UNIVERSIDADE FEDERAL DE MINAS GERAIS

ESCOLA DE ENGENHARIA

CURSO DE ENGENHARIA DE SISTEMAS

MATHEUS SILVA ARAUJO

GRAFOS DE EVOLUÇÃO DA MATURIDADE: PROPOSTA DE UM MODELO PARA REPRESENTAÇÃO DA EVOLUÇÃO DO CONHECIMENTO E DA CAPACIDADE

TRABALHO DE CONCLUSÃO DE CURSO II

BELO HORIZONTE

2019

MATHEUS SILVA ARAUJO

GRAFOS DE EVOLUÇÃO DA MATURIDADE: PROPOSTA DE UM MODELO PARA REPRESENTAÇÃO DA EVOLUÇÃO DO CONHECIMENTO E DA CAPACIDADE

Trabalho de Conclusão de Curso apresentado como requisito parcial à obtenção do título de Engenheiro de Sistemas, da Escola de Engenharia da Universidade Federal de Minas Gerais.

Orientador: Profa. Dra. Ana Liddy Cenni de Castro Magalhães

Co-orientador: Vinicius Matos Paiva

BELO HORIZONTE

2019

Dedico este trabalho aos meus avós,
pelos ensinamentos explícitos e não-
explícitos de toda uma vida.

AGRADECIMENTOS

Agradeço à Professora Ana Liddy, sem a qual este trabalho não seria possível. Por toda sua dedicação enquanto Coordenadora do Curso de Engenharia de Sistemas e como orientadora desse trabalho e da vida. É motivo de honra sua participação aqui.

Agradeço também à DTI Digital, representada aqui pelo Vinicius, outro orientador neste trabalho e na vida; mas também a todas as outras pessoas magníficas que ali trabalham e com quem tenho a alegria de dividir algumas horas do meu dia e algumas ideias mirabolantes. O ambiente desafiador, adaptativo e complexo que ali existe é elemento fundamental neste trabalho e na minha formação.

Agradeço à minha família que permitiu que eu chegasse são e salvo até aqui.

Ao amigo Felippe, pelos diálogos que ajudaram a conduzir este trabalho.

A todos os professores e colegas de todos os cursos pelos quais passei antes de encontrar meu lugar no curso de Engenharia de Sistemas.

Vem cá, sentar aqui do lado. Vamos pra
estrada do acaso, destino e rota tanto faz,
no caminho você escolhe o que vem
primeiro, o que vai atrás. Qual a distância
a percorrer? São quantas milhas até lá?
Que canção vamos eleger, quando
chegar perto do mar?

(ASSUNÇÃO, Leo, 2015)

RESUMO

Modelos de maturidade são ferramentas utilizadas para avaliar um conjunto de capacidades, definir o nível de maturidade de um indivíduo ou organização com relação a um objetivo específico e nortear ações de melhoria visando evoluir segundo aquele modelo. No entanto, observa-se que as formas de representação usuais desses modelos, mais lineares, são pouco aderentes à realidade das ações para a sua implantação. Por outro lado, o Framework Cynefin propõe diferentes abordagens para diferentes contextos baseando-se nas suas relações de causa e efeito, do qual se pode inferir que os modelos atuais de representação da evolução da maturidade estão no domínio do Simples, podendo ser levados para o domínio do Complicado por meio de manipulações adequadas utilizando grafos. Este trabalho tem como objetivo desenvolver um método de representação de modelos de evolução da maturidade utilizando grafos direcionados acíclicos. Para validar tal método, é implementada uma solução computacional em ambiente web para alimentar, modelar e analisar essa representação. Visando melhor contextualizar o potencial do método de representação proposto, possíveis aplicações da representação e da solução são elencadas, entre elas, a modelagem da grade curricular de um curso de graduação e de um modelo de maturidade para DevOps.

Palavras-chave: Modelos de Maturidade e Capacidade. Grafos. Cynefin Framework. Gestão do conhecimento. Desenvolvimento Web.

ABSTRACT

Maturity models are tools used to evaluate a set of capabilities, define the level of maturity of an individual or organization with respect to a specific objective and guide improvement actions aiming to evolve according to that model. However, it is observed that the usual forms of representation of these models, more linear, are little adherent to the reality of the actions for their implantation. On the other hand, the Cynefin Framework proposes different approaches to different contexts based on their cause and effect relationships, from which it can be inferred that the current models of the representation of the evolution of maturity are in the domain of Simple, and can be taken to the domain of the Complicated by means of suitable manipulations using graphs. This work aims to develop a method of representing maturity evolution models using acyclic target graphs. To validate such a method, a computational solution is implemented in web environment to feed, model and analyze this representation. In order to better contextualize the potential of the proposed representation method, possible applications of the representation and its solution are suggested, among them, the modeling of the curriculum of an undergraduate course and a DevOps maturity model.

Keywords: Maturity and Capability Models. Graphs. Cynefin Framework. Knowledge management. Web development.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE SIGLAS

CSS	Cascading Style Sheets
ISACA	Information Systems Audit and Control Association
CMMI	Capability Maturity Model Integration
HTML	Hypertext Markup Language
RDF	Resource Description Framework
SKOS	Simple Knowledge Organization System
SPICE	Software Process Improvement and Capability Determination

SUMÁRIO

1 INTRODUÇÃO

Os modelos de evolução da maturidade e/ou da capacidade, muitas vezes referenciados simplesmente como modelos de maturidade, podem ser entendidos como guias destinados a avaliar processos em diversos contextos visando direcionar oportunidades de melhoria. Esses modelos orientam a determinação do nível de habilidade de uma organização ou indivíduo na prática de um ou mais processos, de acordo com referenciais definidos pelos próprios modelos. A partir da informação disponibilizada nesses modelos, um indivíduo ou empresa pode entender melhor seu estágio atual à luz do modelo, reconhecer suas capacidades e limitações, bem como traçar planos a fim de aprimorar suas habilidades e tornar-se melhor na prática daquele conjunto de processos.

Nesses modelos, a representação da evolução da maturidade é geralmente tratada de forma linear, por níveis, e com pouca ou nenhuma variação. As práticas relacionadas aos processos são sequenciadas em um padrão pré-formatado que é pouco preciso quando tenta representar as especificidades naturais de cada indivíduo ou organização tentando atingir um nível alto de maturidade naquele modelo.

Nesse trabalho, pretende-se utilizar uma forma menos restritiva de representação da evolução da maturidade, por meio da utilização da teoria de grafos. No modelo proposto, as relações entre as capacidades são representadas por uma rede em que a evolução pode ser percebida como uma propriedade emergente do modelo representado.

Além de uma representação mais flexível, a utilização do formalismo matemático da teoria de grafos torna possível o uso de algoritmos conhecidos para o tratamento de grafos. Acredita-se que esse uso poderá potencialmente revelar informações até então desconhecidas desses modelos de evolução da maturidade.

Para fundamentar essa proposta, foram estudados alguns dos modelos de maturidade estabelecidos na literatura (Crosby, 1979) (Weinberg, 1992) (Salviano, 2003). Nesses modelos, a maturidade pode ser vista como uma representação do conjunto de capacidades do indivíduo ou organização. Entendendo a capacidade como a aplicação prática de conhecimento, foram também considerados conceitos relacionados à gestão do conhecimento, como a origem do conhecimento, seus tipos e modos de conversão (Moreira, 2005) e (Zambalde e Alvez, 2004).

Dentro do pensamento sistêmico, foi estudado também o Cynefin Framework (Kurtz e Snowden, 2003). Esse modelo propõe uma estrutura de classificação de sistemas em diferentes domínios: simples, complicados, complexos ou caóticos.

Neste contexto, considerando que os modelos de evolução da maturidade atuais avaliam a evolução da maturidade em um domínio simples, em que as relações de causa e consequência são repetíveis e previsíveis, este trabalho busca responder à seguinte questão: é viável a construção de um modelo de evolução da maturidade em um domínio complicado, em que as relações de causa e efeito são analisadas em maior profundidade, e não de forma imediatista.

O objetivo principal deste trabalho de graduação é desenvolver um método de representação dos modelos de evolução da maturidade utilizando grafos e implementar uma solução computacional que faça essa representação.

Os objetivos específicos são:

1. Dominar os principais conceitos e algoritmos de teoria de grafos;
2. Analisar os principais modelos de evolução da maturidade existentes;
3. Caracterizar e conceituar um domínio complicado;
4. Definir os passos para a representação desses modelos por meio de grafos;
5. Criar grafos que possibilitem avaliar pelo menos um tipo de modelo de evolução da maturidade;
6. Analisar os grafos criados em relação a algoritmos pertencentes à teoria de grafos;
7. Disponibilizar a aplicação implementada para ser utilizada em diferentes contextos e por potenciais pessoas interessadas.

O escopo desse trabalho engloba:

1. A proposta de um método de representação dos modelos de evolução da maturidade por meio de grafos;
2. A identificação de cenários que poderiam explorar a utilização desse tipo de método, visando identificar seus principais requisitos;
3. O desenvolvimento computacional desse método;
4. A aplicação e avaliação desse método utilizando pelo menos um dos cenários identificados.

Apesar de a identificação de cenários que poderiam explorar a utilização desse tipo de método ser parte integrante deste trabalho, não é objeto deste estudo a criação dos vários grafos a serem aplicados a cenários distintos – englobando o levantamento das habilidades e capacidades de cada modelo e suas relações.

Este trabalho está organizado em sete capítulos.

Este Capítulo 1 contemplou a introdução ao tema, definindo o contexto em que os modelos de evolução da maturidade são válidos, a motivação e os objetivos a serem atingidos com o desenvolvimento deste trabalho.

O Capítulo 2 apresenta uma breve revisão de outros trabalhos na literatura relacionados ao tema, além de conceitos e exemplos de modelos de maturidade, conceitos sobre Teoria de Grafos e uma análise do problema, proposta sob a ótica do Framework Cynefin.

O Capítulo 3 apresenta a metodologia utilizada para o trabalho.

O Capítulo 4 detalha a abordagem utilizada, descrevendo o método de trabalho, a implementação realizada, tanto do ponto de vista tecnológico quanto dos algoritmos utilizados para cálculos sobre os grafos modelados.

O Capítulo 5 discute os resultados obtidos a partir dessa implementação.

O Capítulo 6 mostra perspectivas de continuidade do trabalho.

O Capítulo 7 encerra o trabalho e tece algumas considerações finais.

2 REFERENCIAL TEÓRICO

Inicialmente, uma revisão bibliográfica foi realizada a fim de compreender os principais conhecimentos envolvidos no contexto deste trabalho, entre eles: conceitos e formas de se realizar a gestão do conhecimento; conceitos básicos sobre a evolução da maturidade; conceitos e distinção entre sistemas simples, complicados, complexos e caóticos por meio do Cynefin Framework; estudos e algoritmos relacionados à teoria de grafos, com aplicações em gestão do conhecimento.

2.1 CONCEITOS RELACIONADOS À GESTÃO DO CONHECIMENTO

Diversos modelos de evolução da maturidade encontrados na literatura definem a maturidade de uma organização ou indivíduo a partir da análise do conjunto de suas capacidades (Koehlegger et al., 2009). Tais capacidades podem ser entendidas como decorrentes da aplicação do conhecimento adquirido por essa organização ou indivíduo. Em função disso, é necessário compreender a origem e a formação do conhecimento. Os grafos de evolução do conhecimento a serem propostos tentarão representar este avanço de capacidade por meio de uma rede estruturada.

2.1.1 Dado, informação e conhecimento

A definição de conhecimento pode ser obtida a partir das definições de dado e informação (Moreira, 2005).

O conceito de “dado” é consensual: os dados são entidades “dadas”, que estão disponíveis no ambiente, podem ser quantificados e possuir significado. Os dados independem da ação humana e estão presentes em todo o espaço e tempo.

O conceito de “informação” não é consensual. Existem mais de 400 definições para esse termo (Moreira, 2005). Etimologicamente, o termo vem do latim “*informatio”*, que quer dizer “em forma”. A informação pode ter as seguintes formas:

- Informação como dados contextualizados, sendo a informação um conjunto de dados dispostos a fim de transmitir sentido;
- Informação como mensagem comunicada, ou informação como processo, sendo a informação o “ato de informar” ou de transmitir uma informação do emissor ao receptor;
- Informação como conteúdo comunicado, ou informação como entidade subjetiva, sendo a informação não apenas a mensagem comunicada, mas também seu conteúdo, ou aquilo que se deseja comunicar.
- Informação como objeto, ou informação como entidade objetiva, sendo a informação sinônimo de objetos, como dados e documentos.

Assim como a definição de informação, o conceito de conhecimento não é consensual. Para este trabalho, o conhecimento pode ser compreendido como a relação de sentido entre a informação e sua aplicação prática (Moreira, 2005).

É importante observar o ganho de valor à medida que o dado é internalizado e transformado em informação e depois em conhecimento, como mostrado na Figura 1.

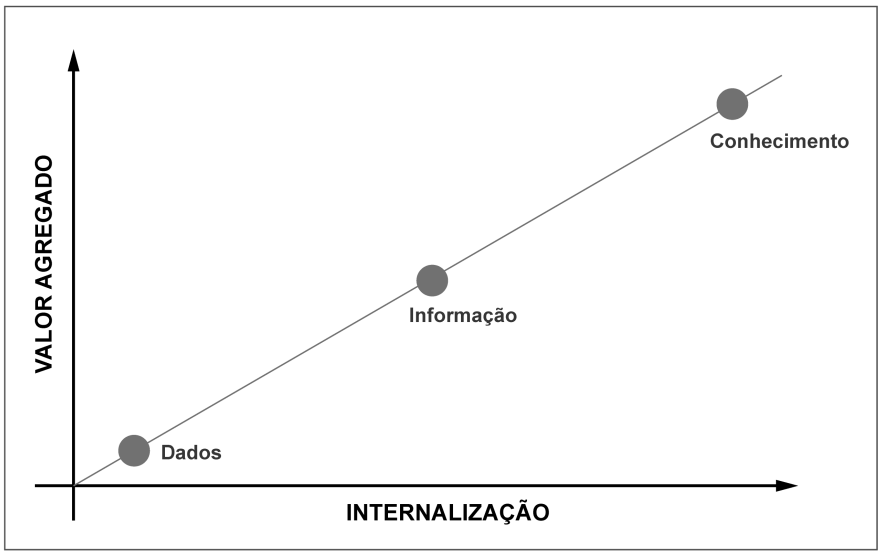

Figura 1 - Ganho de valor obtido com a internalização e transformação de dados em informação e conhecimento - Adaptado de (Moreira, 2005)

2.1.2 Gestão do Conhecimento

A gestão do conhecimento pode ser compreendida como o processo de geração, codificação e transferência de conhecimento (Zambalde e Alvez, 2004).

Segundo Nonaka (1995), conhecimento pode ser classificado em duas categorias:

- Conhecimento explícito: aquele que pode ser transmitido e processado por meio de linguagem formal;
- Conhecimento tácito, ou implícito: aquele transmitido por meio do exemplo ou da convivência.

Ainda segundo Nonaka e Takeuchi (1995), a conversão do conhecimento se dá por quatro diferentes formas, conforme apresentado na Figura 2:

- Socialização – geração do conhecimento tácito por meio do compartilhamento de experiências, como treinamentos, reuniões e interações;
- Externalização – comunicação do conhecimento tácito por meio de analogias, metáforas ou modelos;
- Combinação – troca de informações explícitas, por meio de canais digitais ou analógicos, sendo a base da educação formal;
- Internalização – aprendizado por meio da vivência prática, *learning by doing*.

