças foram realizadas no passado ao filtrá-los de acordo com três critérios: desenvolvedor, os artefatos e tempo.

O Small Project Observatory (SPO) é a ferramenta apresentada em [8][43] que incorpora uma abordagem para analisar ecossistemas de software. O primeiro objetivo ao construir SPO foi fornecer uma interface interativa para a visualização e exploração dos ecossistemas de software. Implementado como uma aplicação online, provê uma linha do tempo do projeto que sumariza os dados analisados baseado em métricas como número de classes e número de *commits*, além disso é possível observar as dependências entre projetos do ecossistema através do Mapa de Dependências de Projeto. É ainda possível através do Mapa de Vocabulário de Projeto ver as expressões mais relevantes usados no código fonte do projeto. E ainda um Mapa de Colaboração entre Desenvolvedores relacionando os desenvolvedores que mais colaboraram entre si entre projetos. Atualmente, o único tipo de repositório apoiado é o chamado Store (Smalltalk Open Repository Environment), escolhido pois mantém o controle de uma rica paleta de informações sobre as versões de projetos de TI. Na implementação atual, importar os dados sobre algumas centenas de projetos pode ser feito em questão de minutos.

Em [52][54] é apresentada uma ferramenta que oferece uma representação 2D chamada Revision Tree com o objetivo de apoiar a consciêntização de desenvolvedores de software e gerentes de projeto no que concerne à evolução dos itens de software por meio da visualização de colaboração entre os desenvolvedores. Esta representação é dada por uma ferramenta para visualizar as contribuições dos membros da equipe e fornece informações sobre o que está acontecendo em um projeto de software, quem está trabalhando no projeto, o que os desenvolvedores estão fazendo, quanto tempo vem trabalhando em uma revisão, como o seu trabalho pode ter impacto no trabalho dos outros e se os desenvolvedores colaboram entre eles ou se trabalham individualmente. A proposta oferece uma estrutura de grade que todos os programadores estão familiarizados, uma linha do tempo para orientar e situar os usuários no tempo, um

painel de controle para filtragem de datas, a exibição de detalhes adicionais e um painel de zoom; e várias possibilidades de interação para tornar as informações disponíveis para satisfazer as necessidades do usuário. Devido a esta representação um usuário pode obter muitas respostas sobre como a evolução do item está acontecendo enquanto a equipe está sempre ciente de quem está trabalhando em baselines e revisões. A representação de linha do tempo, mostra o intervalo de tempo completo, desde que o item foi criado. Ele suporta comparações temporais e torna evidente quando os programadores trabalham de forma simultânea.

Na mesma linha de conscientização da evolução e colaboração na equipe está a ferramenta apresentada em [55] chamada Chronia que implementa a visualização *Ownership Map* para entender quando e como os diferentes desenvolvedores interagem, de que forma e em que parte do sistema. A visualização é baseada numa abordagem [72] que parte do princípio de que o desenvolvedor original de uma linha de código, ou seja, o proprietário, é o mais experiente sobre aquela linha. O proprietário de um pedaço de código é determinado como sendo o desenvolvedor que é proprietário da maior parte desse pedaço de código. Esse conceito de propriedade é usado para proporcionar uma visualização que ajuda a entender como os desenvolvedores interagem com o sistema. A abordagem baseia-se apenas em informações de log do CVS sem a necessidade de verificar todo o repositório. A visualização Ownership Map utiliza a junção de linhas retas, círculos sobre as linhas cores e tamanhos onde cada linha representa o histórico de um arquivo, e cada círculo em uma linha como uma mudança nesse arquivo. A cor do círculo indica o autor que efetuou a alteração. O tamanho do círculo reflete o tamanho da mudança, e a cor da linha indica o autor que é proprietário da maior parte das linhas de código do arquivo nesse período. É possível ver os arquivos agrupados com base em seu histórico de alterações, segundo a ideia de que arquivos “commitados” no mesmo período estão relacionados. Por ser uma ferramenta interativa, com o passar do mouse a visualização revela vários padrões de comportamento do desenvolvedor, como: Monólogo (um autor trabalhando sozinho), o Diálogo (dois autores trabalhando juntos), Takeover (um autor tomando as atividades de outro) por exemplo, isso é visível com a mudança de cor da linha do histórico de um arquivo.

É apresentada em [27] CodeMetropolis uma ferramenta colaborativa para desenvolvedores baseada no Minecraft. Trata-se de um mundo virtual de código-fonte em que os desenvolvedores e outros interessados poderiam explorar e avaliar o seu projeto de forma colaborativa em um mundo Minecraft virtual. Propriedades do código são representados por gráficos primitivos oferecidos pelo motor de jogo, e vários recursos de interatividade são planejadas, por exemplo um prédio é uma classe e os andares são métodos. CodeMetropolis é uma ferramenta de linha de comando escrito em C # e utiliza o Substrate uma biblioteca para .NET Framework. Utiliza o gráfico de saída da ferramenta Columbus e cria um mundo Minecraft a partir dele. Columbus são uma coleção de vários programas, que são capazes de analisar e medir artefatos estáticos relacionados com o código-fonte.

Em [25] é apresentado CoRSA um aplicativo de modelagem UML multi-usuário, capaz de oferecer suporte à modelagem UML colaborativa em tempo real.

Em outro artigo [39], a partir de uma abordagem para reunir e integrar a informação dinâmica do Eclipse foi implementado Senseo, uma ferramenta plugin do Eclipse que permite aos desenvolvedores analisar dinamicamente aplicativos Java com o objetivo de

melhor apoiar as atividades típicas de manutenção de software. Senseo enriquece as exibições de código fonte do Eclipse com vários tipos de informações dinâmicas, como apresentação de quais métodos outro método particular invoca e a freqüência em tempo de execução, e quantos objetos ou a quantidade de memória é alocada em métodos particulares. Senseo possui uma visualização sobre as colaborações dinâmicas entre diferentes artefatos de origem, o que ilustra a comunicação entre pacotes, entre classes e entre métodos. A informação recolhida é armazenada ao longo de várias execuções do sistema.

Em [37] é apresentado SECONDA, trata-se de um ecossistema de software, onde um painel de visualização e análise oferece tanto análise individual e agrupada da evolução dos projetos e desenvolvedores que pertencem ao ecossistema de software, em granularidade grossa e fina. Ecossistemas de software são coleções coerentes de projetos de software que evoluem juntos e são mantidos pela mesma comunidade de desenvolvedores. Exibem alguma caracteristica particular de evolução, por causa das dependências entre os projetos e as interações entre os membros da comunidade, aspectos que são levandos em consideração em ferramentas como SECONDA. A relação entre casos de teste e o sistema podem ajudar na manutenção.

Em [41] MAVIS, uma ferramenta é apresentada para visualização de defeitos baseada em caracteristicas (features), que fornece uma visão combinada dos dados de teste e o sistema em teste. Esta visão proporciona aos testadores uma melhor compreensão de defeitos em todo o sistema, e a localização do recurso com problema rapidamente. Em Mavis, também é implementada uma funcionalidade de gestão de versão para caracterizar a evolução do sistema e os testes, combinada a uma interface visual interativa.

Classroom BRIDGE [28] é uma ferramenta de apoio a conscientização sobre as atividades realizadas em um grupo, ao facilitar o planejamento e a revisão dos objetivos de forma colaborativa para apoiar projetos de grupos distribuídos. Para apoiar o planejamento e compartilhamento de agendas, objetos de calendário estão disponíveis, e a interface modelada através de ferramentas típicas de calendário, permite a criação de eventos programados. Um sistema de linha do tempo permite identificar as tarefas realizadas e os autores. O software Classroom BRIDGE usa a replicação de objeto para apoiar o trabalho distribuído síncrono e assíncrono. mantendo cópias somente leitura, em servidores, dos objetos que estão sendo trabalhados. Foi aplicado ao contexto de salas de aula, em projetos de alunos pertencentes a salas diferentes.

VLEPPO [40] é um sistema integrado de modelagem visual e solução de problemas de planejamento que visa: oferecer uma interface gráfica conveniente, o que simplifica a modelagem, facilita a manutenção e promove a compreensão de planejamento de domínios até mesmo para usuários não-especialistas, em conformidade com as normas PDDL em vigor para a representação de domínios e problemas facilitando assim a comunicação com outros sistemas em conformidade com esta norma. Sua implementação é em Java para apoiar portabilidade e interoperabilidade. Como os algoritmos de planejamento podem estar implementados localmente ou como serviços web, o sistema oferece um módulo cliente para serviços web onde o usuário pode experimentar diferentes algoritmos para resolver os problemas de planejamento.

Code Bubbles [46] é uma tentativa de redefinir a interface do usuário para um ambiente de programação integrado. A visualização da interface de usuário seria através

de “bolhas” onde cada uma apresentaria informações distintas por exemplo, documentação, código fonte, casos de teste, modelos, *tickets* de solicitação de mudanças, de modo que o desenvolvedor poderia editar e manipular um conjunto de trabalho completo. Pode ser usado como um plugin do ambiente Eclipse ou de forma independente, desenvolvido em Python, como um conjunto de plugins. Suporta interação com outros plugins. Possui uma versão para estudantes. As versões estão disponíveis na web para download.

