RELCASI

Volume 11 | Issue 2 Article 5

7-1-2019

Vulnerability detection techniques and tools: a survey with members of agile teams in Brazil

Lígia C. M. de C. Santos
Escola de Artes, Ciências e Humanidades, Universidade de São Paulo, ligia.santos01@usp.br

Edmir Parada Vasques Prado
Escola de Artes, Ciências e Humanidades, Universidade de São Paulo, eprado@usp.br

Marcos Lordello Chaim
Escola de Artes, Ciências e Humanidades, Universidade de São Paulo, chaim@usp.br

Recommended Citation
Santos, Lígia C. M. de C.; Prado, Edmir Parada Vasques; and Chaim, Marcos Lordello (2019) "Vulnerability detection techniques and tools: a survey with members of agile teams in Brazil," *RELCASI*: Vol. 11 : Iss. 2 , Article 5.
DOI: 10.17705/1relc.00070
Available at: https://aisel.aisnet.org/relcasi/vol11/iss2/5

Latin American and Caribbean Journal of the Association for Information Systems
Revista Latinoamericana Y Del Caribe De La Asociación De Sistemas De Información
Revista Latino-americana e Caribenha da Associação de Sistemas de Informação

Research Paper DOI: 10.17705/3relc.00070 ISSN: 1937-8831

Vulnerability detection techniques and tools: a survey with members of agile teams in Brazil

Técnicas y herramientas de detección de vulnerabilidades: una encuesta con miembros de equipos ágiles en Brasil

Técnicas e ferramentas de detecção de vulnerabilidades: um *survey* com membros de equipes ágeis no Brasil

Lígia C. M. de C. Santos
Escola de Artes, Ciências e Humanidades, Universidade de São Paulo
São Paulo, Brasil
ligia.santos01@usp.br

Edmir Parada Vasques Prado
Escola de Artes, Ciências e Humanidades, Universidade de São Paulo
São Paulo, Brasil
eprado@usp.br

Marcos Lordello Chaim
Escola de Artes, Ciências e Humanidades, Universidade de São Paulo
São Paulo, Brasil
chaim@usp.br

Abstract

There is little information on the adoption of agile methodologies in Latin America, and there is a percentage of professionals who receive little formal training. This article aims to identify and describe the interest of team members who use agile methods in software development in vulnerability detection techniques and tools (VDTT). To do so, a survey was proposed listing 18 techniques and tools that make up three well-known processes of secure software development, namely: McGraw Process, OWASP CLASP and Howard and Lipner's activities. The survey was applied to 111 members of agile software development teams who have already deployed, are in the process of deploying, or are going to deploy vulnerability detection techniques and tools in the projects. The participants of the survey were: (1) requirements analysts; (2) software architects; (3) developers; and (4) testers. Some organizations have specific members of agile teams, such as security experts, scrum masters, product owners, and agile coaches. From the collected data it was possible, by means of statistical techniques like Fisher's Exact Test and measures of Lambda and Gama associations, to identify relationships between interest in the use of VDTT and the adoption of agile methods and the adoption of software quality frameworks and services**.**

Keywords: Software Development, Vulnerability Detection, Agile Methods, Survey.

Resumen

Existe poca información sobre la adopción de metodologías ágiles en América Latina, y hay un porcentaje de profesionales que reciben poca formación formal. Este artículo tiene como objetivo identificar y describir el interés de los miembros del equipo que utilizan métodos ágiles en el desarrollo de software en técnicas y herramientas de detección de vulnerabilidades (TFDV). Para ello, se propuso una encuesta que enumera 18 técnicas y herramientas que componen tres conocidos procesos de desarrollo de software seguro, a saber: McGraw Process, OWASP CLASP y las actividades de Howard y Lipner. La encuesta se aplicó a 111 miembros de equipos de desarrollo de software ágil que ya se han implementado, están en proceso de implementación o van a implementar técnicas y herramientas de detección de vulnerabilidades en sus proyectos. Los participantes de la encuesta fueron: (1) analistas de requisitos; (2) arquitectos de software; (3) desarrolladores; y (4) probadores. Algunas organizaciones de la muestra tienen miembros específicos de equipos ágiles, como expertos en seguridad, *Scrum Masters*, propietarios de productos y *coaches ágiles*. A partir de los datos recopilados, fue posible utilizar técnicas estadísticas *Fisher Exact Test* y medidas de asociaciones Lambda y Gama para identificar relaciones entre el interés en el uso de TFDV y la adopción de métodos ágiles y la adopción de marcos de software y calidad de servicio.