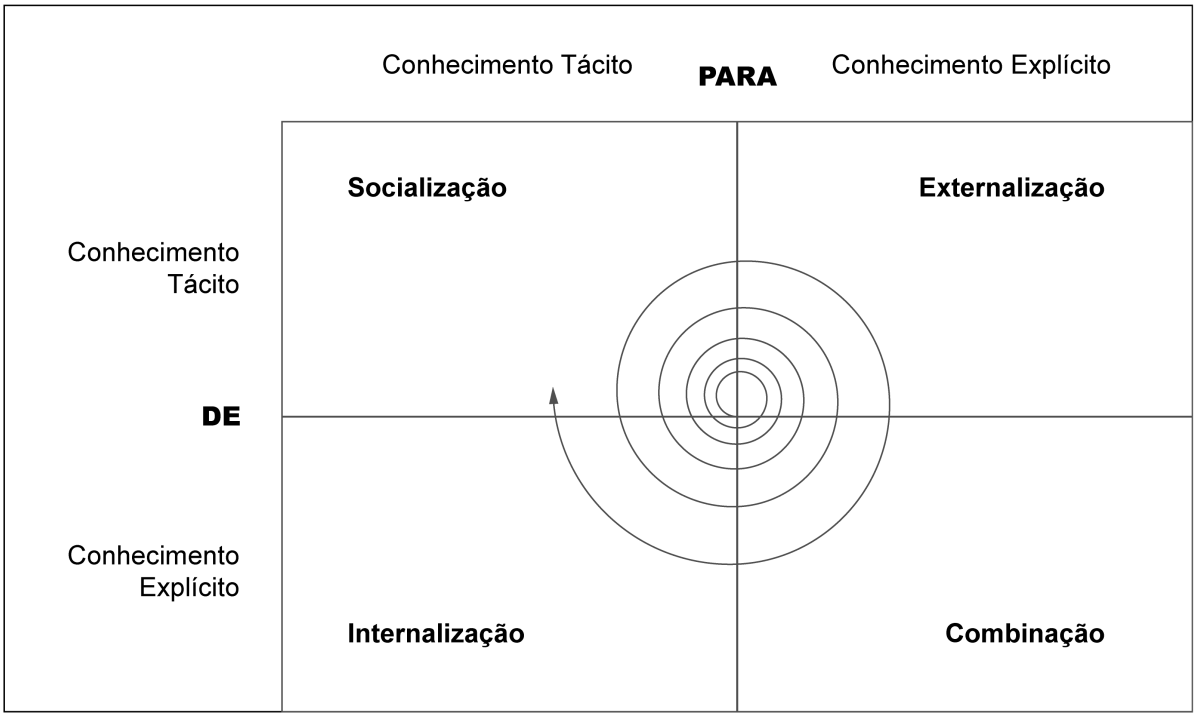

Figura 2 - Modos de conversão do conhecimento - Adaptado de (Zambalde e Alvez, 2004)

2.2 MODELOS DE MATURIDADE

Entendendo a capacidade como a aplicação do conhecimento produzido por informações adquiridas e transformadas, e essas como produto de dados contextualizados e dotados de significado, a maturidade pode ser compreendida como a combinação de conhecimentos e capacidades em torno de um propósito (Koehlegger et al., 2009).

Modelos de maturidade são, portanto, instrumentos utilizados para classificar as capacidades do indivíduo ou organização e definir ações para aumentar o nível de maturidade.

Segundo (Koehlegger, Maier e Thalmann, 2009), à época foram identificados 74 modelos de maturidade relacionados a sistema de informação ou ciência da computação. Ainda assim, esse número representava apenas uma parcela do total de modelos, já que áreas como biologia, sociologia e psicologia também fazem uso dessa ferramenta.

Para este trabalho, serão analisados alguns modelos relacionados ao desenvolvimento de software, devido à experiência do proponente no ramo. No entanto, as características estruturais desses modelos analisados podem ser estendidas a outros modelos de maturidade.

Os modelos de maturidade de software estão relacionados ao modelo inicialmente proposto por Crosby (1979) no contexto de qualidade em manufatura. Nesse modelo, o nível de maturidade pode ser medido a partir da adequação a um padrão de qualidade estabelecido.

2.2.1 Níveis de maturidade propostos por Philip Crosby

A ideia de maturidade relacionada ao conhecimento surge no contexto de Qualidade em Manufatura, com os trabalhos de Crosby (1979), que analisou os custos relacionados à má qualidade de um produto. Para ele, qualidade é definida como “conformidade com os requisitos”, ou ainda “defeito zero”.

A solução proposta por ele parte do princípio de que é necessário “fazer certo desde a primeira vez”. Para tornar isso possível, as organizações devem passar por cinco estágios de maturidade:

- Incerteza – estágio de desconhecimento total das causas da não-qualidade (“Não sabemos por que temos problemas com a qualidade.”);
- Despertar – questiona-se os problemas de qualidade, mas não há uma ação efetiva no sentido de corrigi-los (“Por que sempre temos problemas com a qualidade?”);
- Esclarecimento – ações com o intuito de resolver os problemas relacionados à qualidade são tomadas (“Estamos identificando e resolvendo nossos problemas”);
- Sabedoria – os problemas de qualidade são identificados e solucionados no início da produção (“Prevenção de defeitos é uma parte rotineira da nossa operação.”);
- Certeza – a qualidade se torna parte do processo produtivo (“Sabemos por que não temos problemas com a qualidade.”).

As definições de Crosby foram cunhadas no contexto de manufatura, mas são referência para a criação de diversos modelos de maturidade, em especial na área de software. Especificamente para o desenvolvimento de software, a meta de “defeito zero” de Crosby é utópica, uma vez que é impossível garantir a total ausência de erros em um software que possua alguma complexidade (Beck e Andres, 2004).

2.2.2 Padrões culturais de software propostos por Gerald Weinberg

No contexto de desenvolvimento de software, Weinberg (1992) faz uma adaptação das ideias de Crosby. Ele parte de duas premissas básicas:

- Não existem duas organizações exatamente iguais;
- Não existem duas organizações totalmente diferentes.

Com essas duas premissas, ele passa a considerar as diferenças entre duas organizações. Outra mudança em sua abordagem é a noção subjetiva de qualidade: para ele, a “qualidade é valor para alguma(s) pessoa(s)” (Weinberg, 1992). Portanto, algumas afirmações verdadeiras sobre a qualidade em software podem ser:

- Defeito zero é uma forma de se perceber a qualidade;
- Ter muitas funções é uma forma de se perceber a qualidade;
- Codificação elegante é uma forma de se perceber a qualidade;

- Alto desempenho é uma forma de se perceber a qualidade;
- Baixo custo de desenvolvimento é uma forma de se perceber a qualidade;
- Desenvolvimento rápido é uma forma de se perceber a qualidade;
- Facilidade para o usuário é uma forma de se perceber a qualidade.

Weinberg questiona também o uso da palavra “maturidade”. Para ele, “a palavra *maturidade* não é um fato, mas um julgamento” (Weinberg, 1992).

A palavra “maduro” vem do latim *maturus* e significa atingir a última fase do crescimento e desenvolvimento natural. Para Weinberg, a progressão pelos estágios de Crosby não é exatamente “natural”, mas sim dispendiosa. Assim como a qualidade pode ser subjetiva, a maturidade também deve ser. Não existem padrões culturais mais ou menos maduros, mas sim mais ou menos “adequados*”*.

Colocadas as ponderações sobre a visão de qualidade e maturidade de Crosby, Weinberg propõe seis padrões culturais de software (Weinberg, 1992):

- Esquecido – “Nós nem sabemos que estamos realizando um processo”;
- Variável – “Nós fazemos qualquer coisa que sentimos no momento”;
- Rotina – “Nós seguimos nossas rotinas (exceto quando em pânico)”;
- Direção – “Nós escolhemos entre nossas rotinas segundo os resultados que elas produzem”;
- Antecipação – “Nós estabelecemos rotinas baseados em nossa experiência prévia com elas”;
- Congruência – “Todos estão envolvidos na melhoria de tudo o tempo todo”.

2.2.3 Níveis de capacidade propostos pela norma ISO/IEC 15504 (atual série ISO/IEC 33000)

Desenvolvida desde 1993 pela ISO em conjunto com o projeto SPICE (*Software Process Improvement and Capability Determination*), a norma ISO/IEC 15504 foi inicialmente publicada em outubro de 2003. Recentemente esta norma foi atualizada e transformada na série ISO/IEC 33000 (NORMAS ISO, 2019). Ambas estão estruturadas em duas dimensões: a dimensão de processos e a dimensão de capacidade desses processos.

A dimensão de processos define os processos Primários, Organizacionais e de Apoio que compõem a avaliação de uma organização.

Na dimensão de capacidade de processos, a capacidade dentro de um processo é avaliada em seis níveis crescentes, desde o nível inferior, 0, até o nível superior, 5 (Salviano, 2003):

- Nível 0 – Incompleto, no qual o processo não está implementado ou não é funcional, ou seja, não atinge seus objetivos;
- Nível 1 – Executado, no qual os objetivos do processo são alcançados de alguma maneira;
- Nível 2 – Gerenciado, no qual o processo é executado de maneira planejada e controlada;
- Nível 3 – Estabelecido, no qual o processo, além de executado e gerenciado, é instanciado a partir de um processo padrão definido;
- Nível 4 – Previsível, no qual o processo passa a ser executado dentro de limites quantitativos bem definidos, com acompanhamento estatístico;
- Nível 5 – Otimizado, no qual o processo previsível pode ser aprimorado continuamente.

As normas ISO/IEC 15504 e 33000 definem, ainda, as dimensões que devem ser avaliadas em uma organização e os critérios de avaliação para cada dimensão (Salviano, 2003), porém estes aspectos não foram considerados relevantes para este trabalho.

2.2.4 Os níveis de maturidade do CMMI

O *Capability Maturity Model Integration* (SOFTWARE ENGINEERING INSTITUTE, 2010), mais conhecido como CMMI, foi elaborado pelo Software Engineering Institute da Carnegie Mellon University e é atualmente gerenciado pelo CMMI Institute, uma organização da ISACA (*Information Systems Audit and Control Association*).

Assim como a ISO/IEC 15504, ele define práticas de referência para a maturidade em domínios específicos, como desenvolvimento de produtos, realização de serviços e gestão de fornecimentos.

O CMMI Versão 1.3[1] tem duas representações distintas: a representação contínua e a representação por estágios.

Na representação contínua, semelhante à ISO/IEC 15504, há os mesmos seis níveis de capacidade e cada processo pode estar em um nível diferente, dependendo do interesse ou necessidade da organização.

Já na representação por estágios, há cinco diferentes níveis pré-determinados de maturidade. A sequência dos níveis não deve ser desconsiderada, já que um nível serve como habilitador para o próximo. Em função disso, mesmo que a maioria dos processos de uma organização esteja no nível três, por exemplo, caso existam processos no nível dois, a organização ainda será considerada como estando no nível dois.

Os níveis de maturidade definidos pela CMMI são:

- Nível 1: Inicial, *Ad-hoc*, no qual os processos são imprevisíveis, pouco controlados e reativos;
- Nível 2: Gerenciado, no qual os processos são caracterizados a cada projeto, e as ações relacionadas são frequentemente reativas;
- Nível 3: Definido, no qual os processos são caracterizados para a organização como um todo, gerando instâncias por projeto, e as ações relacionadas são frequentemente proativas;
- Nível 4: Quantitativamente gerenciado, no qual os processos são medidos e controlados estatisticamente sendo, portanto, previsíveis;
- Nível 5: Em otimização, que enfatiza a melhoria contínua dos processos quantitativamente gerenciados.

2.2.5 Modelo de Maturidade DevOps

No universo do desenvolvimento de software contemporâneo, o *DevOps* é a combinação de culturas, práticas e produtos a fim de garantir entregas de software mais rápidas, com menos falhas, mais assertivas e seguras em ambiente produtivo (Kim et al., 2018).

[1] Existe uma nova versão do CMMI, denominada CMMI 2.1, mas não está sendo considerada neste trabalho devido a não possui material disponível de forma ampla.

A palavra *DevOps* vem da junção entre *Development* e *Operations* e representa a união entre os times que desenvolvem e os times que operam e suportam um software em produção.

Estudos estatísticos realizados por Forsgreen et al. (2018) mostram que organizações que possuem sólida aplicação de práticas *DevOps*:

- Reduzem o tempo de desenvolvimento e entrega em produção de uma funcionalidade solicitada pelo time de negócios;
- Aumentam a frequência de novas versões de um software;
- Reduzem o tempo para recuperação em caso de falhas do software;
- Reduzem o índice de falhas no desenvolvimento do software.

A implantação de uma cultura *DevOps* em uma organização passa pela implantação de diferentes práticas nos processos de desenvolvimento e operação de um software (Mendes, 2016).

Para orientar essa implantação, Marco Mendes (2016) propõe um Modelo de Maturidade com oito práticas e cinco níveis de maturidade em cada prática.

Os cinco níveis de maturidade são:

- Maturidade 1 – Inicial
    - “Ausência da prática ou iniciativas *ad hoc* realizadas isoladamente por algumas pessoas”;
    - Entregas de software em produção em bases semestrais ou anuais são comuns;
    - Retrabalhos acima de 30% ocorrem com frequência;
    - Existem conflitos entre desenvolvimento, qualidade e operações.
- Maturidade 2 – Consciente
    - “Prática embrionária no time ou organização, com resultados iniciais e ainda inconsistentes”;
    - Inicia-se a busca por redução de tempo dos ciclos e retrabalhos;
    - Ações iniciais são implantadas em projetos pilotos com ferramentas de automação;
    - Times de desenvolvimento, qualidade e operações aproximam-se.
- Maturidade 3 – Gerenciado
    - “Prática sistematizada pelo time ou organização, com resultados de negócio começando a se tornarem visíveis pelo corpo gestor”;

    - Práticas estão em curso em projetos importantes;
    - Resultados de negócio são mais visíveis;
    - Retrabalhos reduzem para 15 a 30%;
    - Entregas em homologação ocorrem em base semanal e entregas em produção ocorrem em base mensal.
- Maturidade 4 – Avançado
    - “Prática otimizada pelo time, que se torna parte da cultura da organização, com resultados de negócio sempre visíveis pelo corpo gestor”;
    - Práticas estão em curso na maioria dos projetos;
    - Existe forte automação das práticas por meio de ferramentas;
    - Novos projetos e produtos já nascem com o uso das práticas de *DevOps.*
- Maturidade 5 – Melhoria Contínua
    - “Prática em melhoria contínua, utilizada por um time que possui excelência em sua realização, que foi completamente assimilada na cultura da organização”;
    - A excelência no uso das práticas é atingida;
    - Retrabalhos caem para menos de 15%;
    - Entregas em homologação acontecem em base diária e em produção em base semanal, ou menos;
    - Times de desenvolvimento, qualidade e operações trabalham em sinergia.

As oito práticas propostas por Mendes (2016) são:

- Qualidade de código
    - Diz respeito às práticas do time relacionadas à qualidade do código fonte do software.
- Infraestrutura como Código
    - Diz respeito às práticas do time relacionadas à configuração dos ambientes do software por meio de código fonte.
- Gestão de *Builds*

    - *Build*, no contexto de desenvolvimento de software, é a versão compilada do código fonte. Essa prática diz respeito ao processo de compilação do código fonte utilizado pelo time.
- Gestão de *Releases*
    - Em desenvolvimento de software, *release* é o processo de liberação de um software para o usuário final. Essa prática diz respeito ao processo de entrega do software compilado em ambientes de produção.
- Gestão de Testes
    - Em software, testes podem ser automatizados por meio de código fonte. Essa prática diz respeito ao nível de maturidade do time quanto a testes automatizados.
- Testes de Carga, Estresse e Injeção de Falhas
    - Testes de Carga, Estresse e Injeção de Falhas são testes cujo propósito é avaliar o comportamento e o desempenho do software em situações adversas. Essa prática diz respeito tanto ao nível de maturidade quanto aos testes de desempenho (*performance*).
- Gestão de Configuração
    - Diz respeito às práticas do time relacionadas à administração das alterações no software.
- Monitoração de Aplicações
    - Diz respeito às práticas utilizadas pelo time para monitorar e analisar softwares em ambiente de produção.

Para representar o avanço em todas as práticas, Mendes propõe a utilização de um gráfico do tipo radar, como ilustrado na Figura 3. Nessa figura, ele mostra em azul uma empresa XYZ que está iniciando a prática de *DevOps*, e em vermelho uma outra ABC que demonstra possuir alto nível de maturidade em *DevOps*.

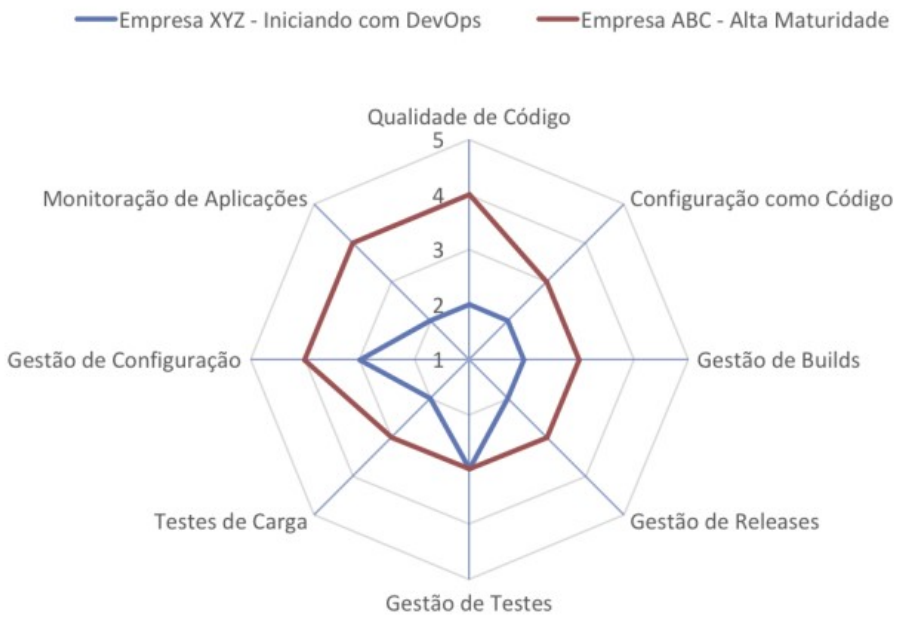

Figura 3 - Modelo de Maturidade DevOps - Fonte (Mendes, 2016)

O modelo de maturidade proposto por Mendes será utilizado como objeto de estudo para validar a proposta de representação por meio de grafos de evolução da maturidade deste trabalho.