Em [44] são apresentados dois protótipos visuais desenvolvidos, que tornam visível o histórico de uma requisição de mudança (RM), ao expor detalhes de rastreio, ou através da agregação de diferentes tipos de dados entre requisições de mudança. Um protótipo, histórico de item de trabalho, mostra as mudanças de estado durante o ano para cada bug, e o segundo protótipo, fornece uma visão contínua, com zoom, de todos os RMs abertas no conjunto de dados. Os RMs são mostrados como círculos e podem ser por exemplo agrupados por responsável.

O sistema proposto em [47] visualização de apoio a tarefa de monitorar informações de atualização de bugs. O crescimento da web possibilitou o surgimento de varios tipos de Fluxo de dados de texto, como os blogs BBS e servicos de rede social. As informações de atualização de bugs gerenciados por sistemas BTS é um tipo de fluxo de dados de texto, este fluxo gera dados constantemente sendo dificil observa-los a toda hora. Assim, o sistema implementa algumas tecnicas de visualização como a animação para visualizar o relacionamento dinâmico entre bugs e o destaque para os dados de atualização possibilitando ao usuário repassar os últimos momentos de monitoramento ajudando o usuário a não perder o contexto. O sistema proposto consiste em um módulo de coleta de dados, um módulo de gerenciamento de dados e uma interface de monitoramneto. O módulo de coleta de dados recebe constantemente emails sobre informações atualizadas de bugs através de múltiplos BTSs e extrai a informação necessária do conteúdo do email recebido. Os dados extraídos são enviados para o módulo de gerenciamento que organiza e armazena informação atualizada do bug. A interface de monitoramento recebe os dados a serem visualizados por meio do módulo de gerenciamento. Os módulos de coleta e gerenciamento foram implementados com Python. A interface foi implementada como uma aplicação web usando HTML, CSS, Javascript, Processing.

Wolf, uma ferramenta apresentada em [48] para apoiar as atividades de análise de impacto, tanto nos aspectos organizacionais e individuais dessa atividade. É especialmente útil em projetos de desenvolvimento distribuídos porque suporta a identificação de desenvolvedores impactados por uma mudança e, ao fazê-lo, facilita a comunicação e coordenação entre as partes interessadas envolvidas na mudança. Ao apoiar a criação semi-automática de links de rastreabilidade, oferece suporte para análise de impacto, tanto do ponto de vista organizacional e individual. Isso significa que os gestores e engenheiros de software podem se beneficiar do uso de Wolf, que gera vínculos de rastreabilidade através de um analisador da documentação do projeto, que deve ser armazenado em um repositório Subversion. Os itens de rastreabilidade e as ligações são, então, armazenados em um banco de dados, e a informação é usada para apoiar as atividades de análise de impacto. A tela principal da ferramenta apresenta os principais comandos da interface, uma área onde o usuário pode selecionar os itens de projeto que seriam modificados e onde ele vê o impacto da mudança através de uma estrutura de árvore, ao lado é apresentado um grafo de dependências com itens do

projeto. è possível ainda mostrar informações adicionais sobre os itens ou autores sempre que o usuário faz uma seleção no grafo de dependência. A ferramenta não se destina a fornecer valores para o custo, hora e esforço, uma vez que dependem do projeto e são difíceis de prever automaticamente. A estratégia de Wolf é oferecer informações úteis para o usuário (gerente ou engenheiro de software) e deixando este usuário usar seu conhecimento para definir a forma de calcular o real impacto da mudança proposta.

Em [51] é apresentado um kit de ferramentas de visualização para TinyOS 2.0 para auxiliar na compreensão do programa e a colaboração entre participantes. Três modos de operação são suportados. O kit é utilizado para gerar um gráfico de chamadas de sistema, que corresponde a uma base de origem do usuário, e diagramas de sequência UML correspondentes ao funcionamento particular do sistema. Os últimos diagramas vêm em duas variantes: O primeiro segue as convenções padrão da UML e retrata o comportamento de tempo de execução local. O segundo introduz várias melhorias para capturar o comportamento de tempo de execução global com participantes distribuídos de uma maneira que evidencia eventos causalmente ligados dentro de uma rede e demarca regiões de atividade logicamente concomitante.

É apresentado e avaliado em [58] um ambiente de visualização / simulação de algoritmos distribuídos chamado LYDIAN que ajuda a estudar algoritmos e protocolos distribuídos.

Em [63] é apresentada Bugarium, uma ferramenta que implementa um controle de movimento 3d que permite ao usuário utilizar as mãos e os dedos para interagir com dados, em larga escala, de sistemas de acompanhamento de defeitos. O sistema é composto por 3 camadas view, controller e model. A camada superior é camada de visão. A finalidade desta camada é exibir as saídas e receber como entradas o movimento do utilizador. “Leap Motion” é um dispositivo periférico USB pequeno, que se destina a ser colocado sobre uma área de trabalho físico, voltada para cima. Usando duas câmeras IR monocromáticas e três LEDs de infravermelho, o dispositivo observa uma área hemisférica. Os LEDs geram um padrão 3D de pontos de luz infravermelho e as câmaras geraram quadros de dados refletidos, que é então enviada para o computador. Os movimentos são analisados pelo software controlador do Leap Motion. A camada intermediaria é controller. Ela tem dois componentes paralelos: um interpretador de movimento 3D e um controlador de dados funcionando juntos. Uma vez que o usuário interage com os dados na camada de visão, a interface de usuário vai passar os dados de movimento através do Leap Motion para o intérprete de movimento 3d para interpretar e enviá-lo para o tratamento dos dados. O controlador de dados comunica com a camada de dados para pegar a estrutura de dados JSON provida através do Bugzilla. Controlador de dados usa D3.js, que é uma biblioteca JavaScript para manipulação de documentos baseado em dados para trazer os dados de volta para o usuário com componentes de visualização avançadas.

Outra ferramenta, no entanto para visualizar decisões arquiteturais de design separadamente da arquitetura de software aos quais se refere, é o que se apresenta em [67]. O propósito é facilitar tanto a navegação pelas decisões quanto a análise detalhada das mesmas. A ferramenta guarda as decisões de projeto num arquivo ou banco de dados para recuperar mais tarde. Além disso, é possível exportar ou importar decisões usando XML através de múltiplas estações de trabalho para facilitar a captura e a

distribuição das decisões. O usuário pode criar, modificar, remover tanto as decisões quanto seus relacionamentos e visualizar as decisões de diferentes formas para facilitar a análise. Possui quatro visualizações principais, a primeira é uma lista tabular de decisões e seus relacionamentos, enquanto outra representa as decisões por um grafo de decisão evidenciando sua estrutura. Permite ainda visualização em ordem cronológica e a quarta representa as decisões numa perspectiva do impacto causado.

Em [70] é proposto um sistema gerenciador de versão, FineVMS, de fina granularidade para gerenciar versões de arquivos de código fonte. Trata-se de uma extensão da linha de comando do Subversion para apoiar o versionamento de código Java numa fina granularidade. Para tal, para cada arquivo Java sob versionamento, um arquivo baseado em XML representando a estrutura lógica do arquivo é criado automaticamente ao analisar o código fonte, um arquivo adicional também é construído com as restrições de colaboração no arquivo. Estas informações são úteis para aumentar a consciência de contexto das mudanças feitas por outros desenvolvedores nas unidades de código como, classe, método ou atributo de um arquivo Java. Uma extensão do Subclipse foi feita para suportar o versionamento em fina granularidade. Subeclipse é um *plugin* do Eclipse para o sistema Subversion.

• *Q4: Quais frameworks estão sendo utilizados para visualização colaborativa de software na manutenção/evolução colaborativa?*

Esta questão tem como objetivo identificar os frameworks conceituais ou ferramentas que são frequentemente utilizadas para a visualização colaborativa de software e geralmente são usadas para resolver problemas específicos da área de engenharia de software. Exemplos identificados no estudo são descritos a seguir:

Glauco et al. [6] propõe um framework chamado *Colaborattive Source Miner* que é o resultado da combinação de um ambiente interativo baseado em múltiplas visões com elementos de percepção que apoiam a compreensão de software no desenvolvimento distribuído. Fornecendo assim, informações que permitam aos membros das equipes conhecerem de forma iterativa, o que os demais participantes, modificaram e pesquisaram em um determinado software.

Em [17] é dito que dados históricos podem dar indicações sobre o processo de evolução no qual um dado software foi submetido e sobre o conhecimento tácito que foi utilizado durante a evolução, quanto fornecer um histórico sobre a comunicação e a colaboração realizada pelos desenvolvedores ao longo do ciclo de vida do software. Neste contexto foi definido um *framework* para automatizar a análise e a visualização da evolução do software, analisando dados históricos de projetos *open sources*. São levados em conta fatores sociais, técnicos e sócio-técnicos. No fator social, está interessado em analisar questões como: de que modo a comunidade que desenvolve projetos *open source* tem mudado ao longo da evolução do software? Existe algum padrão nas mudanças ocorridas com a comunidade de desenvolvedores de software? No fator técnico estão interessados em entender como se dá a evolução de software observando, por exemplo, mudanças no número de linhas de código, número de *commits* realizados, comentários inseridos, complexidade ciclomática, dentre outros. Já em fatores socio-técnicos, analisar, por exemplo, como se deu a comunicação entre membros da equipe de desenvolvimento, historicamente falando, nas atividades de manutenção de software realizadas durante o ciclo de evolução do software.