Palabras clave: Desarrollo de Software, Detección de Vulnerabilidades, Métodos Ágiles, Encuesta.

Resumo

Há poucas informações sobre a adoção de metodologias ágeis na América Latina, e há um percentual de profissionais que recebem pouco treinamento formal. Este artigo visa identificar e descrever o interesse dos membros de equipes que usam métodos ágeis no desenvolvimento de software em técnicas e ferramentas de detecção de vulnerabilidades (TFDV). Para tanto, foi proposto um *survey* que lista 18 técnicas e ferramentas que compõem três conhecidos processos de desenvolvimento de software seguro, a saber: Processo de McGraw, OWASP CLASP e as atividades de Howard e Lipner. O *survey* foi aplicado a 111 membros de equipes de desenvolvimento ágil de software que já implantaram, estão em processo de implantação ou ainda vão implantar técnicas e ferramentas de detecção de vulnerabilidades nos projetos. Os participantes do *survey* foram: (1) analistas de requisitos; (2) arquitetos de software; (3) desenvolvedores; e (4) testadores. Algumas organizações da amostra possuem membros específicos de equipes ágeis, tais como especialistas em segurança, Scrum masters, *product owners* e *agile coaches*. A partir dos dados coletados foi possível por meio das técnicas estatísticas *Fisher Exact Test* e medidas de associações Lambda e Gama identificar relações entre interesse no uso de TFDV e a adoção de métodos ágeis e a adoção de *frameworks* de qualidade de software e serviços**.**

Palavras-chave: Desenvolvimento de Software, Detecção de Vulnerabilidades, Métodos Ágeis, Survey.

1 Introdução

Atualmente, as organizações operam em ambientes com mudanças constantes e forte competitividade. Por essa razão, elas precisam responder de forma ágil às pressões competitivas. Como consequência, as metodologias ágeis têm sido cada vez mais utilizadas em projetos de desenvolvimento de software em todo o mundo. No entanto, para Quelal, Villavicencio e Mendoza (2018) há poucas informações sobre a adoção dessas metodologias na América Latina, e no Equador há um percentual considerável de profissionais que não recebem treinamento formal antes de adotar metodologias ágeis.

Para ambientes com constantes mudanças uma abordagem ágil é recomendável. Esta abordagem, de acordo com Beck *et al*. (2001), possui valores e princípios que devem ser seguidos. Entretanto, devido à agilidade do processo de desenvolvimento e à pressão na entrega, alguns softwares não têm uma análise

completa da segurança para cada fase do ciclo de vida e, com isso, podem surgir vulnerabilidades (OWASP, 2015). A maioria das vulnerabilidades nos sistemas de software são causadas pela falta de aplicação de políticas de segurança no projeto (Beck *et al*., 2001).

Para evitar a ocorrência de vulnerabilidades, segundo Howard e Lipner (2006), é necessário analisar: o uso de práticas de segurança pelas organizações; as habilidades dos membros; e as necessidades de formação da equipe. Além disso, também é necessário adaptar os processos de segurança no desenvolvimento de software tradicional aos processos dos métodos ágeis. Howard e Lipner (2006) apontam que práticas de segurança incorporadas ao ciclo de vida de desenvolvimento de software podem diminuir a probabilidade de existirem vulnerabilidades no software.

Pelo fato de os métodos ágeis trabalharem com incrementos rápidos, muitas vezes a segurança não é verificada adequadamente. Além disso, esses métodos têm se tornado prevalentes na indústria de software por utilizar práticas baseadas em pouca documentação, interação com o cliente, respostas a mudanças, muita comunicação verbal e foco na geração de código executável da aplicação (Beck *et al*., 2001).