2.2.6 O modelo de representação do avanço do conhecimento SHU-HA-RI

A maioria dos modelos de evolução da maturidade disponíveis na literatura, incluindo os aqui apresentados, possuem uma estrutura em que os processos analisados seguem um caminho com as etapas:

1. O processo não é realizado ou acontece aleatoriamente (“Eu não faço ideia do que estou fazendo.”);
2. O processo é realizado de forma instintiva (“Estou apenas fazendo.”);
3. O processo, além de realizado, passa a ser gerenciado (“Entendo o que estou fazendo.”);
4. O processo passa a ser descrito e medido para encontrar melhores práticas (“Posso descrever e quantificar o que estou fazendo.”);
5. O processo ideal é definido e passa a ser executado (“Conheço a melhor forma de fazer o que estou fazendo.”);

6. O processo ideal é executado repetidamente e passa a ser aprimorado (“Estou sempre melhorando o que estou fazendo.”).

Esses seis passos podem ser comparados ao Shu-Ha-Ri nas artes marciais japonesas, especialmente o Aikido (Fowler, 2014).

Nesse modelo de representação do avanço do conhecimento, as três etapas são definidas da seguinte forma:

- Shu – estágio inicial, em que o aluno segue os ensinamentos do mestre com precisão;
- Ha – nesse estágio, o aluno passa a aprender os princípios que norteiam a técnica;
- Ri – no último estágio, o aluno já não está mais aprendendo com um mestre, ele já compreende os princípios e é capaz de criar sua própria abordagem.

É possível inferir um paralelo entre o Shu-Ha-Ri e os modelos de maturidade estudados. Nessa comparação, o Shu se relaciona com as etapas iniciais de um modelo; o Ha se relaciona com as etapas intermediárias, e o Ri com as etapas avançadas desse modelo.

2.3 CYNEFIN FRAMEWORK

Enquanto trabalhavam para a IBM Global Services, Kurtz e Snowden (2003) propuseram o Framework Cynefin, uma estrutura de compreensão e classificação de sistemas para auxiliar gestores e líderes na tomada de decisão dependendo de sua previsibilidade (Snowden e Boone, 2007). A palavra Cynefin é de origem galesa e pode ser interpretada como “habitat” (University of Wales, 2019) ou ainda “lugar a que se pertence”.

O Framework Cynefin sugere que domínios diferentes exigem respostas diferentes, e para distinguir os domínios ele analisa as relações da causa e efeito dos eventos que neles ocorrem. Compreender como essas relações acontecem é importante para selecionar as ferramentas adequadas para se trabalhar no sistema em questão. Dessa forma, o Cynefin não “fornece” uma solução, ele “sugere” uma melhor abordagem para encontrar a solução.

Inicialmente, o Cynefin distingue os sistemas em três grupos: os ordenados, os complexos e os caóticos. Em sistemas ordenados, as relações entre causa e efeito existem, são repetíveis e podem ser previstas. Em sistemas complexos, essas relações continuam existindo, mas não podem ser previstas. Em sistemas caóticos, não existem relações entre causa e efeito, ou elas não podem ser determinadas.

Após essa definição inicial, Kurtz e Snowden (2003) separam os sistemas ordenados em “sistemas simples” e “sistemas complicados”, e criam o domínio do desordenado para sistemas que não podem ser encaixados em nenhum dos domínios, permanecendo, portanto, com cinco diferentes domínios: Simples/Óbvio, Complicado, Complexo, Caótico e Desordenado, como ilustra a Figura 4.

Os domínios são diferentes de categorias, uma vez que os sistemas podem estar próximos de uma ou de outra fronteira; e, eventualmente, podem migrar entre os diferentes domínios à medida que são mais bem compreendidos ou que seus modelos são aprimorados (Keogh, 2018).

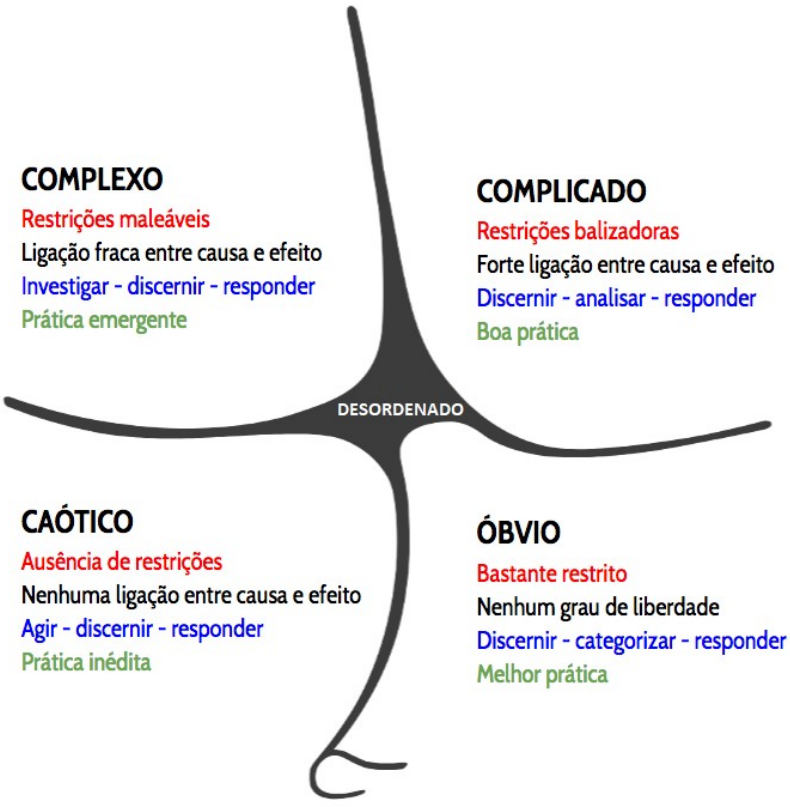

Figura 4 - Cynefin Framework - Fonte (Crescente, 2018)

No domínio do Simples/Óbvio (*Known*) estão sistemas e contextos em que as relações de causa e efeito são repetíveis e previsíveis. O processo de decisão

acontece no formato: “sente, categoriza e responde”. Nesses contextos, podem ser aplicados processos padrão com ciclos de revisão e medidas claras.

Um exemplo de um sistema, ou situação, simples é a atualização do antivírus dos computadores em uma empresa. A equipe responsável por essa atualização já realizou essa atividade antes e pode estabelecer um plano e um cronograma para essa execução (Ribas, 2018).

No domínio do Complicado (*Knowable*), as relações de causa e efeito estão dispersas no tempo, mas ainda assim são repetíveis e podem ser analisadas. O processo de decisão acontece no formato: “sente, analisa e responde”. Para esses contextos, técnicas analíticas e reducionistas são bons guias nas tomadas de decisões.

A construção de um dispositivo de GPS, *Global Position System* é um exemplo de um sistema, ou situação, complicado. Apesar de envolver conceitos e tecnologias avançados, os requisitos e o público alvo do produto são bem conhecidos. Um grupo de engenheiros com conhecimentos específicos é capaz de construir o dispositivo com algum esforço (Ribas, 2018).

No domínio do Complexo, há muitas possibilidades, as relações de causa e efeito só mostram coerência em uma análise retrospectiva. O processo de decisão acontece no formato: “experimenta, sente e responde”. Para decisão nesses contextos, são necessárias múltiplas, pequenas e distintas intervenções baseadas em gerenciamento de padrões, filtros de perspectiva e análise de sistemas complexos adaptativos (Kurtz e Snowden, 2003).

Exemplos de sistemas no domínio do Complexo são empresas *startups* criando produtos inexistentes ou inovadores no mercado. Nesses sistemas não é possível prescrever os resultados, as empresas precisam trabalhar com pequenos ciclos de descoberta e validação de seus serviços ou produtos no mercado. É um cenário de incerteza em que a solução vai sendo construída por meio desses pequenos ciclos (Ribas, 2018).

No domínio do Caótico, os sistemas e contextos não apresentam nenhuma relação entre causa e efeito em um nível sistêmico. O processo de decisão acontece no formato: “age, sente e responde”. Nesses contextos, são possíveis apenas pequenas intervenções direcionadas a estabilizar a situação; ou eventualmente, nem mesmo essas pequenas intervenções.

Para o domínio do Caótico, um exemplo seria uma casa pegando fogo. Uma pessoa sozinha e sem condições de apagar o incêndio se preocupa apenas em sair da casa, não há tempo de planejar uma rota de fuga (Ribas, 2018).

As transições entre os domínios podem acontecer de diferentes formas e por diferentes razões.

A migração do Simples/Óbvio para o Caótico acontece a partir de um "colapso assimétrico". Os sistemas são tratados com relações simétricas e estáveis, ignorando as mudanças na dinâmica do ambiente, até que é tarde demais e um colapso acontece (Fierro et al., 2017). Já a transição do Caótico para o Simples/Óbvio acontece a partir de um movimento de imposição externo ao sistema criando uma estabilidade rígida.

A fronteira entre o Simples/Óbvio e o Complicado é onde o método científico atua. Melhorias incrementais em teorias levam sistemas de um lado para o outro dessa fronteira, ora aumentando a forma como são compreendidos, ora simplificando suas representações (Fierro et al., 2017)

A fronteira entre o Complicado e o Complexo é atravessada por meio de dois movimentos. Do Complicado para o Complexo acontece a "Exploração", quando novos experimentos são colocados à prova. Já a "Transferência *Just In Time*" acontece quando se faz o movimento contrário, ao selecionar padrões estáveis do espaço complexo e tentar estabilizá-los à medida que são necessários (*just in time*) (Fierro, Putino, & Tirone, 2017).

A fronteira entre o Complexo e o Caótico também é difícil de ser delineada. O movimento do Caótico para o Complexo acontece através de movimentos de "Enxames", pequenas iniciativas com indícios de padrão. A transição do Complexo para o Caótico acontece por movimentos cíclicos de divergência e convergência (Fierro, Putino, & Tirone, 2017).

As ferramentas atualmente utilizadas para representar os sistemas de evolução da maturidade consideram as relações entre causa e efeito lineares e repetíveis, à medida que sequenciam as capacidades/conhecimentos em um caminho linear pré-definido, não admitindo variabilidade e desconsiderando as relações emergentes. Isso define uma "melhor prática", característica do domínio do Simples/Óbvio dentro do Framework Cynefin.

O que este trabalho propõe é representar os sistemas de evolução da maturidade por meio dos grafos, possibilitando uma visão sistêmica mais abrangente

das relações entre os elementos (capacidades/conhecimentos) desses sistemas. Dessa forma, as relações de causa e efeito são menos restritivas, permitindo diferentes caminhos na evolução da maturidade, definindo “boas práticas”, característica do domínio do Complicado no Framework Cynefin.

O proponente não entende essa mudança como uma mudança para o domínio do Complexo, uma vez que as estruturas representadas terão um número restrito de elementos e suas relações serão pré-estabelecidas. Aumentar a granularidade e permitir relações dinâmicas entre os elementos poderia levar essa representação para o domínio do Complexo.

2.4 TEORIA DE GRAFOS

Para este trabalho, será utilizada a Teoria de Grafos como fundamentação matemática. A seguir, os principais conceitos utilizados neste trabalho são apresentados.

Um grafo é composto por um conjunto de vértices e por um conjunto de arestas. Um vértice pode representar um objeto simples e unitário qualquer, podendo ter nome e outros atributos. Uma aresta representa a relação entre dois vértices (Ziviani, 2011).

Conforme ilustrado na Figura 5, um grafo dirigido **G** é um par **(V,E)**, em que:

- **V** é um conjunto finito de elementos, denominado “conjunto de vértices”.
- **E** é uma relação binária entre elementos de **V**, denominado “conjunto de arestas”. As arestas em um grafo dirigido estão “direcionadas” em um único sentido, de um vértice de origem para um vértice de destino[2].

[2] Grafos em que as arestas não são “direcionadas” são chamados de grafos não-dirigidos. Neste trabalho serão utilizados grafos dirigidos.

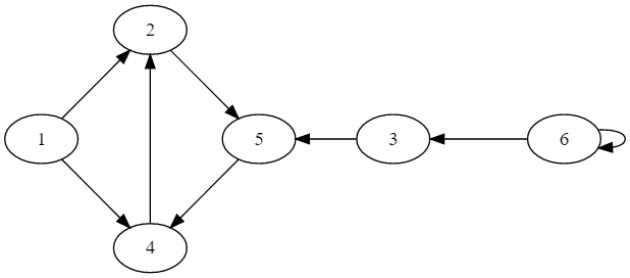

Figura 5 - Grafo dirigido - Adaptado de (Cormen, 2012)

Nos grafos de evolução do conhecimento, as capacidades serão representadas como vértices e as relações de precedência entre elas serão representadas como arestas dirigidas.

Para um grafo dirigido, existem dois tipos de graus em um vértice. O número de arestas que chegam ao vértice é definido como *in-degree*, grau de entrada, e o número de arestas que saem do vértice é definido como *out-degree*, grau de saída.

Na Figura 5, o vértice de número 5, por exemplo, tem grau de entrada 2 (chegam nele arestas dos vértices 3 e 2) e grau de saída 1 (dele sai uma aresta para o vértice 4).

Um caminho em um grafo dirigido pode ser compreendido como uma sequência de vértices que ligam um vértice inicial a um vértice final. Na Figura 5, o caminho entre o vértice 1 e o vértice 5 é o conjunto {1, 2, 5}. Vale destacar que não há um caminho entre o nó 5 e o nó 1.

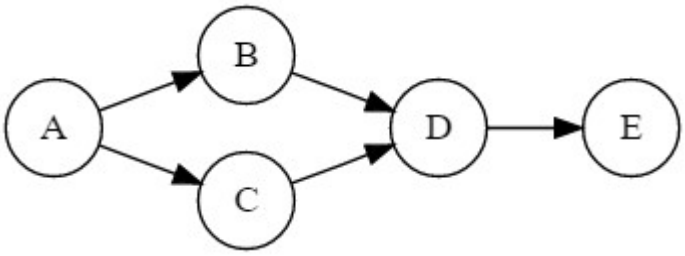

Figura 6 - Grafo dirigido acíclico - Fonte Própria (2019)

Os grafos utilizados neste trabalho serão dirigidos acíclicos, ou seja, seus caminhos não possuem vértices repetidos, não há formação de ciclos. O grafo na Figura 6 é um grafo acíclico. Na Figura 7 foi inserida uma aresta entre os vértices D e A, tornando o grafo cíclico com, no mínimo, os ciclos {A, C, D, A} e {A, B, D, A}.

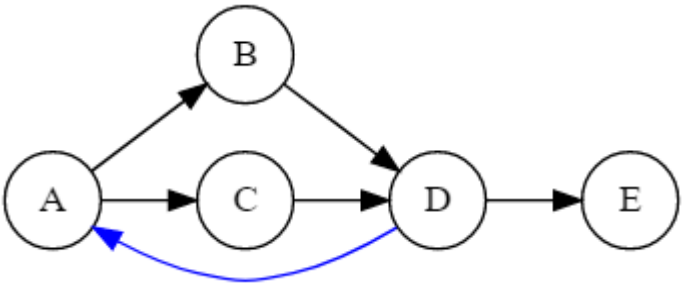

Figura 7 - Grafo dirigido cíclico - Fonte Própria (2019)

Para grafos dirigidos acíclicos, o conceito de ordenação topológica representa a sequência em que, para todo vértice **V**, todos os vértices para os quais **V** tem aresta de saída aparecem depois de **V**.