Em [16] é apresentado *Churrasco*, um framework para apoiar, de forma distribuída e colaborativa, a análise da evolução de software com base em dados históricos extraídos de três diferentes fontes: (i) um único sistema gerenciador de *bugs*, o *Bugzilla* [Bugzilla, 2013]; (ii) histórico de mudanças provenientes de repositórios de código fonte, (SVN e CVS) e (iii) projeto de código fonte do software que se pretende analisar a evolução. *Churrasco* possui uma variedade de visualizações. Entre elas, destaca-se a (a) *Correlation View*, capaz de indicar a quantidade de linhas de código por classes, os métodos que pertencem a uma dada classe e o número de defeitos que afetaram uma classe ao longo de sua evolução e a (b) *Complexity View*, que exibe a complexidade de um *package* (pacote de código fonte java), medida pela quantidade de linhas de código e o número de métodos contidos.

Em [78] é apresentado um *framework* para análise de arquivos de código fonte provenientes de repositórios de código fonte, objetivando fornecer ao usuário do *framework* visualizações sobre as dependências existentes entre as entidades de código presentes nos repositórios de código fonte, como classes. A ideia principal é fornecer uma arquitetura de *framework* que possa ser evoluída de acordo com a necessidade do usuário, por exemplo, implementando suporte a repositórios de código fonte de preferência. Na publicação é apresentado um mecanismo que permite que os arquivos de código fonte obtidos sejam transformados em um formato específico para que possam ser interpretados pelo *framework.* Após a interpretação, torna-se possível visualizar as dependências, bem com um conjunto de métricas calculadas. Um exemplo delas é a *Network Diameter*, que permite exibe o calculo da distância entre entidades de código, como classes. Para entender melhor esta métrica é preciso entender o princípio por trás do *framework* apresentado: rede de dependência. Uma rede de dependência é formada por grafos desconectados ou não, representando, por exemplo, pacotes de código com suas classes. Um pacote seria representado por círculo, tendo as classes contidas no pacote ligadas entre si, se houver acoplamento entre elas. *Network Diameter* permite calcular o caminho entre duas classes, estando elas em um mesmo pacote ou não. Esta métrica pode dar algum tipo de indício de pontos que podem ser impactados quando uma dada classe é alterada.

Em [32] um framework colaborativo em modo TOP-DOWN para projeto de montagem é demonstrado. Baseado em uma arquitetura cliente servidor, o servidor é responsável pelo gerenciamento das informações globais de design, o cliente provê ao projetista a habilidade de interativamente realizar uma tarefa de design e também é responsável pelo gerenciamento local de informações de design. Duas camadas compõe o sistema, a camada de núcleo e a camada de comunicação. Na base da camada de núcleo estão módulos comuns a um sistema CAD como modelagem de feature, engine de geometria, engine de restrições e um gerador global de ID. Além disso, para apoiar a colaboração top-down no projeto de montagem outros módulos foram extendidos com destaque para um gerenciador de colaboração que tem função de apoiar os designers para coordenar o trabalho com o dos outros, e o agente de propagação de variação que é responsável por manter a consistência entre o modelo de montagem do produto presente no servidor e nos clientes. A outra camada, de comunicação, é usada pelos componentes para enviar e receber dados através da rede.

Em [50] o framework JAFF é apresentado para o reparo automático do software auxiliando a manutenção. Por ser baseado em algoritmos de busca, a entrada de dados é um conjunto de código fonte de programa Java e um conjunto de casos de teste que

devem estar escritos como testes jUnit. Os códigos de entrada são analisados e as árvores de sintaxe são geradas para cada método. Operadores de busca são aplicados às arvores, que funcionam na prática como alterações no código fonte objetivando a aprovação do teste em seguida o código é novamente compilado e os casos de teste são rodados mais uma vez. Isso ocorre até que os casos de teste sejam aprovados.

Em [53] é apresentado um framework orientado a serviço, com capacidade de personalização e expansão para a integração com ferramentas de colaboração. Ele pode integrar ferramentas próprias ou de terceiros, em seguida, escolher e personalizar essas ferramentas para atender às necessidades específicas da plataforma de colaboração de vários domínios. O framework fornece gerenciamento virtual de organização e outras funções fundamentais como os serviços centrais, e descreve um pacote padrão de ferramentas de colaboração, que inclui os principais componentes como atribuição de tarefas, comunicação multimídia, compartilhamento de dados, gestão do conhecimento, acesso a instrumentos e serviços de computação.

Já em [4] é apresentado um framework para descrever, comparar e entender ferramentas de visualização que promovem a conscientização das atividades humanas durante o desenvolvimento de software. Este framework tem vários propósitos, podendo servir como um mecanismo de avaliação formativa para ferramentas de design, uma ferramenta de avaliação para potenciais usuários de ferramentas e ainda como uma ferramenta de comparação, assim os pesquisadores de ferramentas podem comparar e compreender as diferenças entre as várias ferramentas e identificar novas áreas de pesquisa. O framework é formado por cinco dimensões chave são elas: Intenção, informação, apresentação interação e efetividade, derivados da análise dos artigos descritos nas pesquisas e ferramentas da área.

ViMos [75], é um framework para gerar serviços de visualização de informação ligado ao contexto de forma dinâmica. O objetivo é oferecer a informação desejada no momento certo e de forma adequada. Avanços na Web Semântica combinado com sistemas de percepção de contexto e técnicas de visualização podem ajudar a alcançar esse objetivo. A infraestrutura apresentada é capaz de gerar interfaces de usuário movidos a ontologia de forma dinâmica, recuperando informações baseadas no contexto do usuário e adaptando a forma visual da interface para exibição.

• *Q5: Quais modelos que estão sendo utilizadas para visualização colaborativa no contexto de manutenção/evolução colaborativa de software?*

Esta questão tem como objetivo identificar os modelos que são frequentemente utilizadas para a visualização colaborativa de software e geralmente são usadas para resolver problemas na manutenção e evolução do software. Exemplos identificados no estudo são: Modelo de graus de conhecimento, que calcula automaticamente, para cada elemento de código-fonte, um valor real que representa o conhecimento de um desenvolvedor desse elemento com base em dados de autoria e de interação de um desenvolvedor [19]. Outro exemplo é o meta-modelo baseado em conhecimento que serve como um modelo de recurso unificado para integrar características dos principais tipos de objetos que aparecem em modelos de desenvolvimento de software [20].

È apresentado em [38] um modelo genérico de dados de código que pode descrever a mudança em vários níveis de granularidade. A partir de mapeamentos visuais e técnicas de interação é possível visualizar este modelo de código. O objetivo é proporcionar aos desenvolvedores e gerentes de projeto um apoio eficaz durante o

desenvolvimento e a manutenção, para o programa e o processo de conhecimento, explorando as informações sobre evolução contidas em repositórios de SCM, como CVS ou Subversion.

Em [26] é proposto um modelo para capturar a noção de domínio de projetos que pertencem a um repositório de código fonte por meio da exploração da rede formada entre os desenvolvedores que contribuem para o mesmo projeto. O nome dado a essa rede foi de "rede latente de co-desenvolvimento". Ela foi usada para detectar grupos de projetos (clusters) relacionados semanticamente. Estes clusters identificados são mapeados para domínios com a ajuda de uma taxonomia construida a partir de metadados de um website de perguntas e respostas. É definido domínio como uma noção coletiva que existe entre uma população estatisticamente significativa da comunidade de desenvolvedores sobre um grupo de projetos que pertencem a um conceito compartilhado. O domínio não é implementação de uma tecnologia mas, um conceito ao qual vários projetos e tecnologias pertencem. O modelo proposto também pode ser usado para outras aplicações, incluindo categorização dos projetos, medir a popularidade de projetos entre a comunidade de desenvolvedores, recomendações etc.

É apresentada em [45] uma avaliação da notação gráfica CIAN (Collaborative Interactive Application Notation), para modelar atividades colaborativas de aprendizagem. CIAN é uma notação que permite a modelagem do trabalho em grupo e questões de interação homem-computador.

Uma abordagem para estudar a evolução do software OO partir de uma perspectiva multi-granularidade usando a teoria de redes complexas, é o foco em [49]. É introduzido um modelo de rede de software multi-granularidade para representar o software em três níveis de granularidade (especificamente, o nível de método,nível de classe/interface e nível de pacote) e usá-lo para explorar a evolução do software. A primeira etapa da abordagem é a coleta de dados. Refere-se ao processo de extrair componentes de software, tais como métodos, classes / interfaces, pacotes e suas dependências. A ferramenta, SSQAT desenvolvida e apresentada pelo mesmo autor em outro trabalho, é usada para analisar códigos Java compilados (arquivos com extensão .class e.jar). Após a coleta de dados, três tipos de redes de software em diferentes níveis de granularidade (ou seja, nível de método, classe e interface, e nível de pacote) podem ser construídos, compondo um modelo de redes de software multi-granularidade que servem como base para a aplicação da abordagem. Em seguida, a teoria de rede complexa é aplicado para investigar a topologia e a evolução de um software OO.