É importante avaliar as práticas de segurança aplicadas pelos membros de equipes que usam métodos ágeis e se estas vêm obtendo êxito. Isso porque os ataques cada vez mais frequentes em softwares exigem das organizações políticas de segurança efetivas (Vianna & Fernandes, 2015). Além disso, as organizações têm dificuldade em controlar as vulnerabilidades, pois são muitas as existentes (OWASP, 2015). As práticas de segurança, sejam elas processos, atividades, técnicas ou ferramentas, podem melhorar a segurança das aplicações.

Apesar da importância do tema, poucos estudos avaliam habilidades, necessidades de formação e práticas para segurança no desenvolvimento ágil de software, pois a maioria dos estudos aborda esse aspecto para desenvolvimento tradicional de software. A literatura destaca diversos processos de desenvolvimento seguro de software tais como Processo de McGraw (McGraw's process), Ciclo de Vida de Desenvolvimento de Segurança da Microsoft (Microsoft Security Development Life Cycle), Modelo de Desenvolvimento Seguro de Software (Secure Software Development Model), Processo Leve e Abrangente de Aplicação de Segurança (Comprehensive, Lightweight Application Security Process) e Processo para Equipes de Desenvolvimento de Software Seguro (Team Software Process for Secure Software Development), entre outros. Entretanto, nenhum deles está voltado exclusivamente para métodos ágeis.

Dentro deste contexto, o objetivo desta pesquisa é identificar e descrever o interesse dos membros de equipes ágeis no desenvolvimento de software na utilização de técnicas e ferramentas de detecção de vulnerabilidades (TFDV). Para atingir este objetivo os seguintes objetivos específicos foram definidos: (1) descrever as TFDV existentes na literatura; (2) identificar quais TFDV são de interesse dos membros de equipes que usam métodos ágeis; (3) identificar a associação entre o uso de métodos ágeis e o interesse por TFDV; e (4) identificar a associação entre o uso de *frameworks* relacionados a qualidade de software e o interesse por TFDV.

No que se refere à estrutura do artigo, primeiramente, a Seção 2 apresenta a fundamentação teórica. Em seguida é apresentado o modelo de pesquisa na Seção 3 e o método de pesquisa empregado na Seção 4, formando a base para a apresentação e análise – subsequentes – dos resultados (Seção 5). A Seção 6 encerra o artigo com a discussão dos resultados e o apontamento das considerações finais.

2 Fundamentação teórica

A fundamentação foi elaborada a partir de uma revisão bibliográfica da literatura e é composta por três tópicos. O primeiro trata dos métodos ágeis no qual são apresentados os principais métodos e suas aplicações. O segundo tópico trata das vulnerabilidades e são apresentadas as principais vulnerabilidades, bem como as técnicas de prevenção e as TFDV. Por fim, o terceiro tópico apresenta os *frameworks* de qualidade de software e serviços mais usados por organizações brasileiras.

2.1 Métodos ágeis

Beck (2000) definiu no Manifesto Ágil os valores e os princípios dos métodos ágeis. Dentre os valores, os seguintes foram definidos: indivíduos e interações acima de processos e ferramentas; software operacional

acima de documentação completa; colaboração dos clientes acima de negociação contratual; e respostas a mudanças antes de seguir um plano.

O Manifesto Ágil não rejeita processos e ferramentas, documentação, negociação de contratos ou o planejamento, mas mostra que eles têm importância secundária quando comparados aos indivíduos e interações, ao software funcionando, à colaboração com o cliente e às respostas rápidas a mudanças e alterações (Beck, 2000). Beck (2000) também definiu, no Manifesto Ágil, 12 princípios que fundamentam os métodos ágeis.

2.1.1 Principais métodos ágeis

Cada um dos métodos ágeis possui diferentes ideias, processos e atividades. Porém, todos compartilham de forma geral a essência, os princípios e valores do manifesto ágil e, por isso, têm muito em comum (Sommerville, 2011). Os métodos ágeis mais conhecidos são: *Extreme Programming* (XP), *Scrum*, *Lean*, *Kanban*, *Crystal Method*, *Dynamic Systems Development Method* (DSDM), *Feature-Driven Development* (FDD) e *Agile Unified Process* (AUP). A seguir são descritos dois métodos ágeis muito utilizados: *Extreme Programming* (XP) e *Scrum*.