Para ilustrar este conceito, uma pessoa ao se vestir pela manhã deve vestir algumas peças da roupa antes de outras, para algumas peças a ordem não é importante. A Figura 8 mostra as relações de precedência. Na Figura 9 o mesmo grafo é apresentado após a ordenação topológica, que sugere uma ordem para vestir as peças, estando todas as setas apontando para a direita. A ordenação topológica neste trabalho poderá ser utilizada para definir um sequenciamento possível entre as capacidades do modelo representado.

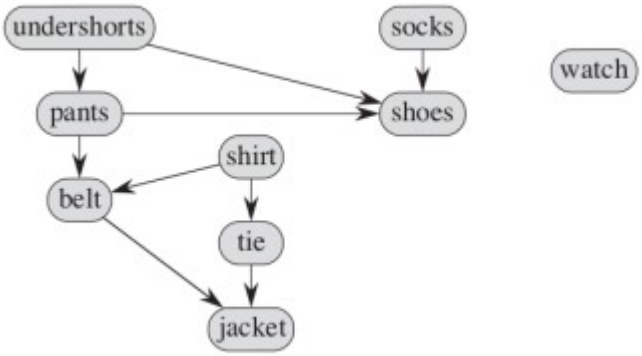

Figura 8 – Grafo dirigido acíclico sem ordenação topológica - Fonte (Cormen, 2012)

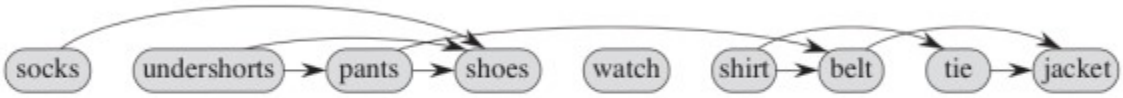

Figura 9 - Grafo dirigido acíclico após ordenação topológica - Fonte (Cormen, 2012)

Em teoria dos grafos, uma “rede de fluxo” é definida como um grafo dirigido acíclico no qual cada aresta tem uma “capacidade[3] não negativa”. Há dois vértices distintos: “fonte” e “sumidouro”. Todo vértice pertencente ao grafo se encontra em um caminho entre a “fonte” e o “sumidouro”. A Figura 10 apresenta uma Rede de Fluxo (Cormen, 2012).

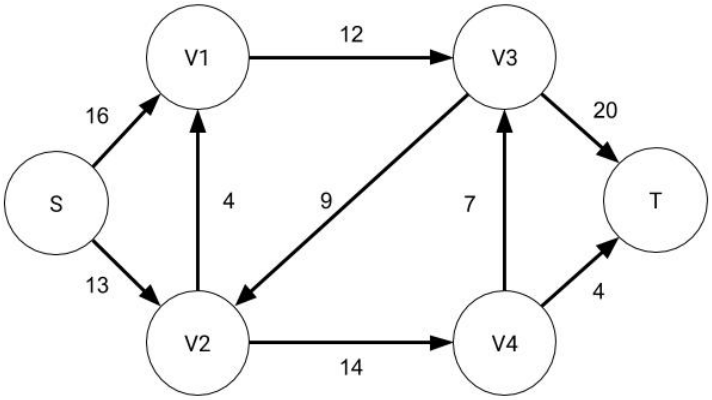

Figura 10 - Exemplo de Rede de Fluxo - Adaptado de (Cormen, 2012)

As “redes de fluxo” nesse trabalho podem ser utilizadas em “subgrafos“[4] definidos a partir de dois vértices do grafo para calcular o esforço existente entre uma capacidade representada no grafo e uma outra subsequente.

Em “redes de fluxo”, o “fluxo máximo” é definido como o fluxo de intensidade máxima que respeita os limites de capacidade dos vértices.

[3] O termo “capacidade” utilizado aqui não deve ser confundido com o mesmo termo utilizado no contexto de “níveis de capacidade”**.** Ele foi mantido nesse contexto porque assim está definido na literatura referenciada.

[4] Um subgrafo SG do grafo G é um grafo cujo conjunto de vértices é um subconjunto do conjunto de vértices do grafo G.

A Figura 11 ilustra esse conceito. O vértice S é a fonte e o vértice T, o sumidouro. Os valores inteiros próximos às arestas na primeira imagem representam as capacidades máximas de cada aresta.

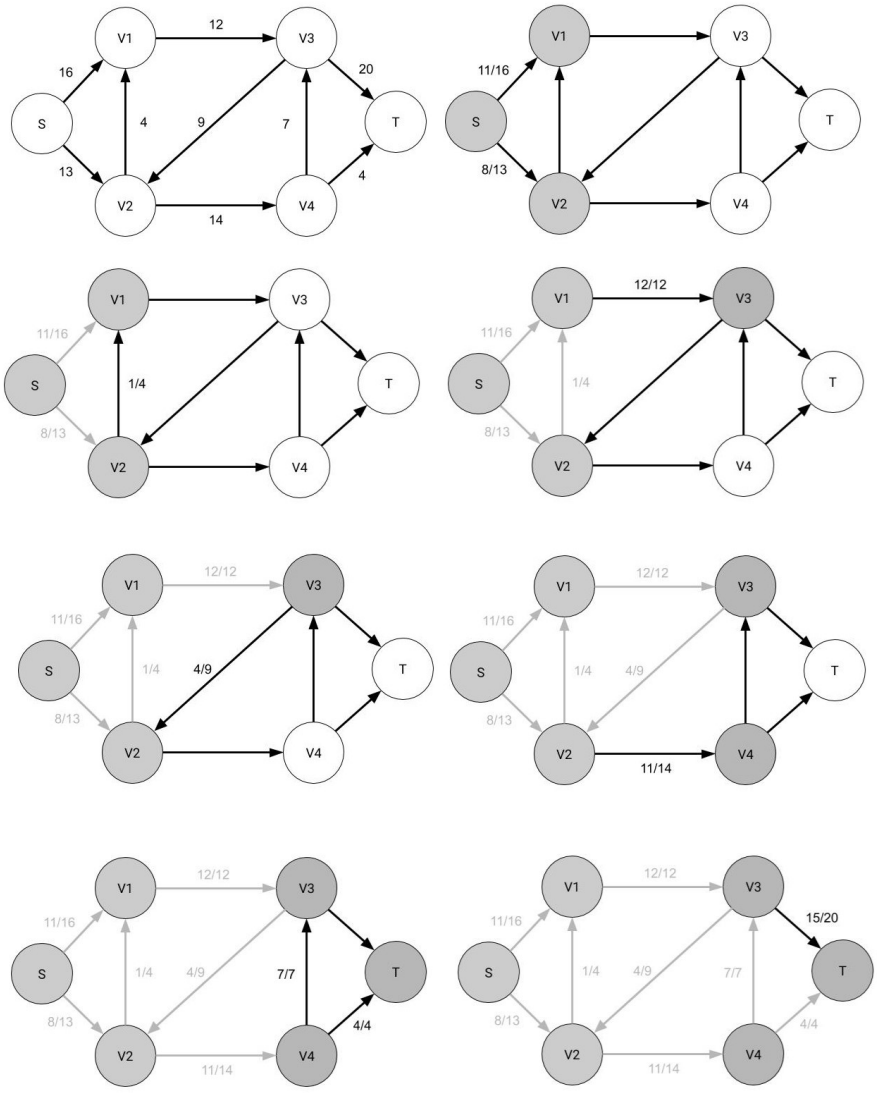

Figura 11 - Fluxo Máximo - Adaptado de (Cormen, 2012)

O cálculo do fluxo a cada iteração no grafo é detalhado na Tabela 1. Como pode-se notar, o fluxo máximo acontece na iteração 1, e tem valor de 19 unidades.

Tabela 1 - Cálculo de fluxo máximo

			Estoque a cada iteração					
Iteração	**Fluxo**	**Origem -> Destino (Unidades)**	**S**	**V1**	**V2**	**V3**	**V4**	**T**
1	19	**S->V1 (11)**	-	11	8	0	0	-

		S->V2 (8)						
2	1	**V2->V1 (1)**	-	12	7	0	0	-
3	12	**V1->V3 (12)**	-	0	7	12	0	-
4	4	**V3->V2 (4)**	-	0	11	8	0	-
5	11	**V2 -> V4 (11)**	-	0	0	8	11	-
6	7	**V4 -> V3 (7)**	-	0	0	15	0	4
7	4	**V4 -> T (4)**						
8	15	**V3 -> T (15)**	-	0	0	0	0	19

2.5 TRABALHOS RELACIONADOS

A representação do conhecimento por meio de grafos já é utilizada em outras aplicações, porém com objetivos diferentes do proposto por esse trabalho.

O SKOS, *Simple Knowledge Organization System*, define uma forma padrão para representar sistemas de organização do conhecimento baseando-se no RDF, *Resource Description Framework*. Seu propósito é compartilhar e associar sistemas de organização do conhecimento pela Web (Miles e Bechhofer, 2009).

O SKOS é utilizado como ferramenta por outros sistemas de organização, como tesauros, taxonomias, sistemas de classificação e de cabeçalhos de assunto, para permitir o compartilhamento de dados em diferentes aplicativos.

O RDF é uma linguagem de representação de informações sobre recursos voltada para a Web. Seu objetivo principal é representar metadados sobre recursos da Web, como data e autor de uma página, informações de direitos autorais e licenciamentos (Manola e Miller, 2004). No entanto, sua estrutura generalista pode ser adaptada e utilizada para representar “coisas” que podem ser identificadas na Web. A Figura 12 mostra um grafo descrevendo Eric Miller, um dos autores da referência bibliográfica sobre RDF.

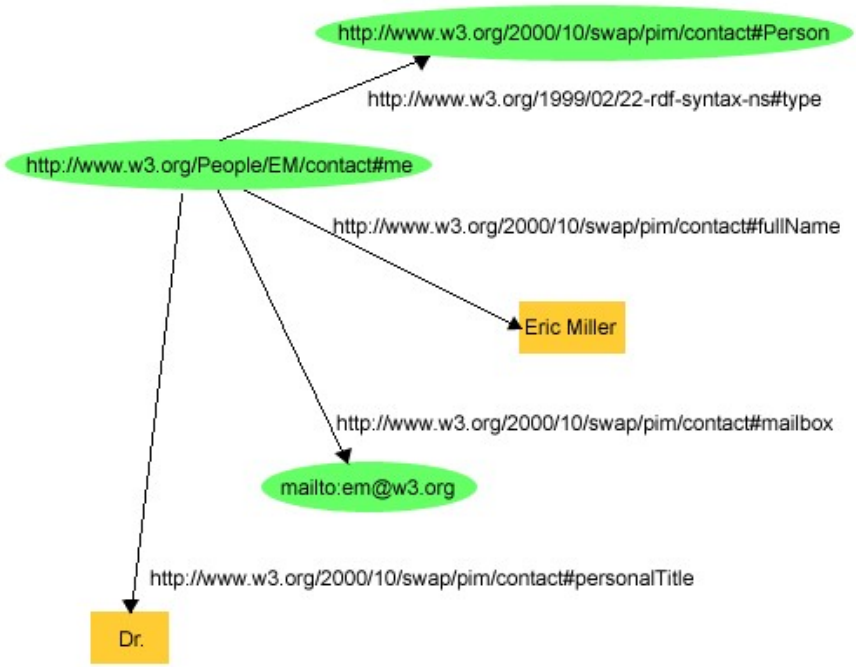

Figura 12 - Grafo RDF descrevendo Eric Miller - Fonte (Manola e Miller, 2004)

O RDF se baseia em três conceitos, que formam uma Tripla RDF:

- Sujeito, a “coisa” descrita pela declaração RDF;
- Predicado, a propriedade específica do sujeito;
- Objeto, o valor da propriedade.

O sujeito e o objeto são representados como vértices e o predicado como aresta de um grafo, como mostrado na Figura 13.

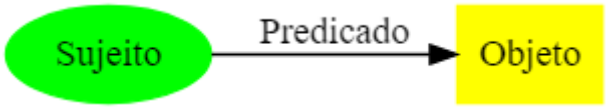

Figura 13 - Estrutura básica de um Grafo RDF - Fonte Própria (2019)

Um exemplo da utilização do RDF para a frase “A página http://www.example.org/index.html foi escrita por **John Smith** em **inglês** no dia **16 de agosto de 1999**” é o conjunto de sentenças abaixo, representadas pelo grafo na Figura 14. São elas:

- A página http://www.example.org/index.html tem um criador cujo valor é John Smith.
- A página http://www.example.org/index.html tem uma data de criação cujo valor é 16 de agosto de 1999

- A página http://www.example.org/index.html tem uma linguagem cujo valor é inglês.

Nas sentenças é possível observar o padrão: o **objeto** tem a **propriedade** cujo valor é **valor.**

A representação por meio de um grafo RDF combina diferentes sentenças (triplas RDF) podendo o objeto de uma sentença ser sujeito em outra.

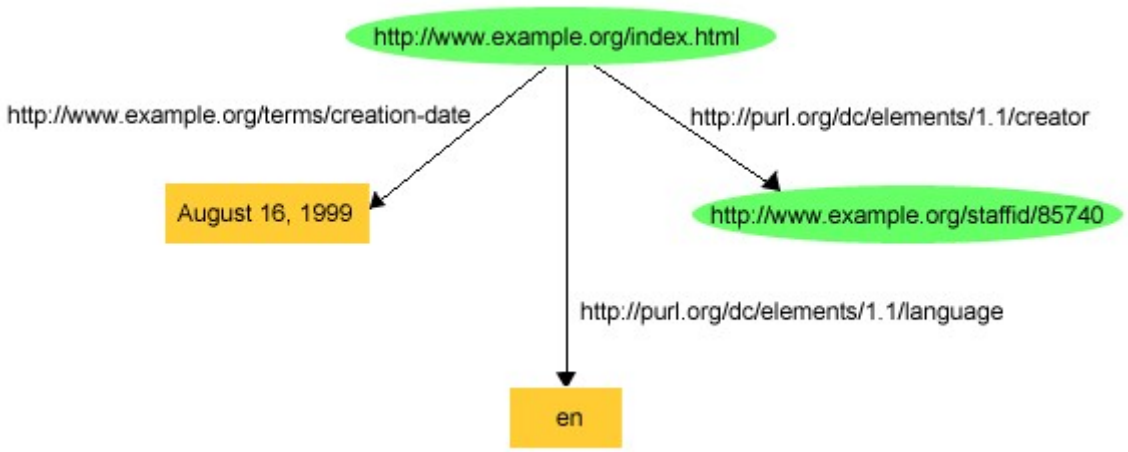

Figura 14 - Conjunto de declarações sobre o mesmo recurso - Fonte (Manola e Miller, 2004)

Como pode ser observado, o SKOS e o RDF aplicam a teoria de grafos na representação do conhecimento com o propósito de descrever as relações entre sujeitos e objetos. A proposta deste trabalho é diferente, à medida que pretende representar as relações de precedência entre capacidades a fim de descrever como se dá sua evolução.

Além dessas duas ferramentas, não foram encontrados outros trabalhos que utilizam grafos explicitamente para representar conhecimento, capacidade ou maturidade.

3 METODOLOGIA

Para a elaboração deste trabalho foram realizados estudos a fim de dominar os principais conceitos relacionados à teoria de grafos. Estes conceitos foram utilizados para estruturar a representação dos modelos de maturidade por meio dos grafos e seus resultados foram apresentados na seção 2.4.

Foram analisados também modelos de maturidade diversos com o objetivo de compreender suas estruturas e como os grafos devem ser modelados a fim de representá-los. Este levantamento foi apresentado na seção 2.2.

Para justificar a utilização dos grafos, a distinção entre os domínios do simples e do complicado foi definida, como apresentado na seção 2.3, utilizando o Framework Cynefin.

Após estes estudos, os passos para representação dos modelos de maturidade utilizando grafos foram definidos, como apresentado na seção 4.1.

De posse dessa definição, a ferramenta proposta foi implementada – como explicado na seção 4.4 – utilizando boas práticas de desenvolvimento, tais como: integração contínua – seção 4.4.3 –, automação de testes – seção 4.4.7 – e revisão automatizada de código – seção 4.4.6.

Durante a implementação foi utilizada uma abordagem iterativa e incremental, detalhada na seção 4.3.

Além disso, a aplicação foi disponibilizada para utilização de eventuais interessados em ambiente web, conforme descrito na seção 4.4.2

Ao longo de seu desenvolvimento, a ferramenta foi verificada e validada utilizando modelos de evolução de conhecimento e de maturidade conhecidos, como a grade de um curso de graduação, apresentado na seção 4.5 e modelo de maturidade DevOps, apresentado na seção 4.6.