Outro trabalho ainda em curso [59] para promover a conscientização da equipe e estimular a colaboração no contexto de desenvolvimento de software global. Uma abordagem que trabalha integrada ao sistema Syde. A novidade da abordagem é apresentar os modelos de mudança de Syde como entidades de primeira classe sendo, portanto, é capaz de fornecer informações de sensibilização no nível orientado a objetos. As contribuições feitas até então: (i) Portar uma abordagem centrada na mudança de um único desenvolvedor para projetos multi-desenvolvedores. Nós modelamos sistemas orientados a objetos em Java como ASTs e mudanças nestes sistemas como operações em árvore. O Servidor Syde combina este modelo com informações extraídas do SCM adotado pelo projeto para informar precisamente aos desenvolvedores interessados quais entidades foram alterados. (ii) A utilização de dados históricos coletados por Syde para refinar a analise de evolução. Mais especificamente, foram utilizadas informações

históricas de mudança para refinar a noção de propriedade através da medição do esforço real que cada desenvolvedor tem feito. (iii) Integração de visualizações leves em um IDE para oferecer informações de conscientização. Os plugins de Syde que apresentam diferentes visualizações que dão aos desenvolvedores a noção de quem está editando quais arquivos em tempo real, e também, quem mudou arquivos no passado recente.

Em [64] é proposto um modelo de dados que representa o desenvolvimento de software como um processo de decisões de projeto apoiado por ferramentas que operam em objetos abstratos do projeto. Este modelo é diferente de outras tentativas em que considera explicitamente a funcionalidade das ferramentas, mas, ao mesmo tempo enfatiza a natureza não determinística de decisões de projeto feito por humanos. O modelo de dados proposto pode ser visto como uma extensão substancial de uma abordagem entidade-relacionamento que enfatiza a orientação do processo de suporte a decisões de design e integração de objetos ativos heterogêneos para a base de conhecimento do processo de software.

• *Q6: Quais fontes de dados e ou repositórios que estão sendo utilizadas para visualização colaborativa no contexto de manutenção e ou evolução colaborativa de software?*

Esta questão tem como objetivo identificar as fontes de dados e ou repositórios que são frequentemente utilizadas para a visualização colaborativa de software e geralmente são usadas para resolver problemas na manutenção e evolução do software. Exemplos identificados no estudo são: SVN[1] - Repositório de código fonte, GIT[2] - Repositório de código fonte, Bugzilla[3] - Repositório de defeitos, Jira[4] - Repositório de defeitos [16], CVS[5] - Repositório de código fonte [3], Syde - Repositório de código fonte [14].

1 SVN - Apache *Subversion* é um sistema de controle de versão desenhado especificamente para ser um substituto moderno do CVS, que se considera ter algumas limitações.

2 GIT - é um sistema de controle de versão distribuído e um sistema de gerenciamento de código fonte, com ênfase em velocidade.

3 BugZilla - é uma ferramenta baseada em Web e e-mail que dá suporte ao desenvolvimento do projeto Mozilla, rastreando defeitos e servindo também como plataforma para pedidos de recursos.

4 Jira - é uma ferramenta de acompanhamento de bugs de produto, desenvolvido pela Atlassian . Ele fornece rastreamento de bugs , acompanhamento de problemas e gerenciamento de projetos funções.

5 CVS - *Concurrent Version System* é um sistema de controle de versão que permite que se trabalhe com diversas versões de arquivos organizados em um diretório e localizados local ou remotamente, mantendo-se suas versões antigas e os logs de quem e quando manipulou os arquivos.

2.1.2 Estratégia de pesquisa

Os estudos primários selecionados devem contribuir diretamente para a resposta das questões de pesquisa do mapeamento para que esses trabalhos não sejam desconsiderados pela estratégia de busca [11]. Foram definidos 3 artigos de controle que pertencem ao domínio investigado, que devem ser encontrados na execução da busca:

1) Hattori, L. and D'Ambros, M. and Lanza, M. and Lungu, M. (2011) Software evolution comprehension: Replay to the rescue.

2) Margaret-Anne D. Storey, Davor Čubranić, Daniel M. German. (2005) On the use of visualization to support awareness of human activities in software development: a survey and a framework, Proceedings of the 2005 ACM symposium on Software visualization.

3) B.A. Price, R.M. Baecker, I.S. Small. (1993) A principled taxonomy of software visualization Journal of Visual Languages and Computing.

A definição das palavras-chave de busca, utilizadas na pesquisa, foram feitas com base na estratégia PICOC [21], conforme Tabela 1.

Tabela 1. PICOC

PICOC	**KEY WORDS**
Population	Visualization, Evolution, Maintenance, Software
Intervention	Collaboration, Cooperation, Cooperative
Outcome	Technique, Methodology, Tool, Software, Application, System, Framework, Model
Context	Visualization, Evolution, Maintenance, Software
Comparison	**Não se aplica**

De acordo com o critério citado anteriormente, foram selecionadas, seis bases de dados digitais, as quais também são sugeridas em revisões sistemáticas no domínio de engenharia de software [23] e, algumas, recomendadas por Kitchenham, 2007:

1) Science@Direct (http://www.sciencedirect.com)
2) Engine Village (http://www.engineeringvillage.com)
3) IEEE Digital Library (http://ieeexplore.ieee.org)
4) ACM Digital Library (http://portal.acm.org)
5) ISI Web of Science (www.isiknowledge.com)
6) Scopus (http://www.scopus.com)

A *string* de busca completa, utilizada nas bases acima citadas, é apresentada na tabela 2, a qual está disposta abaixo:

Tabela 2: String de busca completa

String de busca
(Visualization AND Evolution AND Maintenance AND Software) AND (Collaboration OR Cooperation OR Cooperative) AND (technique OR methodology OR (tool OR Software OR Application OR System) OR framework OR model)

A *string* de busca foi adaptada para algumas bases para que as mesmas pudessem retornar, de forma igualitária, os resultados. Devido ao grande volume de resultados da base SCienceDirect foi utilizado limitador de resultado, fornecido pela própria ferramenta, que possibitou uma seleção por áreas de interesse em comum ao trabalho sugerido, com as seguintes palavras chaves:

Tabela 3: Limitadores da base SCienceDirect

Limitadores da base SCienceDirect
(software, decision support, semantic web, information system,inter face, internet, knowledge management, design process, cad system, web service, business process, manufacturing system, support system, virtual reality)

2.1.3 Critérios e inclusão e exclusão

Para a seleção dos resultados foram definidos alguns critérios para a inclusão e exclusão, os quais auxiliaram a avaliação para seleção dos artigos.

2.1.3.1 Critérios de Inclusão: Foram selecionados artigos que continham uma técnica ou metodologia ou ferramenta ou framework ou tecnologia e que sejam sobre o tema: Visualização, Evolução e Manutenção colaborativa de software .

2.1.3.2 Critérios de Exclusão: Foram excluídos os resultado que não continham alguma técnica ou metodologia ou ferramenta ou framework ou tecnologia mesmo sendo sobre o tema Visualização, Evolução e Manutenção colaborativa de software.

2.1.4 Procedimento de armazenamento e extração de artigos:

Foi utilizado para o armazenamento de artigos, a ferramenta desenvolvida pelo aluno de mestrando da UFJF (Vitor Freitas) de nome Parsifal [22]. Esta ferramenta importa padrões ***bibTex*** e é utilizada por diversos pesquisadores em suas revisões sistemáticas. A escolha da ferramenta se deu em função dos recursos disponibilizados pela mesma, os quais são descritos abaixo. A ferramenta está disponível na Web, facilitando o processo de uso, sem a necessidade de instalação de dependências e quaisquer requisitos não funcionais, como, por exemplo, máquinas virtuais (Java), utilizadas em ferramentas tradicionais.

Por meio da ferramenta podemos criar um projeto de revisão sistemática, dividido em 4 (quatro) passos, os quais encontram-se dispostos e explicados abaixo:

Review – Dados da revisão como Titulo, Autor, Descrição da revisão.

Planning – São definido os objetivos, as questões de pesquisa o PICOC, as palavras chaves, String de Busca e as bases desejadas, bem como os critérios de inclusão e exclusão.

Conducting – São definidos nesse passo, os processos de condução do revisão. Pode-se importar os arquivos padrão ***bibTex*** das bases de dados e fazer a exclusão dos artigos não desejáveis. São apresentados os artigos de uma forma geral ou separados por base o que permite a classificação e filtragem dos mesmos. E possível também acessar o abstract dos artigos bem como obter informações relevantes como Titulo, Autor, Meio de Publicação e Ano de escrita.

Reporting – É possível por meio dessa opção obter os gráficos de artigos por base, artigo aceitos e rejeitados, bem como o gráfico de artigos por ano.

Os artigos que foram importados na ferramenta supracitada, foram extraídos das bases constantes na sessão **2.1.2**. A maioria das bases, exceto ACM, permitem a seleção de todos os artigos resultantes da *string* de busca aplicada, e sua exportação para um único arquivo no formato ***bibTex.*** Para a base de dados da ACM, foram selecionados manualmente cada um dos 27 artigos resultantes da *string* de busca, e exportados um a um para o formato ***bibTex***, visto que a ferramenta não disponibiliza o recurso de extração de múltiplos arquivos. Após a exportação dos artigos para o formato, foram importados para a ferramenta Parsifal, todos os artigos separados por base.

Após esses passos, foi possível, realizar diversas analises e o prosseguimento do processo de revisão sistemática. O resultado das importações, bem como o gráfico de valores totais por base e os critérios de exclusão e inclusão, são detalhados na sessão seguinte.

2.2 Execução da pesquisa

Após a conclusão do protocolo definido na seção anterior, foi iniciada a execução do protocolo conforme descrita nessa seção, que é dividida de acordo com a execução de cada etapa do protocolo definido.