***Extreme Programming* (XP)**. É um método focado na excelência de suas aplicações que adota boas práticas de programação, clareza na comunicação entre seus membros e trabalho em equipe. Beck (2000) lista as seguintes características do XP:

- Filosofia de desenvolvimento de software baseada nos valores de comunicação, *feedback*, simplicidade, coragem e respeito.
- Um conjunto de práticas para desenvolvimento de software que se complementam, ampliando assim os seus efeitos.
- Um conjunto de princípios e técnicas que traduzem valores em prática.
- Uma comunidade que compartilha esses valores e as mesmas práticas.

Scrum. É um *framework* dentro do qual pessoas podem tratar e resolver problemas complexos e adaptativos, enquanto produzem e entregam produtos com o mais alto valor possível. Entre os valores fundamentais desse *framework* destacam-se o emprego de uma abordagem iterativa e incremental para aperfeiçoar a previsibilidade e o controle de riscos dos eventos e artefatos produzidos durante o projeto. Schwaber (2013) descreve os três pilares do Scrum que apoiam a implementação de um controle de processo: transparência, inspeção e adaptação.

Cada um dos componentes do *Scrum* tem um propósito específico que é essencial para obter sucesso (Schwaber, 2013). De acordo com este autor, o Scrum possui os seguintes papéis-chave:

- *Scrum master*: responsável por garantir que o Scrum seja entendido e aplicado pela equipe.
- Proprietário do produto: responsável por maximizar o valor do produto e do trabalho da equipe de desenvolvimento.
- Equipe de desenvolvimento: profissionais que realizam o desenvolvimento e entrega de uma versão funcional do sistema.

2.1.2 Aplicação dos métodos ágeis

Métodos ágeis foram desenvolvidos para serem usados por pequenas equipes que atuam juntas no desenvolvimento de pequenos a médios sistemas (Sommerville, 2011). A aplicação de métodos ágeis oferece benefícios importantes entre os quais destacam-se:

- Adaptação de processos tradicionais. Leffingwell (2007) lista possíveis adaptações das atividades dos processos tradicionais de desenvolvimento de software em métodos ágeis com incorporação de práticas de segurança:
  - No desenvolvimento de sistemas de grande porte não é possível focar apenas no código é necessário realizar atividades de arquitetura de software e documentações que descrevam os aspectos mais críticos do sistema.

- Mecanismos de comunicação eficientes e eficazes entre as equipes devem ser desenvolvidos e usados. Deve-se introduzir uma variedade de canais de comunicação, tais como e-mail, ferramentas de videoconferência e mensagens instantâneas para facilitar a comunicação.
- É essencial manter o sistema atualizado por meio de entregas regulares de versões atualizadas do sistema. Além disso, novas ferramentas de gerenciamento de desenvolvimento de software devem ser introduzidas.

(2) Projetos específicos. Grandes empresas têm adotado alguns princípios e fundamentos dos métodos ágeis em projetos específicos (Lindvall *et al*., 2004).

Autores como Lindvall *et al*. (2004) alegam a necessidade de adaptações em situações específicas. Para usar métodos ágeis é essencial que a equipe adapte os processos e atividades existentes e mantenha apenas artefatos realmente essenciais, utilizando processos enxutos e enfatizando a estratégia de entrega incremental. De forma semelhante, Pressman (2009) alega que a aplicação de métodos ágeis não é indicada para todos os tipos de projetos, softwares e situações. Segundo este autor, gestão de mudanças e procedimentos de testes são exemplos de atividades que podem não ser compatíveis com métodos ágeis. Isto porque gestão de mudança é um processo que controla alterações em um sistema de modo que o impacto da alteração e custos sejam controlados.

Por último, cabe destacar a importância do processo de mudança de métodos tradicionais para métodos ágeis. Para Sommerville (2011), organizações de grande porte que desejam usar métodos ágeis no desenvolvimento de software precisam dedicar tempo e recursos significativos para o processo de mudança. Howard e Lipner (2006) corroboram essa afirmação e alegam que a adoção de métodos ágeis em organizações de grande porte é um processo de mudança cultural que leva muito tempo para ser consolidada, requerendo uma mudança na forma de gestão antes de ser aplicada.