4 DESENVOLVIMENTO

Neste capítulo é apresentado o trabalho realizado no desenvolvimento da solução proposta, bem como a descrição funcional da aplicação com seus detalhes técnicos de implementação e com a visão geral dos algoritmos por ela executados.

4.1 MODELOS DE MATURIDADE UTILIZANDO GRAFOS

Para representar um modelo de maturidade utilizando grafos, deve-se seguir dois passos:

1. Identificar os elementos que representam as unidades do modelo;
    a. Esses elementos devem ser tão granulares quanto possível;
    b. Eles representam cada capacidade ou habilidade que deve ser adquirida para avançar dentro daquele modelo;
    c. Esses elementos serão os vértices do grafo;
    d. Para o caso do curso de graduação, os elementos são as disciplinas; para o modelo de maturidade DevOps, as habilidades dentro de cada prática.
2. Identificar as relações de dependência entre as unidades.
    a. Relações de precedência entre unidades consecutivas do modelo;
    b. A definição dessas relações deve ser feita por um especialista no tema do modelo e no próprio modelo;
    c. Esses elementos serão as arestas do grafo;
    d. Para o curso de graduação, as relações são os pré-requisitos entre as disciplinas; no modelo de maturidade DevOps, o sequenciamento entre as habilidades.

Realizados estes dois passos, o modelo de maturidade está representado em um grafo direcional acíclico. A Figura 15 mostra um exemplo de modelo de maturidade utilizando grafo.

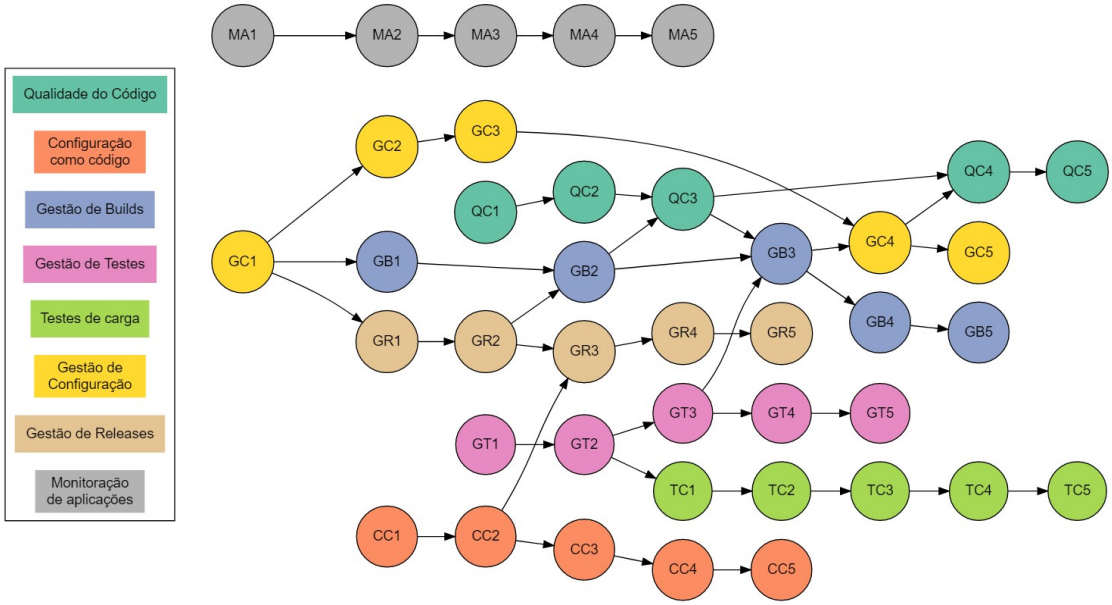

Figura 15 - Exemplo de grafo de maturidade - Fonte Própria (2019)

4.2 A APLICAÇÃO

A aplicação criada para este trabalho está hospedada *online* no endereço https://grafos-evolucao-maturidade.web.app/. Nela é possível visualizar e editar grafos de maturidade e executar alguns algoritmos de cálculo já implementados[5].

4.2.1 Funcionalidades

As funcionalidades da aplicação estão representadas na Figura 16.

- Visualizar grafo
    - Há duas formas de visualização do grafo: hierárquica e não hierárquica. A primeira visa ordenar os vértices pelo sequenciamento entre eles e a segunda visa desenhar um grafo cujas arestas tenham aproximadamente o mesmo tamanho. Na seção 4.4.5, as duas formas são mais bem detalhadas.

[5] No momento da redação deste texto, o algoritmo para seleção de próximas atividades a partir de objetivos futuros, aplicado aos modelos de maturidade DevOps não estava implementado, em função disso não foi apresentado como item da descrição funcional.

- Analisar grafo
  - O algoritmo de análise implementado até o momento calcula potenciais próximas atividades. A seção 4.7 detalha seu funcionamento.
- Editar grafo
  - A edição dos grafos pode ser feita por campos de textos em formato JSON, como mostrado na Figura 17.
  - É possível editar vértices e arestas, grupos de vértices e opções.
  - Neste trabalho foi introduzido o conceito de grupos de vértices apenas para representação visual do grafo, sendo que cada grupo é distinguido por uma cor. No entanto, essa informação não foi tratada em nenhum algoritmo até o momento, mas é considerada relevante para futuras evoluções do produto obtido.
  - Os vértices também têm os atributos “título” e “detalhes” que não considerados nos algoritmos, mas são utilizados para exibir uma mensagem quando o mouse é passado sobre os vértices no grafo e informações adicionas quando se clica sobre eles, Figura 18.
  - Em “Opções” há quatro possíveis configurações:
    - Algoritmo de desenho do grafo: hierárquico ou não;
    - Direção, utilizado apenas no modo hierárquico: para a esquerda, para a direita, para cima e para baixo;
    - Animação: sim ou não;
    - Suavização das arestas: sim ou não.

Figura 16 - Funcionalidades - Fonte Própria (2019)

```
[
 {
  "id": 1,
  "groupId": 1,
  "label": "1",
  "title": "One",
  "details": "**bold**",
  "level": 1
 },
 {
  "id": 2,
  "groupId": 2,
  "label": "2",
  "title": "Two",
  "details": "Two  \n lines",
  "level": 1
 },
 {
  "id": 3,
  "groupId": 1,
  "label": "3",
  "title": "Three",
  "details": "_italic_",
  "level": 2
 },
 {
  "id": 4,
  "groupId": 2,
  "label": "4",
  "title": "Four",
  "details": "_italic_",
  "level": 3
 },
 {
  "id": 5,
  "groupId": 1,
```

Figura 17 - Editar texto JSON - Fonte Própria (2019)

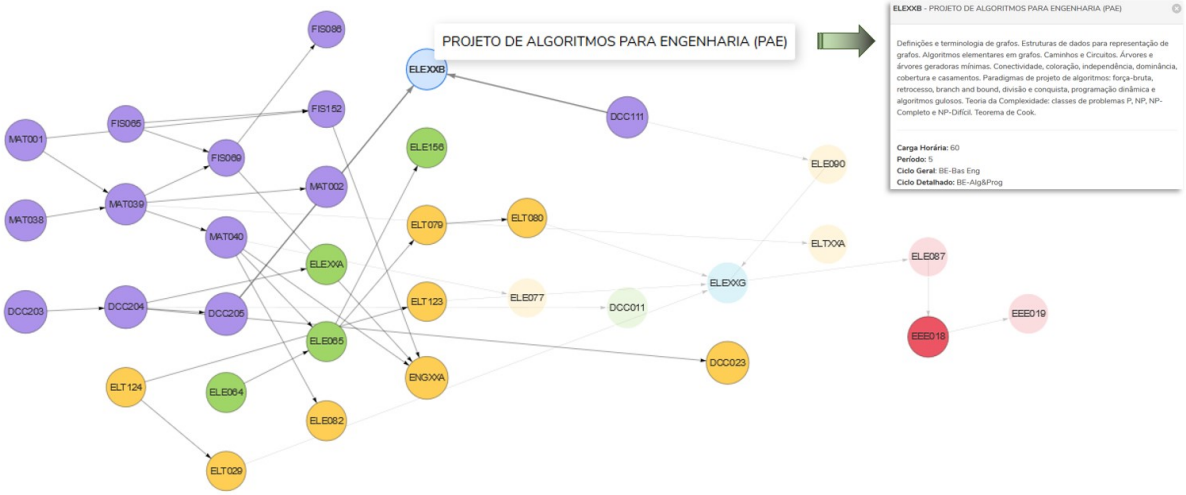

Figura 18 - Atributos "Título" e "Detalhes" - Fonte Própria (2019)

4.2.2 Modelos disponíveis

Para representar a utilização da ferramenta, foram criados três grupos de grafos: grades de Engenharia de Sistemas, variações do modelo de maturidade para

DevOps proposto por Mendes (2016) e um grafo simplificado com seis vértices e seis arestas para demonstração da ferramenta.

Para o grupo de Engenharia de Sistemas, há quatro grafos disponíveis

- Grade curricular proposta para o curso na reforma de 2019-20;
- Grade curricular versão segundo semestre de 2018 (2018-2);
- Grade curricular versão 2018-2, considerando as disciplinas cursadas pelo autor deste trabalho até o segundo semestre de 2019, semestre da elaboração deste trabalho;
- Grade curricular versão 2018-2, considerando um aluno calouro.

Para as duas últimas versões da grade 2018-2, é possível executar o algoritmo de cálculo de novas disciplinas.

Para o grupo de DevOps há três versões de grafos:

- DevOps versão 1 - neste grafo os níveis de maturidade dentro de cada disciplina DevOps são considerados como vértices e suas relações de dependências são representadas como arestas. A definição das relações de dependência foi feita pelo autor deste texto.
    - Nesta versão do modelo, há 40 vértices e 55 arestas.
- DevOps versão 2 - neste grafo, cada tópico dentro de cada nível de maturidade é considerado como um vértice e, novamente, as relações de dependências são representadas como arestas, sendo essas também definidas pelo próprio autor deste texto.
    - Nesta versão do modelo, há 97 vértices e 124 arestas.
- DevOps versão 3 - este é o mesmo grafo da versão 2, mas no entanto o desenvolvimento deste próprio trabalho foi analisado à luz desse modelo, possibilitando definir um estado de maturidade DevOps para a aplicação desenvolvida neste trabalho.
    - Das 97 habilidades do modelo, 63 foram consideradas como atingidas, definindo um avanço de 65% de maturidade DevOps com relação ao modelo proposto por Mendes (2016).

Na Figura 19 são apresentados os grafos disponíveis na aplicação.

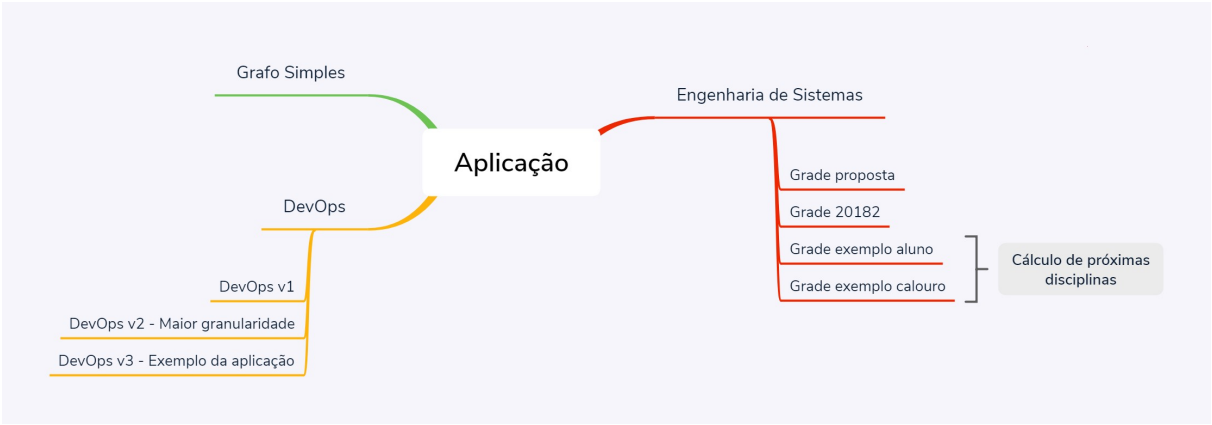

Figura 19 - Grafos disponíveis na aplicação - Fonte Própria (2019)

4.3 DESENVOLVIMENTO ITERATIVO E INCREMENTAL

Durante o desenvolvimento da aplicação descrita neste trabalho foi utilizada uma abordagem iterativa e incremental, cuja ideia central é a utilização de iterações curtas. A cada iteração, uma nova funcionalidade é entregue e *feedbacks* são colhidos (Sommerville, 2018).

As funcionalidades foram implementadas pelo autor do trabalho e validadas com a orientadora (também Coordenadora do Colegiado do Curso) durante o semestre letivo em que o trabalho foi desenvolvido.

As entregas sequenciais foram:

- Ambiente *online* e esqueleto da aplicação, i.e., configuração do servidor de hospedagem, de verificação de código e de integração contínua;
    - Dessa forma, desde a primeira publicação, a aplicação está configurada para fazer verificação automática de código e publicação automática de novas versões.
- Visualização do grafo, integração com a biblioteca Vis.js;
    - Este era um dos pontos de maior risco da aplicação, e em função disso, foi realizado primeiro.
- Edição do grafo em formato JSON;
- Grafos para a grade curricular de Engenharia de Sistemas;
- Algoritmo de cálculo de próximas disciplinas;
- Grafos para o modelo de maturidade DevOps;

4.4 IMPLEMENTAÇÃO

Uma visão geral da solução utilizada para a aplicação é apresentada na Figura 20. O código fonte está armazenado na plataforma de hospedagem de código fonte GitHub. O site, por sua vez, está hospedado na plataforma de desenvolvimento web Firebase.

Para analisar de qualidade do código fonte, foi utilizada a plataforma de inspeção e revisão automáticas com análise estática do código fonte Sonar, por meio da plataforma SonarCloud.

Para fazer a integração entre essas três plataformas, foi utilizada a ferramenta Travis CI, um serviço de integração contínua gratuito para código fonte aberto.

O código fonte foi desenvolvido utilizando a linguagem JavaScript e o Framework React. Para desenhar os grafos, foi utilizada a biblioteca Vis.js.

Essas plataformas e ferramentas são mais bem detalhadas nas próximas seções.

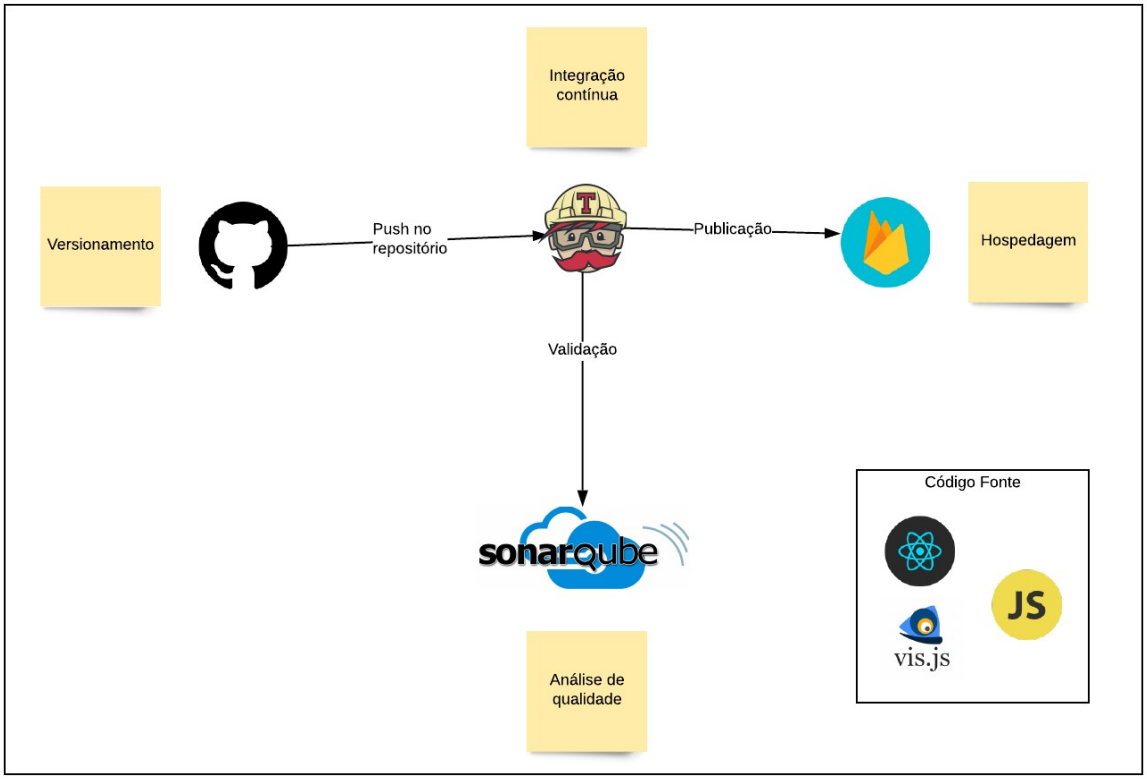

Figura 20 - Visão geral das tecnologias envolvidas na solução - Fonte Própria (2019)

4.4.1 VERSIONAMENTO

O versionamento de código fonte visa gerenciar as diferentes versões de um software por meio do controle de histórico. Suas principais vantagens são a possibilidade de trabalho em equipe e rastreabilidade do código, incluindo marcação e resgate de versões. Além disso, oferece maior segurança e confiança no armazenamento do código, uma vez que há redundância para armazenamento dos arquivos (Sommerville, 2018).