2.2.1 Execução da busca nas bases

A pesquisa abrangeu todos os trabalhos publicados até 2014 (inclusive). A *String* de busca definida na Tabela 2 foram calibrados e aplicados em cada base de dados digital. O gráfico disposto na Figura 1, ilustra o processo de busca. Ele indica que a primeira

pesquisa devolveu **692** resultados, distribuídos conforme ilustrado nas seis bases selecionadas.

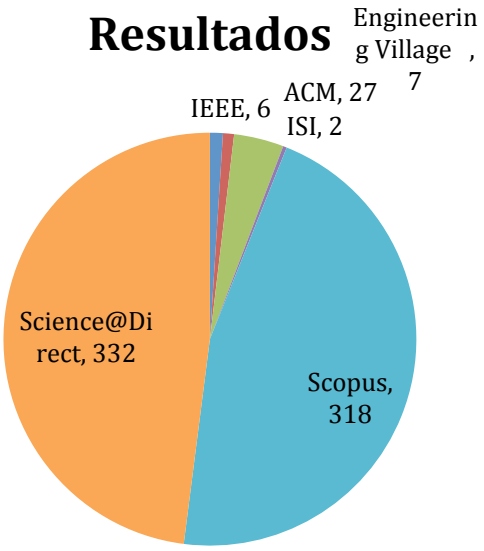

Figura 1 – Resultados obtidos pela *string* de busca

2.2.2 Execução do processo de Inclusão/Exclusão

O processo de exclusão foi conduzido de forma a privilegiar somente os artigos que atendessem os critérios previamente estabelecidos neste artigo. O título e resumo dos artigos selecionados foram então analisados para verificar se eles de fato tratam do tema sobre as tarefas de visualização colaborativa de software estão sendo utilizadas no contexto de manutenção ou evolução colaborativa de software. O título e análise do resumo reduziu a lista total de **692** para **72** publicações. A figura 2 demonstra as exclusões totais feitas em cada base.

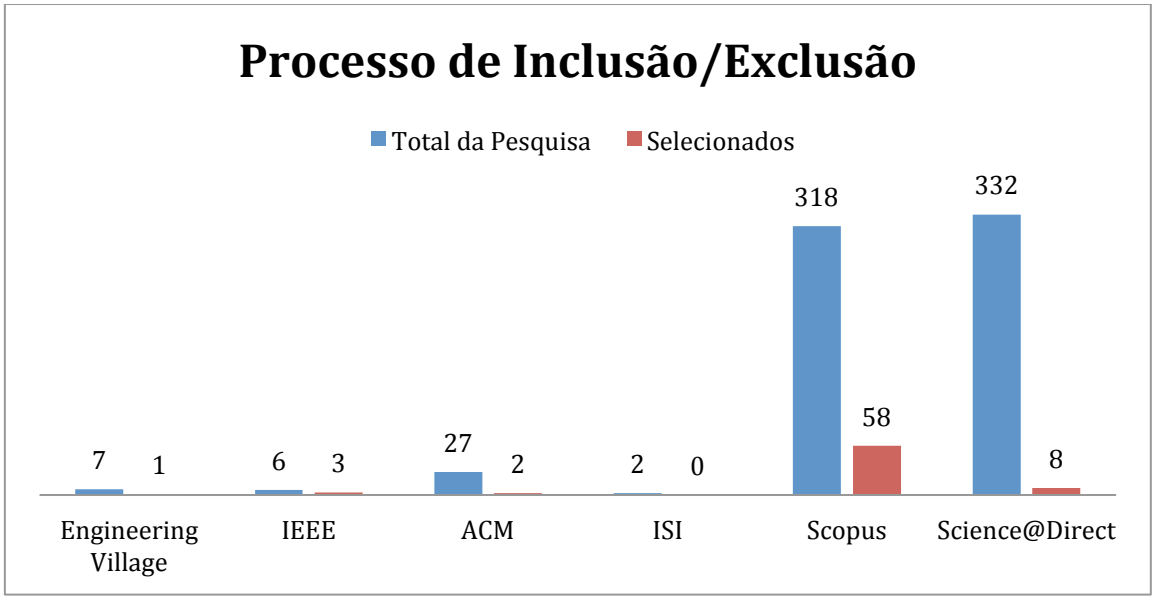

Figura 2 – Gráfico do processo de inclusão e exclusão

Foram identificados 2 casos especiais de exclusão. Ambos retornaram na string de busca aplicada e após a análise do título e do abstract deveriam ser aceitos por serem pertinentes ao assunto, no entanto não foi possível acessá-los e obter o texto na íntegra para a análise completa do mesmo. Os artigos em questão apresentam respectivamente, um framework [88] e uma técnica chamada Team Radar [89].

3. Apresentação dos Resultados

Mediante a finalização do processo de seleção dos artigos, foi realizada uma análise quanto aos resultados obtidos. Na sequência serão descritas as análises efetivadas, bem como as constatações obtidas por meio desta verificação.

3.1 - Variações de publicações ao longo do ano:

O gráfico na Figura 3 demonstra o número de publicações por ano.

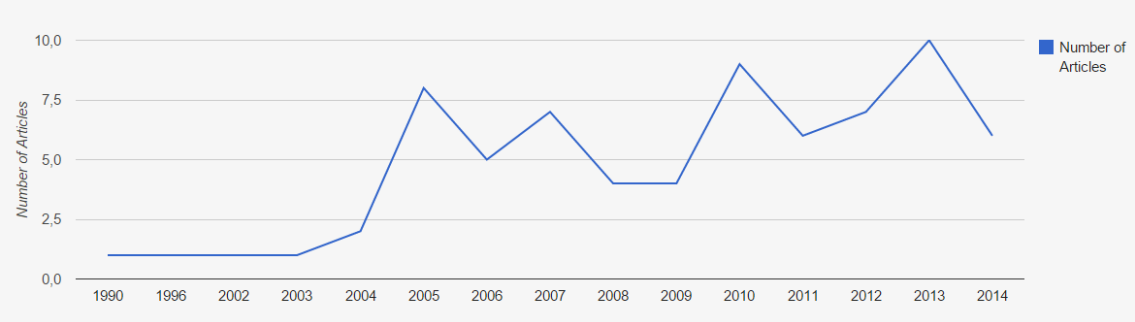

Figura 3 – Gráfico de publicações ao longo do ano

Analisando as informações pode-se notar que o aumento significativo de publicações nesta área se deu a partir do ano de 2005, com um total de 8 publicações. Este aumento substancial justifica-se pela ocorrência de importantes conferências na área, tais como **CSCW** - *Computer Supported Cooperative Work*, **IWPSE-**

International Workshop on Principles of Software Evolution, **SIGSOFT** - *Symposium on the Foundations of Software Engineering*. Como as conferências acontecem anualmente, observa-se que a partir do ano de realização dos eventos, as publicações mantiveram uma média, conforme apresentado na figura 3.

3.2 - Locais de publicações

Os trabalhos de visualização e evolução do software são publicadas em muitos locais diferentes e foram identificados **71** meios de publicação. A figura 4 demonstra os 11 meios, considerados mais importantes, de acordo com o critério de número de publicações maior que dois, com base nos resultados obtidos pela seleção dos artigos.

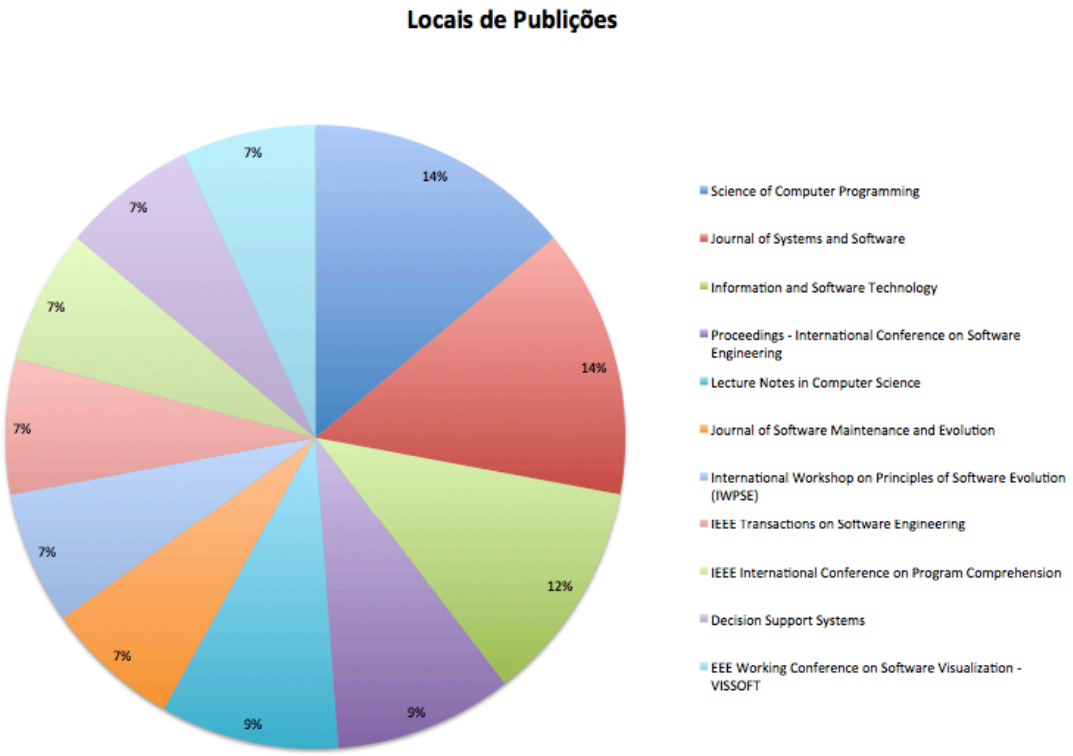

Figura 4 – Locais de publicação

Pode-se ainda, relatar algumas conferências importantes na área, de acordo com documentos retornados, tais como:

Empirical Software Engineering, International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE), IEEE *International Conference on Software Maintenance* (ICSM) e *Proceedings SoftVis* - ACM *Symposium on Software Visualization*.