2.2 Vulnerabilidades

Uma vulnerabilidade em software é um conjunto de condições que pode levar à violação de uma política de segurança (Seacord & Householder, 2005). Tais condições podem ser oriundas da má especificação de requisitos de segurança, problemas de design, práticas de codificação insegura, falhas nas atividades de garantia de segurança ou problemas de manutenção do sistema (Howard & Lipner, 2006).

A vulnerabilidade é uma fraqueza na aplicação que permite que um atacante cause danos aos stakeholders de um sistema (OWASP, 2010). Em um sistema, procedimentos de segurança, controles internos ou de implementação que podem ser explorados por ameaças são considerados vulnerabilidades (NIST, 2009). É, portanto, importante detectar vulnerabilidades durante o desenvolvimento de software, pois informações perdidas, utilizadas incorretamente e acessadas por pessoas não autorizadas podem prejudicar uma organização.

Quando é mencionado o termo vulnerabilidade em software, vários outros correlatos vêm juntamente a este, como é o caso de ameaças (Grieskamp *et al*., 2002). Uma ameaça é uma ocorrência que pode acarretar dano para o sistema ou organização em um ativo ou grupos de ativos (Stallings, 2012). É uma possível violação de um sistema computacional que pode ser acidental ou intencional. Uma ameaça acidental pode ser, por exemplo, uma possível falha no hardware ou software. Já a ameaça intencional está associada à premeditação, como um monitoramento não autorizado por *hackers* (Pinheiro, 2011). Assim, uma ameaça está relacionada ao termo vulnerabilidade, pois esta é definida como uma condição que, quando explorada por um atacante, pode resultar em uma violação de segurança (CERT.BR, 2012).

Um ataque ocorre quando uma ameaça intencional é realizada por motivos variados como: ganhos financeiros, venda de informações, espionagem ou sabotagem (Pinheiro, 2011). Uma das possíveis causas do sucesso de um ataque é a dificuldade em desenvolver softwares com a devida segurança (Vianna & Fernandes, 2015).

Outro termo associado a vulnerabilidades é o risco, o qual pode ser definido como a probabilidade da ocorrência de uma ameaça (Verdon & McGraw, 2004). Uma vulnerabilidade também está ligada ao termo falha que pode ser definido como a incapacidade de executar uma função requerida dentro dos limites especificados (GROUP, 2010). De acordo com Weber (2003), componentes físicos envelhecem e sofrem

com interferências externas, sejam ambientais, sejam humanas. Assim, as falhas geralmente são classificadas em falhas físicas, aquelas de que padecem os componentes, e falhas humanas, as que compreendem às falhas de projeto e de interação.

As principais causas de falhas são problemas de especificação, problemas de implementação, componentes defeituosos, além de variações ambientais, como temperatura, pressão, umidade e problemas operacionais (Weber, 2003). É possível associar uma vulnerabilidade ao termo malicioso, que pode ser um código adicionado, alterado ou removido de um sistema de software, a fim de causar intencionalmente danos ou subverter a função pretendida do sistema (McGraw & Morrisett, 2000).

Uma vulnerabilidade também está ligada ao termo engano que é definido como uma ação humana que produz um defeito (GROUP, 2010).

Um defeito pode ser introduzido em qualquer uma das fases do ciclo de desenvolvimento de software, como na especificação de requisitos, design, implementação, verificação ou manutenção (Jones, 2013).

2.2.1 Principais vulnerabilidades de software

Segundo OWASP (2010), aplicações seguras são resultado da decisão de produzi-las por parte da organização. Essa organização listou as principais vulnerabilidades em aplicações web:

- Injeções: injeções de falhas, particularmente injeções SQL (*Structured Query Language*), são comuns em aplicações web. Elas acontecem quando os dados que o usuário dá como entrada são enviados como parte de um comando ou consulta.
- *Cross site scripting - XSS*: ocorre em aplicações que passam informações do usuário ao navegador sem validar ou codificar seu conteúdo.
- Referência insegura direta a objetos: ocorre quando um desenvolvedor expõe uma referência a um objeto de implementação interna como um registro na base de dados ou um parâmetro de um formulário.
- *Cross site request forgery*: força o navegador da vítima a enviar requisições para uma aplicação web vulnerável, realizando uma ação maliciosa em nome da vítima.
- Ausência de configurações: é necessário definir e implantar uma configuração segura em aplicações, *frameworks*, servidor de aplicativos, servidor web, servidor de banco de dados, entre outros.
- Exposição de dados sensíveis: muitas aplicações web não protegem adequadamente seus dados sensíveis, tais como informações de cartões de crédito, documentos e credenciais de autenticação.
- Falta de controle de nível de acesso: as aplicações web, que verificam credenciais dos seus usuários, precisam executar verificações de controle de acesso no servidor.
- Falha de autenticação e gerenciamento de sessão: funções de aplicação como autenticação e gerenciamento de sessão, muitas vezes, não são implementadas corretamente, permitindo que possíveis ataques comprometam senhas, chaves ou *tokens* de sessão.
- Uso de componentes vulneráveis conhecidos: bibliotecas, estruturas e outros módulos de software são executados com privilégios totais. Aplicações que utilizam componentes com vulnerabilidades conhecidas podem permitir ataques.
- Redirecionamento e encaminhamento inválido: aplicações web redirecionam e encaminham seus usuários para páginas usando dados não confiáveis para determinar as páginas de destino.

2.2.2 Técnicas para prevenção e detecção de vulnerabilidades

A literatura apresenta diversas técnicas para prevenção e detecção de vulnerabilidade. A seguir destacam-se as mais citadas.

- Prevenção de vulnerabilidades. É feita por meio da especificação de requisitos de segurança. Existem diversas linguagens para especificações de requisitos de segurança e cada uma destas possui diferentes propriedades. A seguir serão descritas algumas das linguagens de especificação utilizadas e identificadas na literatura.

- *Misuse case*: é um tipo especial de diagrama de casos de uso da UML que descreve o comportamento indesejável do software (Khan & Zulkernine, 2009). É importante empregar *misuse cases* para modelar e analisar cenários de segurança, pois auxilia na melhoria da segurança e redução de ameaças (Alexander, 2003).
- *Abuse case*: é uma linguagem de especificação de requisitos de segurança baseada na interação entre atores e o sistema, na qual os resultados da interação são prejudiciais ao sistema. São definidas as interações que resultam em dano a um recurso, aos atores ou ao próprio sistema (Mcdermott & Fox, 1999).

- *Design* seguro. De acordo com Khan e Zulkernine (2009), a fase de *design* identifica a estrutura e os requisitos gerais do software. As linguagens de especificação de requisitos de segurança e modelagem de ameaças compõem a fase de análise de requisitos e modelagem, mas podem ser implantadas na fase de *design*, abordando aspectos estruturais da segurança da aplicação e com um nível de detalhamento maior. Além disso, outras técnicas, tais como *design* seguro, contramedidas de segurança e avaliação de vulnerabilidades podem ser implantadas nesta fase.
- Técnicas para detecção de vulnerabilidades. De acordo com Wiesmann *et al*. (2005) uma das principais técnicas de detecção de vulnerabilidades é a revisão de código. Porém, tal técnica está sujeita a equívocos dos desenvolvedores, além de exigir bom conhecimento na linguagem de programação utilizada. A seguir destacam-se as técnicas para detecção de vulnerabilidades mais citadas na literatura.
  - Técnicas e ferramentas estáticas. Técnicas estáticas são técnicas para detecção de vulnerabilidades aplicadas diretamente no código-fonte sem executar a aplicação. De acordo com Petukhov e Kozlov (2008) exemplos de técnicas estáticas para detecção de vulnerabilidades são: parsing, que realiza uma análise do código fonte do programa, criando uma representação do programa por meio de uma árvore sintática (Freitez, Mammar, & Cavalli, 2009); análise de fluxo de dados, que se refere a um conjunto de técnicas que derivam informações sobre o fluxo de dados ao longo da seleção de caminhos de execução de um programa (Aho, Sethi, & Ullman, 2007); análise de fluxo de controle, que é modelado como um fluxo de controle para representar blocos de código na forma de nós com dependências de controle sob a forma de arestas direcionadas (Zhioua, Short, & Roudier, 2014); e qualificador de tipos, que fornecem um mecanismo para especificar e verificar propriedades de programas (Foster, Fähndrich, & Aiken, 1999).
  - Técnicas dinâmicas. São técnicas de testes de software que, para detectar dinamicamente vulnerabilidades, necessitam da execução do código-fonte da aplicação. Elas têm por objetivo verificar se os mecanismos de proteção construídos para o sistema irão proteger o sistema de acessos indevidos. O testador simula o papel do indivíduo que deseja acessar o sistema (Santos, 2011). De acordo com Petukhov e Kozlov (2008), exemplos de técnicas dinâmicas são: teste de penetração, que é o processo de tentar obter acesso a um sistema sem conhecimento de nomes de usuários, senhas e outros meios de acesso (Northcutt *et al*., 2006); e *fuzz testing*, que é uma técnica que fornece entradas inválidas, inesperadas ou aleatórias a um sistema e observa os possíveis defeitos, como lançamento de exceções imprevistas, colapso do cliente ou servidor, entre outros (Freitez, Mammar, & Cavalli, 2009).