O Sistema de Controle de Versão (SCM) utilizado neste trabalho foi o Git, um sistema de controle de versão distribuído, projetado por Linus Torvalds para ser utilizado no desenvolvimento do kernel Linux. O Git é um software livre sob a licença GNU (Software Freedom Conservancy, 2019). Uma pesquisa do site Stackoverflow de 2018 mostra o Git sendo utilizado por cerca de 88% dos desenvolvedores (Stack Overflow, 2018).

O código fonte deste trabalho está hospedado na plataforma GitHub, Figura 21,que é uma implementação do Git que contava em agosto de 2019 com mais de 40 milhões de usuários e 100 milhões de repositórios (GitHub Inc., 2019). O código fonte está disponível no link https://github.com/matheusaraujo/grafos-evolucao-maturidade.

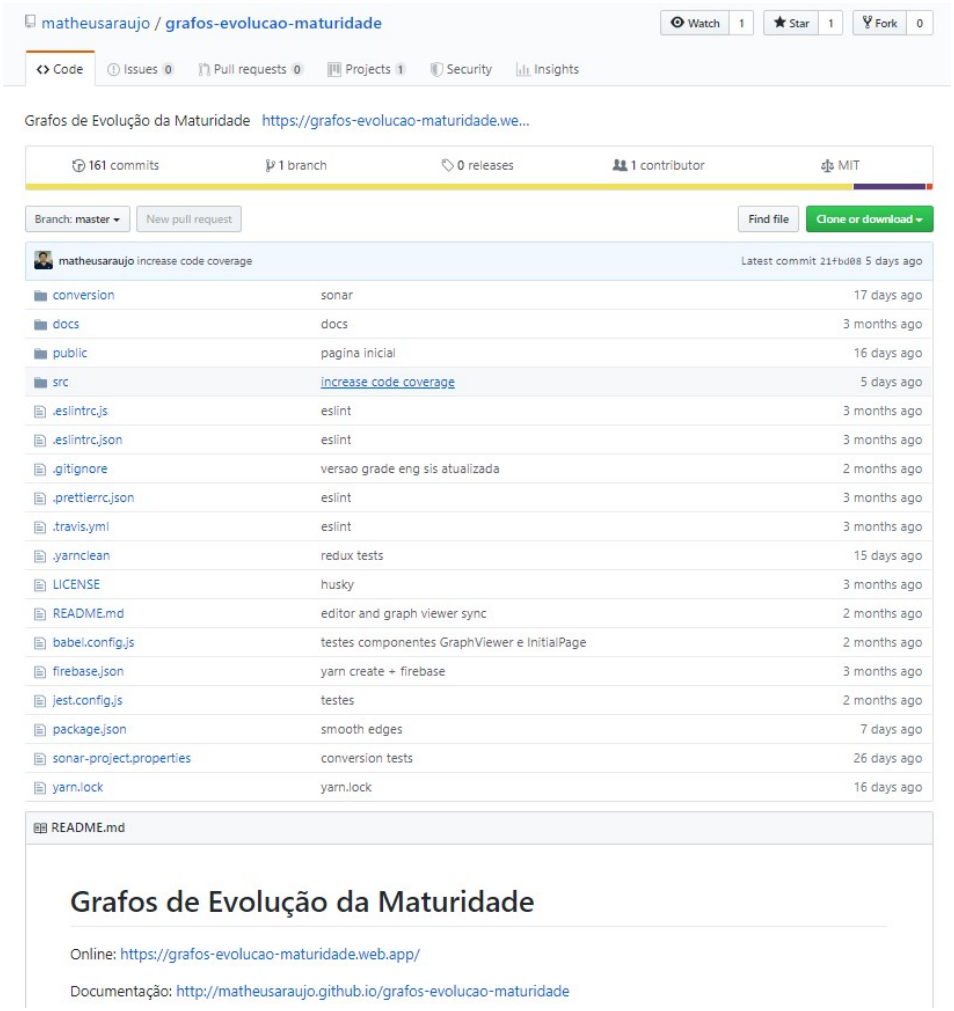

Figura 21 - Repositório no GitHub - Fonte Própria (2019)

4.4.2 HOSPEDAGEM

Hospedar um site significa armazenar os arquivos desse site em um provedor de hospedagem disponível na Internet.

Para hospedar a aplicação deste projeto, foi utilizada a plataforma de desenvolvimento móvel e web Firebase. Esta é uma plataforma mantida hoje pela Google e possui um plano gratuito no qual é possível hospedar até 1GB de dados e trafegar 10GB por mês (Firebase Inc., 2019).

A aplicação deste trabalho publicada tem cerca de 2.08MB, portanto neste plano gratuito é possível acessá-la cerca de 5000 vezes durante o mês. Esse número pode ser ainda maior, uma vez que os navegadores modernos armazenam

localmente cópias dos sites depois do primeiro acesso, portanto alguns acessos podem acontecer sem que necessariamente a aplicação seja baixada para o computador do usuário, não sendo esse cenário contabilizado no tráfego.

A Figura 22 mostra o gráfico gerado pelo próprio Firebase contendo o histórico de utilização entre os dias 6 de outubro de 2019 e 5 de novembro de 2019.

Figura 22 - Histórico de utilização da aplicação - Fonte (Firebase Inc., 2019)

4.4.3 INTEGRAÇÃO CONTÍNUA

Por integração contínua, entende-se a prática em que o código fonte de uma aplicação é constantemente levado para um repositório central, a partir do qual processos automáticos são disparados para executar validações do código fonte e criar artefatos publicáveis da aplicação, que serão então levados para um ambiente de produção (AWS, 2019).

Para este projeto, foi utilizada o Travis CI, um serviço de integração contínua hospedado no GitHub e gratuito para projetos de código aberto.

A cada nova versão do código fonte, disponibilizada no repositório central de código fonte, é disparado um processo no Travis CI que roda todos os testes automatizados, dispara a análise de código pelo do SonarQube e, caso tenha sucesso nesses dois processos, publica a aplicação no Firebase. A Figura 23 representa esse processo.

O relatório de publicação da aplicação por ser visualizado em https://travis-ci.com/matheusaraujo/grafos-evolucao-maturidade

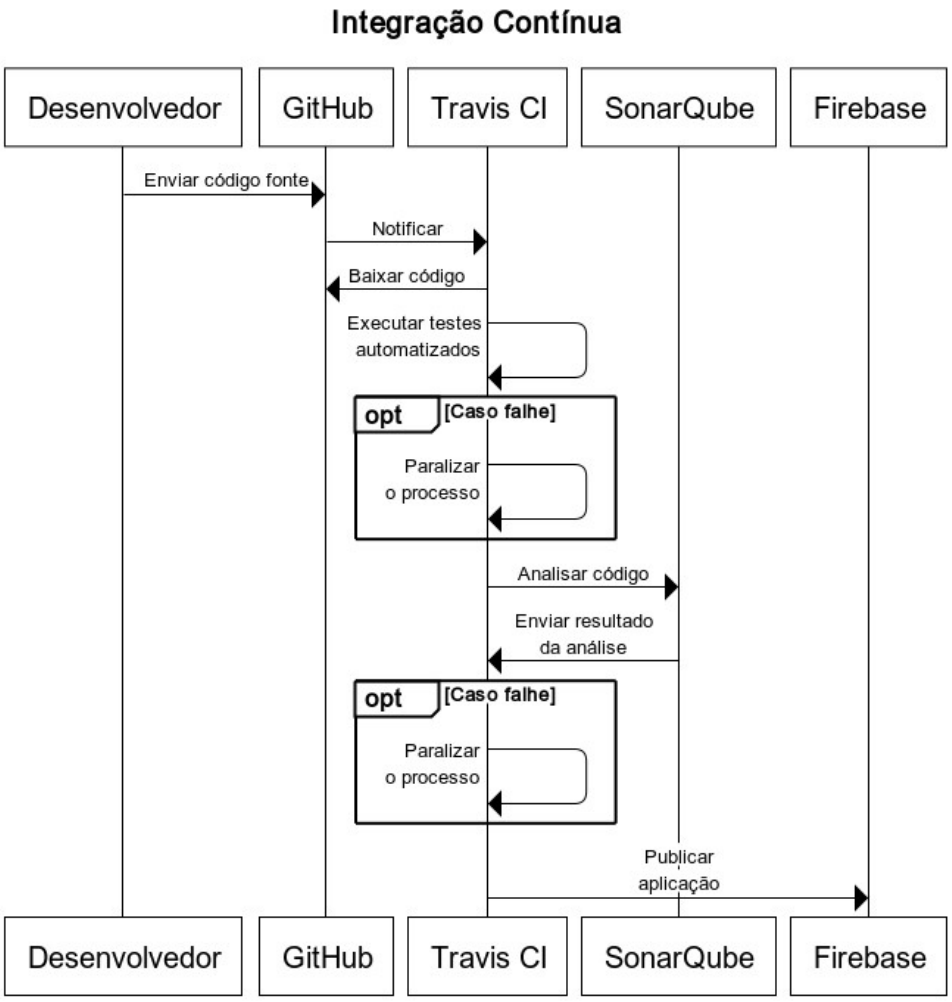

Figura 23 - Processo de integração contínua utilizado - Fonte Própria (2019)

4.4.4 CÓDIGO FONTE

O código fonte da aplicação deste trabalho foi desenvolvido utilizando a linguagem JavaScript que, juntamente com HTML e CSS, são as três principais tecnologias utilizadas por sites na Internet. JavaScript é uma linguagem de programação multi-paradigma mantida pelo grupo ECMA International.

Há diversas bibliotecas disponíveis para a linguagem JavaScript, sendo que para este trabalho foi utilizado a biblioteca React.

React é uma biblioteca utilizada para a construção de interfaces declarativas e baseadas em componentes. Atualmente ela é mantida pelo Facebook e seu código fonte está disponível no GitHub (Facebook Open Source, 2019).

A estrutura básica do código fonte é apresentada na Figura 24. Os principais elementos deste código são:

- app\
  - Nesta pasta estão os arquivos relacionados à biblioteca React.
- componentes\
  - Nesta pasta estão os componentes visuais da aplicação. Dois exemplos de componentes são o visualizador dos grafos e a caixa de texto para edição de JSON.
- examples\
  - Nesta pasta estão os objetos para os grafos de exemplo da aplicação.
- pages\
  - Aqui estão os arquivos relacionados às páginas da aplicação, sendo que neste momento existem apenas duas páginas: a página inicial, com a relação dos grafos de exemplo, e a página principal, para visualização e edição dos grafos.
- redux\
  - Combinada com o React, foi utilizada nesta aplicação a biblioteca Redux. A ideia central desta biblioteca é descrever a aplicação como uma máquina de estados, bem como centralizar o gerenciamento desta máquina de estados. Nesta pasta, estão os arquivos que implementam o Redux para a aplicação.
- services\
  - Aqui estão os códigos que manipulam os grafos e implementam os algoritmos de cálculo da aplicação.
- utils\
  - Nesta pasta estão códigos genéricos utilizados em toda a aplicação.

Dentro de cada uma destas pastas há o código fonte e o teste automatizado correspondente.

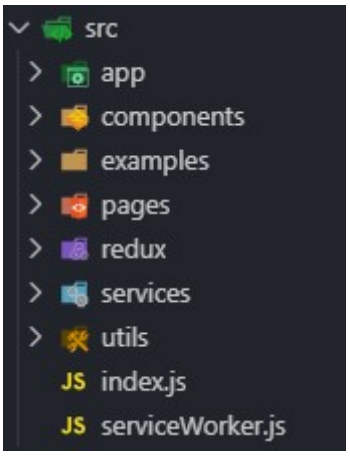

Figura 24 - Estrutura básica do código fonte - Fonte Própria (2019)

4.4.5 VIS.JS

Vis.js foi a biblioteca utilizada para desenhar os grafos na aplicação deste trabalho.

Esta é uma biblioteca de visualização de gráficos de propósito geral. Um de seus componentes é o Network, que pode ser utilizado para representar grafos. É uma biblioteca nativa para o uso em ambiente Web. Sua utilização é feita por meio de código estruturado, em que os vértices e arestas são representados como objetos (Almenda, 2019).

Há duas formas de desenhar os grafos: hierárquica ou não hierárquica. Na forma não-hierárquica, ilustrada na Figura 25, é utilizada uma abordagem de *desenho forçado de grafo*. Nessa abordagem, o objetivo é posicionar os vértices buscando deixar todas as arestas aproximadamente com o mesmo tamanho e reduzindo a quantidade de cruzamentos entre elas (Kobourov, 2012).

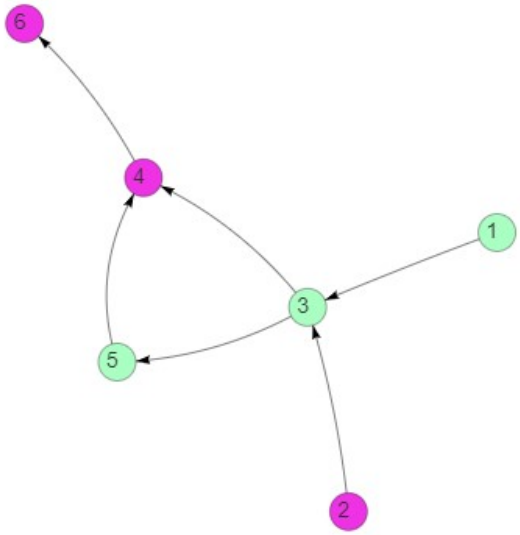

Figura 25 - Visão do grafo em forma não-hierárquica - Fonte Própria (2019)

Já na forma hierárquica, ilustrada na Figura 26, é usado um algoritmo de quatro passos que consiste em: encontrar um ranqueamento ideal para os vértices do grafo, agrupando por fileiras; definir uma ordem para os vértices dentro dessas fileiras, buscando reduzir cruzamentos; encontrar coordenadas ideiais para os vértices e, por fim, usar *splines* para desenhar as arestas (Gansner, 1993).

A seleção do algoritmo de desenho pode ser feita diretamente nas opções existentes na aplicação.

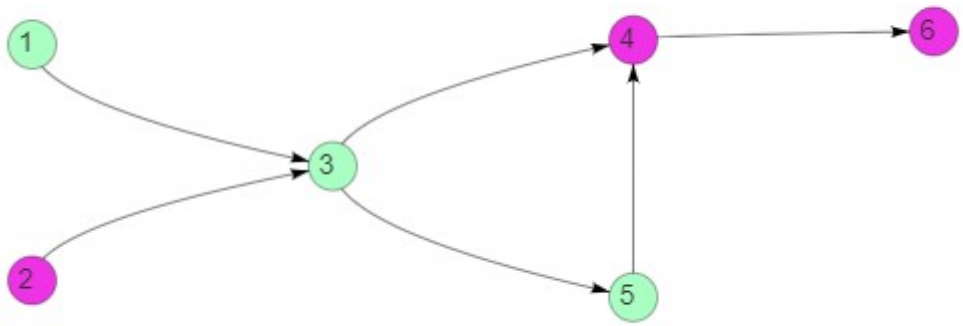

Figura 26 - Visão do grafo em forma hierárquica - Fonte Própria (2019)

4.4.6 QUALIDADE DE CÓDIGO

Para assegurar a qualidade do código da aplicação, foram utilizadas duas principais ferramentas, a ESLint e o SonarQube.

O ESLint é uma ferramenta de análise estática de código específica para JavaScript. Além de padronizar o estilo do código – posicionamento de parênteses e quantidade de espaços em branco, por exemplo – ela é capaz de encontrar pequenas falhas de implementação, como a não utilização de uma variável declarada.