3.3 – Principais autores da área

O objetivo deste tópico é identificar os pesquisadores mais influentes na área. Descobrimos que **285** autores publicaram trabalhos sobre o tema pesquisado. A Tabela

4 demonstra os **10** autores que mais publicaram na área. A figura 5 demonstra em forma gráfica as publicações por autores.

Tabela 4 – Total de publicações

-	Autor	Total de Publicações
1	M. Lanza	12
2	D'Ambros	6
4	Hattori, L.	6
5	Lungu, M.	6
8	Telea, A.	4
7	Storey, M.-A.	3
3	De Souza, C.R	2
6	Matthias, Jarke	2
9	Theron, R.	2
10	Rajlich, V.	1

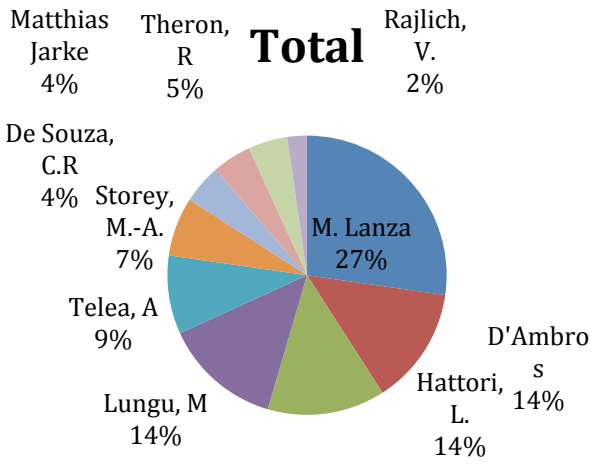

Figura 5 – Principais autores da area

3.4 – Dez artigos mais referenciados

Investigou-se nesse artigo também os 10 artigos mais referenciados em outros estudos, que são apresentados na Tabela 5, abaixo. O artigo *The three dimensions of requirements engineering*: *A framework and its applications* [A1] recebeu 277 citações visto se tratar de um artigo seminal para a área de visualização de software.

Tabela 5 - Dez artigos mais referenciados	Ano	Citaçoes
DECOR: A method for the specification and detection of code and design smells [60]	2010	164
On the use of visualization to support awareness of human activities in software development: A survey and a framework [4]	2005	136
How developers drive software evolution [72]	2005	121
Supporting collaborative software development through the visualization of socio-technical dependencies [12]	2007	97
Using dynamic information for the iterative recovery of collaborations and roles [35]	2002	93
Classroom BRIDGE: Using Collaborative Public and Desktop Timelines to Support Activity Awareness [28]	2003	83
A meta-model for formulating knowledge-based models of software development [20]	1996	76
A software process data model for knowledge engineering in information systems [64]	1990	72

Designing Task Visualizations to Support the Coordination of Work in Software Development [44]	2006	66
The Visual Code Navigator: An interactive toolset for source code investigation [84]	2005	55

3.5 – Categoria dos artigos

Nessa subseção são mostrados através do gráfico da Figura 6 a divisão final dos 72 artigos selecionados nas categorias especificadas pelas questões de pesquisa de 1 a 5. Assim, 21 representam técnicas, 8 representam modelos, outros 9 representam frameworks, 3 representam metodologias adotadas, e 31 representam ferramentas.

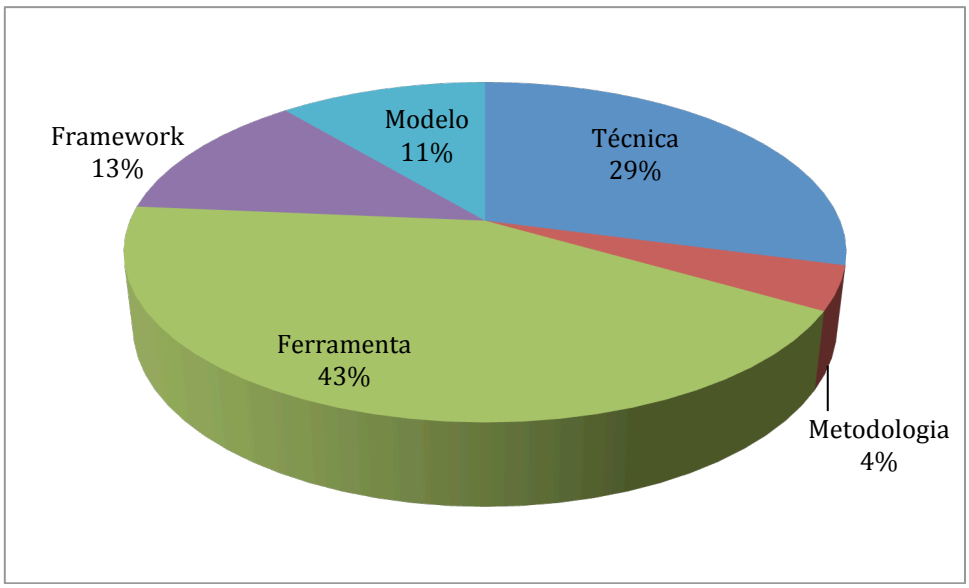

Figura 6: Artigos divididos por categoria

Vale uma resalva que o número total de artigos classificados na categoria ferramenta, por exemplo, não representa o número total de ferramentas encontradas. Isso porque em alguns casos mais de um artigo fazem referência a uma mesma ferramenta ou técnica, como é o caso da ferramenta Replay referenciada em [31][33] e [3]. O mesmo ocorre com outras ferramentas como [8,43] e [52,54], e a técnica apresentada em [77,84].

4. Ameaças à validade

O presente trabalho possui limitações quanto a sua validade, pois embora tenha sido definido um procedimento sistemático para a realização das etapas de inclusão e ou exclusão, esta deve ser feita por pares e apenas em comum acordo deve haver exclusão ou inclusão. Apenas o autor foi responsável pela execução e desse procedimento, portanto, existe uma possibilidade de erro de interpretação, sobretudo nas etapas preliminares de inclusão/exclusão.

As questões de investigação definidas neste estudo não podem fornecer cobertura completa da área de visualização de software. A definição das perguntas foram definidas do ponto de vista de quais são as tarefas de visualização colaborativa de

software estão sendo utilizadas no contexto de manutenção ou evolução colaborativa de software, do ponto de vista que acreditou-se ser importante para a pesquisa.

Mesmo com a tentativa de ser obter um *string* de busca abrangente e que atende-se as necessidades da pesquisa não foi possível com ela abordar todos os estudos possíveis na área de visualização colaborativa de software. Utilizando-se das técnicas de *snow-balling* [24] pode-se recuperar muitos artigos importantes e interessantes a pesquisa que não foram abrangidos na *string* de busca.

5. Considerações Finais

O objetivo deste trabalho foi realizar uma revisão sistemática que ajuda nas pesquisas a respeito da de visualização colaborativa de software estão sendo utilizadas no contexto de manutenção ou evolução colaborativa de software, identificando assim literaturas relevantes para este contexto bem como autores, ferramentas frameworks, técnicas apresentadas nos resultados da pesquisa.

Foi observado que grande parte dos artigos abordam ferramentas de visualização e seu uso no contexto de visualização e manutenção evolutiva de softwares. Identificou-se também importantes fontes de publicações desses estudos e congressos importantes da área.

Foi demonstrado um gráfico de publicações relevantes por ano que remete a importância, visto o baixo número de publicações indexadas e retornadas de um fortalecimento de estudos nas áreas e mais publicações importantes melhorando assim a maturidade da área.

Também foi mostrado no artigo uma lista dos autores mais influentes na área, segundo os resultados obtidos pela *string* de busca, o que possibilita uma analise de que técnicas, metodologias, modelos ou ferramentas estão sendo usados como alvos para as pesquisas realizadas por estes, mostrando tendências de pesquisa da área de evolução manutenção e visualização de software.

Foi respondido ao longo do estudo uma importante questão, ao se descobrir quais fontes de dados e ou repositórios que estão sendo utilizadas para visualização colaborativa no contexto de manutenção e ou evolução colaborativa de software como: SVN, GIT, Bugzilla e CVS. Esses resultados, contêm fontes de informações que podem ser utilizadas em trabalhos futuros, realizados na área de visualização nas Atividades de Evolução de Software no Processo de Manutenção Colaborativa.

Referências

[1] S. Diehl. (2007) Software Visualization: Visualizing the Structure, Behaviour, and Evolution of Software, Springer-Verlag, New York.

[2] Václav Rajlich. (2014) Software Evolution and Maintenance, FOSE 2014 Proceedings of the on Future of Software Engineering.

[3] Hattori, L. and D'Ambros, M. and Lanza, M. and Lungu, M. (2011) Software evolution comprehension: Replay to the rescue.

[4] Margaret-Anne D. Storey , Davor Čubranić , Daniel M. German. (2005) On the use of visualization to support awareness of human activities in software

development: a survey and a framework, Proceedings of the 2005 ACM symposium on Software visualization.
[5] B.A. Price, R.M. Baecker, I.S. (1993) Small A principled taxonomy of software visualization Journal of Visual Languages and Computing.
[6] Carneiro, Glauco, Conceição Carlos, David José Maria N. (2012) A Multiple View Environment for Collaborative Software Comprehension, ICSEA 2012: The Seventh International.
[7] Marco D'Ambros, Michele Lanza. (2009) Visual Software Evolution Reconstruction, Journal of Software Maintenance and Evolution: Research and Practice.
[8] M. Lungu, M. Lanza. (2010) The small project observatory: a tool for reverse engineering software ecosystems, in: ICSE'10: Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering – Volume 2, ACM, New York, NY, USA.
[9] P. Isenberg and S. Carpendale. (2007) Interactive Tree Comparison for Co-located Collaborative Information Visualization. IEEE Transactions on Visualization and Computer Graphics, pages 1232–1239.
[10] R. L. Novais, A. Torres, T. S. Mendes, M. Mendonc¸a, and N. Zazworka. (2013) Software evolution visualization: A Systematic Mapping Study.
[11] KITCHENHAM, B.A., CHARTERS, S. (2007) Guidelines for performing systematic literature reviews in software engineering. Tech. Rep. EBSE-2007-01, KeeleUniversity.
[12] De Souza, C.R. and Quirk, S. and Trainer, E. and Redmiles, D.F. (2007) Supporting collaborative software development through the visualization of socio-technical dependencies, GROUP'07 - Proceedings of the 2007 International ACM Conference on Supporting Group Work.
[13] Pakhira, A. and Andras, P. (2012) Using network analysis metrics to discover functionally important methods in large-scale software systems, International Workshop on Emerging Trends in Software Metrics, WETSoM 2012 – Proceedings.
[14] Hattori, L. and Lanza, M. (2010) Syde: A tool for collaborative software development, Proceedings - International Conference on Software Engineering.
[15] Wen, L. and Kirk, D. and Dromey, R.G. (2007) A tool to visualize behavior and design evolution. 9th International Workshop on Principles of Software Evolution, IWPSE 2007, Held in Conjunction with the 6th ESEC/FSE Joint Meeting.
[16] D'Ambros, M. and Lanza, M. (2010) Distributed and Collaborative Software Evolution Analysis with Churrasco, Science of Computer Programming.
[17] Mahbubul Syeed, M.M. and Hammouda, I. and Berko, C. (2013) Exploring socio-technical dependencies in open source software Projects, Proceedings of the 17th International Academic MindTrek Conference.
[18] Lungu, M. and Lanza, M. and Nierstrasz, O. (2014) Evolutionary and collaborative software architecture recovery with Softwarenaut, Science of Computer Programming.
[19] Fritz, T. and Murphy, G.C. and Murphy-Hill, E. and Ou, J. and Hill, E. (2014) Degree-of-knowledge: Modeling a developer's knowledge of code. ACM Transactions on Software Engineering and Methodology.

[20] Peiwei Mi and Walt Scacchi. (1996) A meta-model for formulating knowledge-based models of software development, Decision Support Systems.
[21] Petticrew, M., Roberts, H. (2006) Systematic Reviews in the Social Sciences: A Practical Guide. Blackwell Publishing, Malden.
[22] Parcifal, Disponível em: http://parsif.al/, último acesso 01 de setembro de 2014.
[23] Steinmacher, I., Chaves, A. P. and Gerosa, M. A. (2012) Awareness Support in Distributed Software Development: A Systematic Review and Mapping of the Literature" in EASE'08 Proceedings of the 12th international conference on Evaluation and Assessment in Software Engineering.
[24] D. Budgen, M. Turner, P. Brereton, B. Kitchenham (2008) Using mapping studies in software engineering, in: PPIG'08: 20th Annual Meeting of the Psychology of Programming Interest Group, Lancaster University.
[25] Liu, S., Zheng, Y., Shen, H., Xia, S., & Sun, C. (2006, November). Real-time collaborative software modeling using UML with rational software architect. In*Collaborative Computing: Networking, Applications and Worksharing, 2006. CollaborateCom 2006. International Conference on* (pp. 1-9). IEEE.
[26] Venkataramani, R., Asadullah, A., Bhat, V., & Muddu, B. (2013, September). Latent Co-development Analysis Based Semantic Search for Large Code Repositories. In *Software Maintenance (ICSM), 2013 29th IEEE International Conference on* (pp. 372-375). IEEE.
[27] Balogh, G., & Beszédes, A. (2013, September). CodeMetrpolis—A minecraft based collaboration tool for developers. In *Software Visualization (VISSOFT), 2013 First IEEE Working Conference on* (pp. 1-4). IEEE.
[28] Ganoe, C. H., Somervell, J. P., Neale, D. C., Isenhour, P. L., Carroll, J. M., Rosson, M. B., & McCrickard, D. S. (2003, November). Classroom BRIDGE: using collaborative public and desktop timelines to support activity awareness. In *Proceedings of the 16th annual ACM symposium on User interface software and technology* (pp. 21-30). ACM.
[29] Wu, L., Sahraoui, H., & Valtchev, P. (2004, November). Automatic detecting code cooperation. In *Software Engineering Conference, 2004. 11th Asia-Pacific* (pp. 204-211). IEEE.
[30] Fogli, D., & Piccinno, A. (2005). Environments to support context and emotion aware visual interaction. *Journal of Visual Languages & Computing*,*16*(5), 386-405.
[31] Hattori, L., Lungu, M., & Lanza, M. (2010, September). Replaying past changes in multi-developer projects. In *Proceedings of the Joint ERCIM Workshop on Software Evolution (EVOL) and International Workshop on Principles of Software Evolution (IWPSE)* (pp. 13-22). ACM.
[32] Gao, S., Zhang, S., Chen, X., & Yang, Y. (2013). A framework for collaborative top-down assembly design. *Computers in Industry*, *64*(8), 967-983.
[33] Hattori, L., D'Ambros, M., Lanza, M., & Lungu, M. (2013). Answering software evolution questions: An empirical evaluation. *Information and Software Technology*, *55*(4), 755-775.
[34] Staron, M., Hansson, J., Feldt, R., Meding, W., Henriksson, A., Nilsson, S., & Hoglund, C. (2013, October). Measuring and Visualizing Code Stability--A Case Study at Three Companies. In *Software Measurement and the 2013*

Eighth International Conference on Software Process and Product Measurement (IWSM-MENSURA), 2013 Joint Conference of the 23rd International Workshop on (pp. 191-200). IEEE.

[35] Richner, T., & Ducasse, S. (2002). Using dynamic information for the iterative recovery of collaborations and roles. In *Software Maintenance, 2002. Proceedings. International Conference on* (pp. 34-43). IEEE.

[36] Hattori, L. P., Lanza, M., & Robbes, R. (2012). Refining code ownership with synchronous changes. *Empirical Software Engineering*, *17*(4-5), 467-499.

[37] Pérez, J., Deshayes, R., Goeminne, M., & Mens, T. (2012, March). Seconda: Software ecosystem analysis dashboard. In *Software Maintenance and Reengineering (CSMR), 2012 16th European Conference on* (pp. 527-530). IEEE.

[38] Voinea, L., Lukkien, J., & Telea, A. (2007). Visual assessment of software evolution. *Science of Computer Programming*, *65*(3), 222-248.

[39] Rothlisberger, D., Harry, M., Binder, W., Moret, P., Ansaloni, D., Villazon, A., & Nierstrasz, O. (2012). Exploiting dynamic information in IDEs improves speed and correctness of software maintenance tasks. *Software Engineering, IEEE Transactions on*, *38*(3), 579-591.

[40] Hatzi, O., Vrakas, D., Bassiliades, N., Anagnostopoulos, D., & Vlahavas, I. (2010). A visual programming system for automated problem solving. *Expert Systems with Applications*, *37*(6), 4611-4625.

[41] Wang, H., Zhang, X., & Zhou, M. (2012, May). MaVis: Feature-Based Defects Visualization in Software Testing. In *Engineering and Technology (S-CET), 2012 Spring Congress on* (pp. 1-4). IEEE.

[42] Heller, B., Marschner, E., Rosenfeld, E., & Heer, J. (2011, May). Visualizing collaboration and influence in the open-source software community. In*Proceedings of the 8th Working Conference on Mining Software Repositories*(pp. 223-226). ACM.

[43] Lungu, M., Lanza, M., Gîrba, T., & Robbes, R. (2010). The small project observatory: Visualizing software ecosystems. *Science of Computer Programming*, *75*(4), 264-275.

[44] Halverson, C. A., Ellis, J. B., Danis, C., & Kellogg, W. A. (2006, November). Designing task visualizations to support the coordination of work in software development. In *Proceedings of the 2006 20th anniversary conference on Computer supported cooperative work* (pp. 39-48). ACM.

[45] Molina, A. I., Redondo, M. A., Ortega, M., & Lacave, C. (2014). Evaluating a graphical notation for modeling collaborative learning activities: A family of experiments. Science of Computer Programming, 88, 54-81.