Para este código fonte, foi configurada a execução automática do ESLint a cada *commit* no repositório. Dessa forma, toda vez que o desenvolvedor faz um *commit* no repositório, a ferramenta é executada, e caso sejam encontradas falhas, o *commit* não é aceito e esse código não pode ser enviado para o repositório central.

O SonarQube é outra ferramenta de análise estática capaz de detectar bugs, *codesmells* e vulnerabilidades de segurança em códigos fonte. Em códigos JavaScript, o SonarQube é capaz de encontrar falhas como *loops* infinitos, chamadas inválidas de funções e trechos de código duplicados.

Além da análise de código, o Sonar pode exibir o índice de cobertura de código dos testes automatizados, mais bem detalhados na próxima seção.

No momento em que este texto foi escrito, o código fonte da aplicação não tem nenhum *bug* e nenhum *codesmell*. Há uma vulnerabilidade que está em um código auxiliar utilizado para converter arquivos de texto em objetos de grafos da aplicação. O índice de cobertura do código fonte está em 82.4%.

A Figura 27 mostra o relatório da análise do Sonar, que também está disponível *online* no link https://sonarcloud.io/dashboard?id=matheusaraujo_grafos-evolucao-maturidade

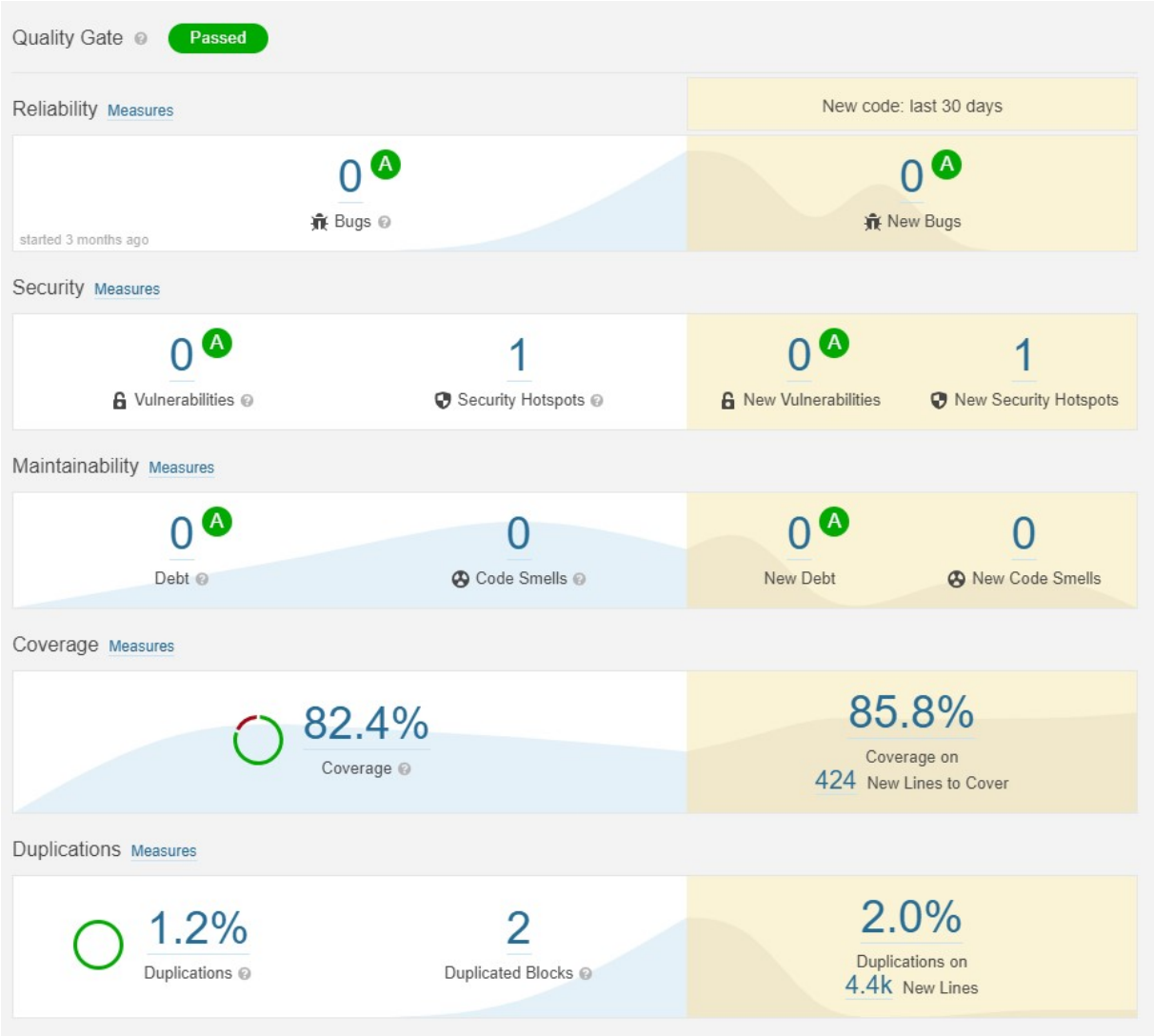

Figura 27 - Relatório Sonar - Fonte Própria (2019)

4.4.7 TESTES AUTOMATIZADOS

Testes automatizados são trechos de código escritos para validar o comportamento do restante do código. Ao automatizar o processo de validação do código, eles dão maior agilidade e segurança ao desenvolvimento de software.

A estrutura básica de um teste é do tipo: Dado, Quando, Então:

- Dado um estado inicial;
- Quando acontecer o evento em análise;
- Então o estado final deve ser ...

Um teste pode ser unitário, quando está validando um elemento unitário do código, ou de integração, quando valida a integração entre diversos elementos.

Neste projeto foram feitos testes automatizados unitários para os componentes de tela, os elementos em *redux* e os em *services.* Para as telas foram usados testes de integração.

Até o momento em que este documento foi escrito, existiam 147 testes e o índice de cobertura da aplicação estava acima de 80%. O índice de cobertura do código indica o percentual do código que é percorrido quando os testes são executados.

A Figura 28 mostra o relatório de cobertura fornecido pelo Sonar.

Figura 28 - Índice de Cobertura - Fonte Própria (2019)

4.5 MODELO GRADE CURRICULAR DE ENGENHARIA DE SISTEMAS

A Figura 29 mostra o grafo para a grade curricular do curso de Engenharia de Sistemas versão 2018-2, a versão vigente durante a execução deste trabalho.

Aqui, os vértices representam as disciplinas e as arestas os pré-requisitos formais. As cores distinguem os diversos grupos de disciplinas.

Foi utilizado o método hierárquico para desenhar o grafo.

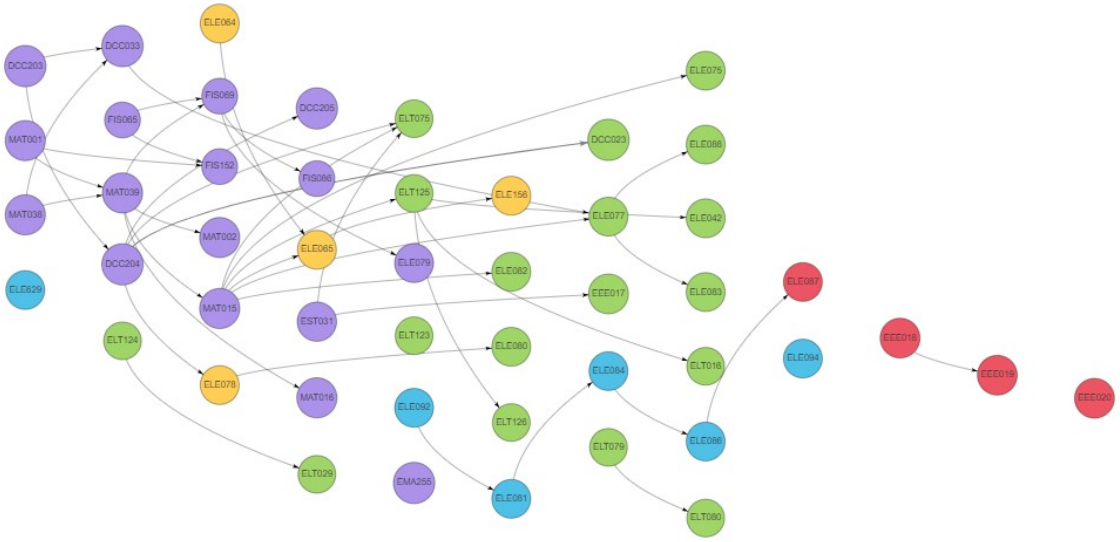

Figura 29 - Grafo Engenharia de Sistemas - Fonte Própria (2019)

4.6 MODELO DEVOPS

A Figura 30 mostra o grafo para o modelo de maturidade proposto por Mendes (2016) com maior granularidade.

Aqui, os vértices representam as habilidades dentro de cada disciplina e as arestas, suas relações de precedência. As cores distinguem as disciplinas.

Foi utilizado o método não-hierárquico para desenhar o grafo.

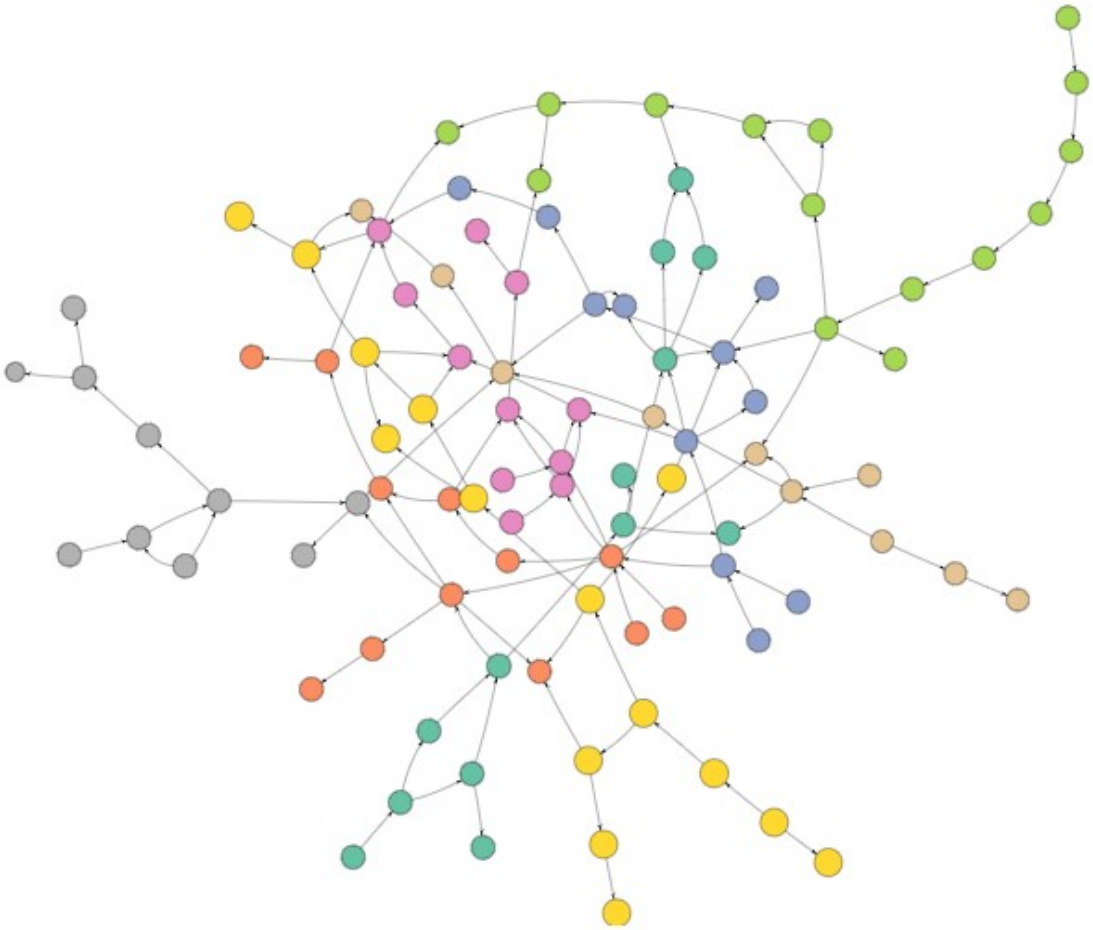

Figura 30 - Grafo Modelo Maturidade DevOps - Fonte Própria (2019)

4.7 ALGORITMOS

Para este trabalho foram consideradas duas perguntas a serem respondidas sobre os grafos de exemplo:

- Qual a melhor combinação de disciplinas para o próximo semestre? Considerando o grafo para a grade curricular, explorado na seção 4.7.1.
- Qual caminho deve ser escolhido para atingir um determinado grau de maturidade? Considerando o grafo para o modelo de maturidade DevOps, explorado na seção 4.7.2.

4.7.1 Próximas disciplinas

Para responder à pergunta de melhor combinação de disciplinas foram consideradas as informações:

- Período da disciplina;
- Carga horária da disciplina;
- Horário em que a disciplina é ofertada;

Neste momento foram consideras apenas disciplinas obrigatórias e não foram consideradas múltiplas ofertas de uma mesma disciplina.

Foram considerados também as seguintes regras:

- **Pré-requisitos formais.** As relações formais de pré-requisitos entre as disciplinas, modeladas como arestas no grafo.
- **Período mais inferior**. O aluno deve matricular-se em todas as disciplinas do período mais inferior não concluídas.
- **Carga horária**. Segundo as normas de graduação da UFMG, o estudante deve matricular-se em pelo menos 13 créditos – ou 195 horas – e no máximo 25 créditos – ou 375 horas.
- **Períodos consecutivos**. O aluno deve matricular-se em disciplinas de até dois períodos consecutivos, com a possibilidade de ser até três períodos quando permitido pelo Colegiado.

Além dessas regras, outras duas adicionais foram consideradas:

- Disciplinas que devem obrigatoriamente ser selecionadas;
- Horários que obrigatoriamente não devem ser considerados.

Consideradas essas informações e regras, foi implementado um algoritmo de força bruta com alguns filtros, descrito pelos passos a seguir:

1. Encontrar todas as disciplinas disponíveis;
    a. Uma disciplina é considerada disponível caso não tenha pré-requisitos ou todos seus pré-requisitos tenham sido cumpridos.
2. Considerando todas as disciplinas disponíveis, gerar todas as combinações possíveis;
3. Caso alguma disciplina deva estar obrigatoriamente na combinação, filtrar as combinações considerando esse critério;
4. Filtrar as combinações, considerando a faixa de carga horária obrigatória;
5. Caso algum horário obrigatoriamente não deva estar na combinação, filtrar as combinações sem esse horário;
6. Filtrar as combinações em que haja conflito de horários;

7. Filtrar as combinações cujos períodos consecutivos extrapolem o limite estabelecido.

Após esses sete passos, as combinações são ordenadas considerando os seguintes critérios:

- Distância entre os períodos
  - A diferença entre o último e o primeiro períodos na combinação.
- Distância total entre os períodos, ou somatório das distâncias;
  - Para cada disciplina é considerada a distância entre ela e o primeiro período considerado na combinação, sendo que a soma de todas elas define esse parâmetro.
- Carga horária da combinação
  - Combinações com menor carga horária têm menor prioridade.
- Total de horários das disciplinas
  - Esse critério é necessário para desempatar combinações com disciplinas cuja carga horária é diferente da quantidade de horários, como Engenharia de Controle, por exemplo;
  - Nesse critério, combinações com mais horários têm menor prioridade.

A resposta do algoritmo é uma lista de combinações filtradas e ordenadas pelos critérios acima apresentados. Após sua execução, o usuário pode escolher uma combinação e incluir suas disciplinas na lista de disciplinas cursadas e calcular novas combinações considerando essas novas disciplinas, de forma iterativa.

No momento da escrita deste documento, este algoritmo já estava disponível na aplicação *online*.

4.7.2 Caminho para maturidade DevOps

Para o modelo de maturidade DevOps, a pergunta a ser respondida é levemente diferente. Nesse modelo, o objetivo não é atingir todas as arestas, mas atingir algumas arestas específicas.

O critério considerado neste cenário é a relação de precedência entre as habilidades.

Para este cenário, o algoritmo utilizado é:

1. Definir as habilidades que se deseja alcançar;
2. A partir das habilidades já alcançadas, definir um subgrafo que contenha todas as habilidades entre as que já foram alcançadas e as que se deseja alcançar;
3. Executar neste subgrafo o algoritmo de ordenação topológica.
    a. O algoritmo de ordenação topológica define uma ordem linear em que todo nó N aparece antes de todos os nós para os quais N tem arestas de saída.