[46] Reiss, S. P., Bott, J. N., & La Viola, J. J. (2014). Plugging in and into code bubbles: the code bubbles architecture. Software: Practice and Experience,44(3), 261-276.

[47] Takama, Y., & Kurosawa, T. (2014). Visualization System for Monitoring Bug Update Information. *IEICE TRANSACTIONS on Information and Systems*,*97*(4), 654-662.

[48] Figueiredo, M. C., & de Souza, C. R. (2012, June). Wolf: Supporting impact analysis activities in distributed software development. In *Cooperative and Human Aspects of Software Engineering (CHASE), 2012 5th International Workshop on* (pp. 40-46). IEEE.

[49] Pan, W., Li, B., Ma, Y., & Liu, J. (2011). Multi-granularity evolution analysis of software using complex network theory. *Journal of Systems Science and Complexity*, *24*(6), 1068-1082.

[50] Arcuri, A. (2011). Evolutionary repair of faulty software. *Applied Soft Computing*, *11*(4), 3494-3514.

[51] Dalton, A. R., Wahba, S. K., Dandamudi, S., & Hallstrom, J. O. (2009). Visualizing the runtime behavior of embedded network systems: A toolkit for TinyOS. *Science of Computer Programming*, *74*(7), 446-469.

[52] Theron, R., Gonzalez, A., & Garcia, F. J. (2008, September). Supporting the understanding of the evolution of software items. In *Proceedings of the 4th ACM symposium on Software visualization* (pp. 189-192). ACM.

[53] Luo, T., Song, J., Chen, S., Liu, Y., Xu, Y., Du, C., & Liu, W. (2007, August). A services oriented framework for integrated and customizable collaborative environment. In *Information Reuse and Integration, 2007. IRI 2007. IEEE International Conference on* (pp. 385-393). IEEE.

[54] Therón, R., González, A., García, F. J., & Santos, P. (2007). The use of information visualization to support software configuration management. In*Human-Computer Interaction–INTERACT 2007* (pp. 317-331). Springer Berlin Heidelberg.

[55] Seeberger, M., Kuhn, A., Girba, T., & Ducasse, S. (2006, March). Chronia: Visualizing how developers change software systems. In *Software Maintenance and Reengineering, 2006. CSMR 2006. Proceedings of the 10th European Conference on* (pp. 2-pp). IEEE.

[56] Rötschke, T., & Schürr, A. (2006). Temporal graph queries to support software evolution. In *Graph Transformations* (pp. 291-305). Springer Berlin Heidelberg.

[57] D'Ambros, M., Lanza, M., & Gall, H. (2005). Fractal figures: Visualizing development effort for cvs entities. In *Visualizing Software for Understanding and Analysis, 2005. VISSOFT 2005. 3rd IEEE International Workshop on*(pp. 1-6). IEEE.

[58] Koldehofe, B. (2005). *Distributed Algorithms and Educational Simulation/Visualisation in Collaborative Environments*. Gothenburg, Sweden: Chalmers University of Technology.

[59] Hattori, L. (2010, May). Enhancing collaboration of multi-developer projects with synchronous changes. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 2* (pp. 377-380). ACM.

[60] Moha, N., Gueheneuc, Y. G., Duchien, L., & Le Meur, A. (2010). DECOR: A method for the specification and detection of code and design smells.*Software Engineering, IEEE Transactions on*, *36*(1), 20-36.

[61] Rilling, J., & Mudur, S. P. (2005). 3D visualization techniques to support slicing-based program comprehension. *Computers & Graphics*, *29*(3), 311-329.

[62] Fronza, I., Janes, A., Sillitti, A., Succi, G., & Trebeschi, S. (2013, May). Cooperation wordle using pre-attentive processing techniques. In *Cooperative and Human Aspects of Software Engineering (CHASE), 2013 6th International Workshop on* (pp. 57-64). IEEE.

[63] Yongpisanpop, P., Hata, H., & Matsumoto, K. (2014, May). Bugarium: 3d interaction for supporting large-scale bug repositories analysis.

In *Companion Proceedings of the 36th International Conference on Software Engineering*(pp. 500-503). ACM.

[64] Jarke, M., Jeusfeld, M., & Rose, T. (1990). A software process data model for knowledge engineering in information systems. *Information Systems*,*15*(1), 85-116.

[65] Del Rosso, C. (2009). Comprehend and analyze knowledge networks to improve software evolution. *Journal of software maintenance and evolution: research and practice*, *21*(3), 189-215.

[66] Cleary, B., Exton, C., Buckley, J., & English, M. (2009). An empirical analysis of information retrieval based concept location techniques in software comprehension. *Empirical Software Engineering*, *14*(1), 93-130.

[67] Lee, L., & Kruchten, P. (2008). A tool to visualize architectural design decisions. In *Quality of Software Architectures. Models and Architectures*(pp. 43-54). Springer Berlin Heidelberg.

[68] Bohnet, J., Voigt, S., & Dollner, J. (2008, June). Locating and understanding features of complex software systems by synchronizing time-, collaboration-and code-focused views on execution traces. In *Program Comprehension, 2008. ICPC 2008. The 16th IEEE International Conference on* (pp. 268-271). IEEE.

[69] D'Ambros, M., Lanza, M., & Pinzger, M. (2007, June). " A Bug's Life" Visualizing a Bug Database. In *Visualizing Software for Understanding and Analysis, 2007. VISSOFT 2007. 4th IEEE International Workshop on* (pp. 113-120). IEEE.

[70] De Lucia, A., Fasano, F., Oliveto, R., & Santonicola, D. (2007, September). Improving context awareness in subversion through fine-grained versioning of Java code. In *Ninth international workshop on Principles of software evolution: in conjunction with the 6th ESEC/FSE joint meeting* (pp. 110-113). ACM.

[71] Greevy, O., Ducasse, S., & Gîrba, T. (2006). Analyzing software evolution through feature views. *Journal of Software Maintenance and Evolution: Research and Practice*, *18*(6), 425-456.

[72] Girba, T., Kuhn, A., Seeberger, M., & Ducasse, S. (2005, September). How developers drive software evolution. In *Principles of Software Evolution, Eighth International Workshop on* (pp. 113-122). IEEE.

[73] Volzer, H., MacDonald, A., Atchison, B., Hanlon, A., Lindsay, P., & Strooper, P. (2004). SubCM: A tool for improved visibility of software change in an industrial setting. *Software Engineering, IEEE Transactions on*, *30*(10), 675-693.

[74] Servant, F., & Jones, J. A. (2012, November). History slicing: assisting code-evolution tasks. In *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering* (p. 43). ACM.

[75] Hervás, R., & Bravo, J. (2011). Towards the ubiquitous visualization: Adaptive user-interfaces based on the Semantic Web. *Interacting with Computers*, *23*(1), 40-56.

[76] Shrestha, A., Zhu, Y., & Miller, B. (2013, September). Visualizing time and geography of open source software with storygraph. In Software Visualization (VISSOFT), 2013 First IEEE Working Conference on (pp. 1-4). IEEE.

[77] Telea, A., & Voinea, L. (2005, September). Interactive visual mechanisms for exploring source code evolution. In *Visualizing Software for Understanding and Analysis, 2005. VISSOFT 2005. 3rd IEEE International Workshop on*(pp. 1-6). IEEE.

[78] Mao, C. (2011, November). Structure visualization and analysis for software dependence network. In Granular Computing (GrC), 2011 IEEE International Conference on (pp. 439-444). IEEE.

[79] Anslow, C., Marshall, S., Noble, J., & Biddle, R. (2013, September). Sourcevis: Collaborative software visualization for co-located environments. In *Software Visualization (VISSOFT), 2013 First IEEE Working Conference on* (pp. 1-10). IEEE.

[80] Voigt, S., Bohnet, J., & Dollner, J. (2009, September). Object aware execution trace exploration. In *Software Maintenance, 2009. ICSM 2009. IEEE International Conference on* (pp. 201-210). IEEE.

[81] Anslow, C., Marshall, S., Noble, J., & Biddle, R. (2010, October). Co-located collaborative software visualization. In *Human Aspects of Software Engineering* (p. 4). ACM.

[82] Kevic, K., Muller, S. C., Fritz, T., & Gall, H. C. (2013, May). Collaborative bug triaging using textual similarities and change set analysis. In *Cooperative and Human Aspects of Software Engineering (CHASE), 2013 6th International Workshop on* (pp. 17-24). IEEE.

[83] Telea, A., & Auber, D. (2008, May). Code flows: Visualizing structural evolution of source code. In *Computer Graphics Forum* (Vol. 27, No. 3, pp. 831-838). Blackwell Publishing Ltd.

[84] Lommerse, G., Nossin, F., Voinea, L., & Telea, A. (2005, October). The visual code navigator: An interactive toolset for source code investigation. In*Information Visualization, 2005. INFOVIS 2005. IEEE Symposium on* (pp. 24-31). IEEE.

[85] Motschnig-Pitrik, R., & Mittermeir, R. T. (1996). Language features for the Interconnection of Software Components. *Advances in Computers*, *43*, 51-139.

[86] Chen, C., & Zhang, K. (2013). Team Radar: A Radar Metaphor for Workspace Awareness. In *Evaluation of Novel Approaches to Software Engineering* (pp. 145-154). Springer Berlin Heidelberg.