A resposta deste algoritmo é, portanto, uma sequência de atividades que deve ser executada a partir do nível de maturidade atual para atingir um nível de maturidade desejado.

Este algoritmo estava sendo implementado concomitantemente à escrita deste documento.

5 RESULTADOS

Neste capítulo, as possibilidades e os resultados obtidos pelos algoritmos implementados são discutidos.

5.1 GRADE DO CURSO DE ENGENHARIA DE SISTEMAS

O algoritmo implementado para a definição de potenciais combinações de próximas disciplinas foi utilizado para calcular as opções de caminhos a serem seguidos a partir das disciplinas restantes no curso para o autor deste trabalho. A primeira iteração desse cálculo pode ser vista na Figura 31

Considerando as disciplinas cursadas até o segundo semestre de 2019, e as regras citadas anteriormente, seriam necessários mais quatro períodos até concluir o curso:

- Período 1
  - Engenharia de controle
  - Processamento de sinais
  - Computação evolucionária
  - Técnicas de modelagem de sistemas dinâmicos
- Período 2
  - Confiabilidade de sistemas
  - Dispositivos e circuitos eletrônicos básicos
  - Laboratório de projeto V
- Período 3
  - Laboratório de projeto IV
  - Laboratório de circuitos eletrônicos e projetos
- Período 4
  - Projeto multidisciplinar

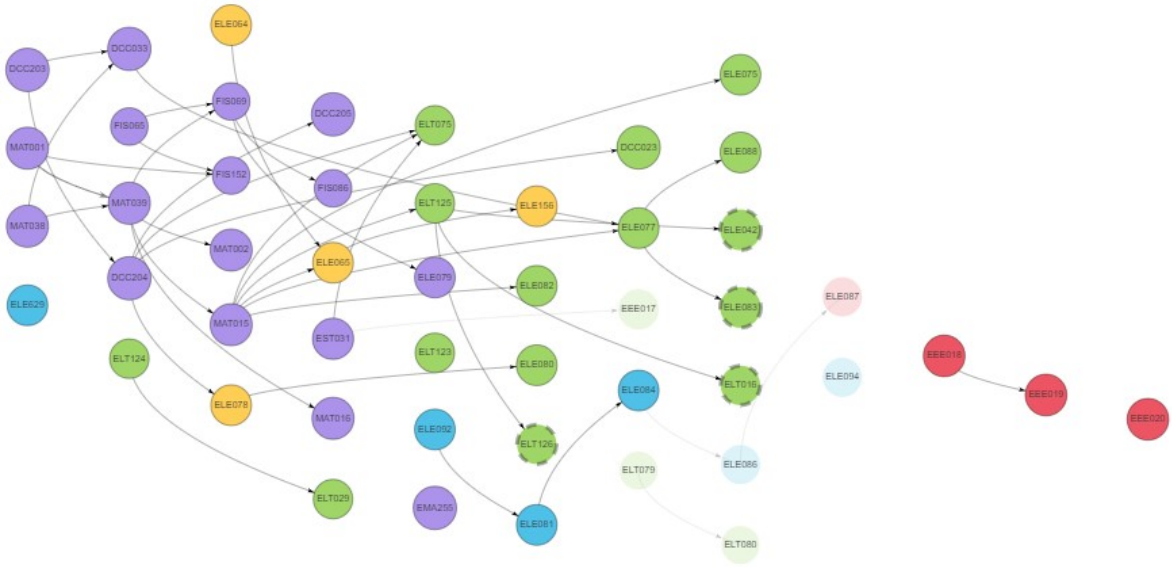

Figura 31 - Cálculo próximas disciplinas, grafo Engenharia de Sistemas – Fonte Própria (2019)

A partir do Período 2, a regra de carga horária mínima passou a não ser considerada, por não existirem disciplinas obrigatórias suficientes para aquele período. No entanto é possível completar a carga horária mínima com disciplinas optativas, que não estão sendo consideradas neste modelo.

Essa funcionalidade da aplicação pode ser utilizada por qualquer estudante na escolha de quais disciplinas cursar. Atualmente, essa decisão é tomada de forma empírica no momento da matrícula. Apesar de ser possível concluir todo a graduação seguindo a ordem pré-definida das disciplinas na grade curricular do curso – que respeita o cruzamento de horários entre as disciplinas –, é possível que um estudante não seja aprovado em uma disciplina, ou até não possa cursá-la. Nesses cenários, a conciliação de horários é trabalhosa e pode ser facilitada por esta ferramenta.

O modelo aqui descrito considera apenas as disciplinas obrigatórias do curso e as relações formais de precedência. É possível melhorá-lo acrescentando disciplinas optativas e relações não-formais de precedência.

Acrescentando essas informações, acredita-se que será possível sugerir combinações de disciplinas mais assertivas e mais bem adequadas ao percurso dos estudantes.

5.2 MODELO DE MATURIDADE DEVOPS

No momento da escrita deste texto, o algoritmo para o modelo de maturidade DevOps estava sendo implementado. Dessa forma, não foi possível demonstrar resultados significativos de sua execução.

No entanto, acredita-se que a representação dos modelos de maturidade DevOps no grafo já traz um avanço em relação à representação usual, dado que a visão gráfica fornecida pelo grafo evidencia relações entre as diversas habilidades requeridas pelo modelo, antes percebidas apenas por abstrações mentais ou pela intuição.

Para o primeiro grafo do modelo – considerando cada nível proposto por Mendes como uma aresta – é possível definir um caminho a ser trilhado para atingir um determinado nível de maturidade apenas analisando mentalmente o grafo, em função da pequena quantidade de elementos existentes no grafo, que totalizam 40 vértices e 55 arestas.

Já para o segundo modelo – com maior granularidade, 97 vértices e 124 arestas – a utilização de um algoritmo como aquele proposto neste trabalho pode ser de grande utilidade.

O algoritmo proposto define uma sequência das habilidades utilizando ordenação topológica, exibindo um sequenciamento factível de habilidades a serem adquiridas. Essa forma de exibição pode ser útil para um tomador de decisão durante a escolha de habilidades nas quais investir, dado um objetivo definido.

No estágio atual de desenvolvimento deste trabalho, o algoritmo considera apenas relações de precedência e define a sequência que deve ser seguida. É possível aprimorá-lo acrescentando o conceito de iterações. Dessa forma, cada iteração teria uma capacidade definida e cada habilidade um custo relacionado, tal como a carga horária das disciplinas e os limites de carga horária por período. Com essa informação, é possível definir um número mínimo de iterações necessárias para atingir o objetivo especificado.

Outra informação que pode ser considerada é a utilização de recursos para cada habilidade. Por exemplo, para atingir uma habilidade H1, é necessário que o recurso R1 seja alocado; pode ocorrer de outra habilidade H2 também necessitar do recurso R1. Dessa forma, as habilidades H1 e H2 não podem estar em uma mesma

iteração. Os recursos teriam o mesmo papel que os horários no modelo de grade curricular. Essa restrição definiria melhor o número mínimo de iterações necessário.

6 DESDOBRAMENTOS FUTUROS

Como mostrado na última seção, a implementação realizada até o presente momento mostra alguns avanços com relação ao que foi proposto desde o Trabalho de Conclusão de Curso 1. No entanto, ainda existem melhorias a serem feitas nas implementações realizadas, tais como:

- Incluir as relações de pré-requisitos não formais no grafo da grade curricular;
- Incluir disciplinas não obrigatórias neste mesmo grafo;
- Incluir mais informações e definir mais critérios para o algoritmo do modelo de maturidade DevOps;
- Definir novos algoritmos a serem implementados. Um deles é o cálculo de arestas críticas, que podem ser entendidas como disciplinas com maior peso no desenvolvimento de um estudante, ou habilidades cruciais nos modelos de maturidade DevOps.
- Melhorias na interface gráfica da aplicação, principalmente na edição dos grafos, passando a ser feita por interfaces mais amigáveis em vez de edição de textos JSON.
- Análise de aplicabilidade também em outras áreas e contextos.

Tendo em vista as perspectivas para a continuidade deste trabalho, foi solicitado e aprovado pelo Colegiado do Curso de Engenharia de Sistemas um projeto de ensino, no qual este trabalho continuará a ser estudado e implementado durante os próximos dois semestres, visando apoiar o Colegiado do Curso de Engenharia de Sistemas na realização da reforma curricular.

7 CONSIDERAÇÕES FINAIS

Modelos de maturidade são ferramentas utilizadas para avaliar as capacidades de um indivíduo ou organização com relação a um processo ou objetivo específico. As representações usuais dos modelos de maturidade são restritivas e comumente deixam de lado especificidades de indivíduos ou organizações que tentam evoluir seguindo aquele modelo.

A teoria de grafos é um ramo da matemática responsável por estudar e definir estruturas denominadas grafos, que representam entidades e suas relações. São diversas as aplicações de grafos, entre elas, a gestão do conhecimento por meio dos grafos RDF. Este trabalho apresentou uma proposta de se utilizar grafos para representar modelos de evolução da maturidade.

Para justificar essa proposição, foi utilizado o Framework Cynefin, uma ferramenta de auxílio à classificação de sistemas. Nesse *framework*, sistemas com relações de causa e efeito evidentes são tratados no domínio do Simples/Óbvio, enquanto sistemas com relações de causa e efeito não evidentes e dispersas são tratados no domínio do Complicado. Em seu formato atual, os modelos de maturidade descritos na literatura podem ser entendidos como ferramentas no domínido do Simples/Óbvio. Ao analisar os modelos de maturidade por meio da teoria de grafos, novas relações de causa e efeito poderão ser evidenciadas, surgindo relações emergentes, antes não percebidas. Dessa forma, a proposta desse trabalho busca elevar a análise da evolução da maturidade para o domínio do Complicado. Além de possibilitar perceber novas relações, a representação por meio dos grafos tentará exprimir de maneira mais assertiva as especifidades de indivíduos ou organizações na implantação de modelos para a evolução de sua maturidade.

Como parte deste trabalho, foi também realizado o desenvolvimento de uma solução computacional que representasse os modelos de maturidade por meio de grafos. Essa solução foi aplicada e avaliada utilizando dois exemplos de grafos de evolução da maturidade em diferentes contextos, entre eles a grade curricular de um curso de gradução e no universo das práticas de DevOps.

REFERÊNCIAS BIBLIOGRÁFICAS

ALMENDA, O. N. Documentação vis.js. **vis.js**, 2019. Disponivel em: <http://visjs.org/>.

AWS. O que significa integração contínua? **AWS DevOps**, Novembro 2019. Disponivel em: <https://aws.amazon.com/pt/devops/continuous-integration/>.

BECK, K.; ANDRES, C. **Extreme Programming Explained:** Embrace Change. [S.l.]: Bookman, 2004.

CORMEN, T. H. **Algoritmos, Teoria e Prática**. Rio de Janeiro: Elsevier, 2012.

COSTA FILHO, B. A.; ROSA, F. D. Maturidade em Gestão Ambiental: Revisitando as melhores práticas. **REAd. Revista Eletrônica de Administração**, p. vol. 23 no. 2, 2017.

CRESCENTE, C. Cynefin para todos!, 7 Agosto 2018. Disponivel em: <https://medium.com/@clau.crescente/cynefin-para-todos-123668785ac4>.

CROSBY, P. B. **Quality Is Free**. New York: McGraw-Hill, 1979.

FACEBOOK OPEN SOURCE. React - A JavaScript library for building user interface. **React**, Novembro 2019. Disponivel em: <https://reactjs.org/>.

FIERRO, D.; PUTINO, S.; TIRONE, L. The Cynefin Framework and the Technical Leadership: How to Handle the Complexity. **INCOSE Italia Conference on Systems Engineering**, Novembro 2017. 72-81.

FIREBASE INC. Planos de preços. **Firebase**, Novembro 2019. Disponivel em: <https://firebase.google.com/pricing?authuser=0>.

FORSGREEN, N.; HUMBLE, J.; KIM, G. **Accelerate:** The Science of Lean Software and DevOps: Building and Scaling High Performing Technology Organizations. [S.l.]: IT Revolution Press, 2018.

FOWLER, M. ShuHaRi. **Martin Fowler**, 22 Agosto 2014. Disponivel em: <https://martinfowler.com/bliki/ShuHaRi.html>.

GANSNER, E. R. E. A. A Technique for Drawing Directed Graphs. **IEEE Transactions on Software Engineering, Volume 19**, p. 214-230, 1993.

GITHUB INC. GitHub is how people build software. **GitHub About**, 6 Novembro 2019. Disponivel em: <https://github.com/about>.

KEOGH, L. Cynefin for Everyone!, 19 Julho 2018. Disponivel em: <https://medium.com/@lunivore/cynefin-for-everyone-d5f47d9bd102>.

KIM, G.; HUMBLE, J.; DEBOIS, P. **Manual de Devops. Como Obter Agilidade, Confiabilidade e Segurança em Organizações Tecnológicas**. [S.l.]: Alta Books, 2018.

KOBOUROV, S. Spring Embedders and Force Directed Graph Drawing Algorithms, 2012.

KOEHLEGGER, M.; MAIER, R.; THALMANN, S. Understanding Maturity Models Results of a Structured. **Proceedings of I-KNOW '09: 9th international conference on knowledge management and knowledge technologies**, 2009.

KURTZ, C. F.; SNOWDEN, D. J. The new dynamics of strategy: Sense-making in a complex and complicated world. **IBM Systemas Journal - Volume: 42, Issue: 3**, p. 462-483, 2003.

MANOLA, F.; MILLER, E. RDF Primer. **The World Wide Web Consortium (W3C)**, 10 Fevereiro 2004. Disponivel em: <https://www.w3.org/TR/2004/REC-rdf-primer-20040210/>.

MENDES, M. Maturidade em Práticas DevOps. **Marco Mendes - Agile, lean, arquitetura, programação e devops**, 13 Dezembro 2016. Disponivel em: <https://marco-mendes.com/2016/12/13/maturidade-em-praticas-devops/>.

MILES, A.; BECHHOFER, S. SKOS Simple Knowledge Organization System Reference. **The World Wide Web Consortium (W3C)**, 18 Agosto 2009. Disponivel em: <https://www.w3.org/TR/skos-reference/>.

MOREIRA, D. A. **Teoria e prática em gestão do conhecimento**. Belo Horizonte: Dissertação (Mestrado em Ciência da Informação, UFMG), 2005.

NONAKA, I.; TAKEUCHI, H. **The Knowledge-creating Company:** How Japanese Companies Create the Dynamics of Innovation. Nova York: Oxford University Press, 1995.

NORMAS ISO. INTRODUCCIÓN SPICE ISO/IEC 33000. **NORMAS ISO**, Novembro 2019. Disponivel em: <https://www.normas-iso.com/spice-iso-iec-33000/>.

RIBAS, T. Como o gestor pode tomar melhores decisões, 1 Outubro 2018. Disponivel em: <https://thomazribas.com/como-o-gestor-pode-tomar-melhores-decisoes/>.

SALVIANO, C. F. **Melhoria e Avaliação de Processo com ISO/IEC 15504 (SPICE) e CMMI**. Lavras: Universidade Federal de Lavras, 2003.

SNOWDEN, D. J.; BOONE, M. E. A leader's framework for decision making. A leader's framework for decision making. **Harvard Business Review**, Novembro 2007. 69-76.

SOFTWARE ENGINEERING INSTITUTE. **CMMI for Development (CMMI-DEV), Version 1.3, Technical Report CMU/SEI-2010-TR-034**. Pittsburgh, PA. 2010.

SOFTWARE FREEDOM CONSERVANCY. git --distributed-even-if-your-workflow-isnt. **git**, 5 Novembro 2019. Disponivel em: <https://git-scm.com/>.

SOMMERVILLE, I. **Engenharia de Software 10ed**. São Paulo: Pearson Education do Brasil, 2018.

STACK OVERFLOW. Stack Overflow Developer Survey Results. **Developer Survey Results 2018**, Janeiro 2018. Disponivel em: <https://insights.stackoverflow.com/survey/2018#work-_-version-control>.

UNIVERSITY OF WALES. Welsh-English / English-Welsh On-line Dictionary. **Welsh-English / English-Welsh On-line Dictionary**, 12 Junho 2019. Disponivel em: <http://www.geiriadur.net/index.php?page=ateb&term=cynefin&direction=we&type=all&whichpart=beginning>.

WEINBERG, G. M. **Software com qualidade**. Rio de Janeiro: Makron Books, 1992.

ZAMBALDE, A. L.; ALVEZ, R. M. **Gestão do Conhecimento e Inovação**. Lavras: Universidade Federal de Lavras, 2004.

ZIVIANI, N. **Projeto de Algoritmos**. São Paulo: Cengage Learning, 2011.