2.3 *Frameworks* de qualidade de software e serviços

Entre os modelos de referência existentes na literatura, Cristofoli, Prado e Takaoka (2012) destacaram o *Information Technology Infrastructure Library* (ITIL) como um dos modelos mais utilizados no gerenciamento da tecnologia da informação (TI). Especificamente na área de qualidade de software, os *frameworks* mais citados são o *Capability Maturity Model Integration* (CMMI) e o MPS.br apresentados a seguir.

2.3.1 Information Technology Infrastructure Library

O ITIL é uma biblioteca de infraestrutura de TI que reúne as melhores práticas para a gestão dos serviços de TI (Axelos, 2018). Ele mapeia o ciclo de vida de serviços de TI por meio de cinco pilares.

- Estratégia do serviço. Trata das decisões estratégicas relacionadas aos serviços que serão desenvolvidos para o alcance dos objetivos de negócio.
- Desenho de serviço. Organiza as decisões estratégicas, elaborando as descrições das especificações dos serviços.
- Transição de serviço. Gerencia as implantações de serviços e transfere os serviços definidos para o ambiente de produção.
- Operação do serviço. Envolve os processos usados diariamente e que mantêm os serviços funcionando.
- Melhoria contínua do serviço. Promove a melhoria dos serviços, aplicando o ciclo PDCA (*Plan-Do-Check-Act*).

Esses cinco pilares ainda são compostos por 26 processos e quatro funções que contribuem para a aplicação dos serviços de acordo com as áreas, fases do ciclo de vida e funções. Os principais benefícios proporcionados pelo uso do ITIL são: eficiência operacional, redução dos custos e esforços desprendidos pela área de TI; cumprimento das atividades; e alinhamento do setor de TI com a área de negócios.

2.3.2 Capability Maturity Model Integration

O CMMI (*Capability Maturity Model Integration*) é uma abordagem para a melhoria de processos. Ele fornece às organizações elementos essenciais para promover processos eficazes e pode ser usado para guiar a melhoria em um projeto, em uma divisão ou em uma organização inteira (CMMI, 2017).

O modelo é composto por quatro áreas de conhecimento, a saber: Engenharia de Sistemas (CMMI-SE), Engenharia de Software (CMMI-SW), Seleção de Fornecedores (CMMI-SS) e Desenvolvimento Integrado de Produto e Processo (CMMI-IPPD). Assim, o CMMI contribui para o desenvolvimento de produtos, prestação de serviços e aquisição por meio de cinco níveis:

- Nível 1 - Inicial. Processos são imprevisíveis, pouco controlados e reativos.
- Nível 2 - Gerenciado. Processos são caracterizados por projeto e as ações são reativas.
- Nível 3 - Definido. Processos são caracterizados para a organização e são proativos.
- Nível 4 - Quantitativamente gerenciado. Os processos são medidos e controlados.
- Nível 5 - Otimizado. Foca na melhoria dos processos.

2.3.3 Melhoria do processo de software brasileiro (MPS.br)

É um modelo de qualidade de processos criado pela Softex (Associação para Promoção da Excelência do Software Brasileiro). A implantação do modelo MPS-BR tem como principal benefício o melhoramento na qualidade dos produtos aumentando assim sua competitividade (Softex, 2016). O objetivo do programa MPS.br é promover a melhoria de desenvolvimento de software nas empresas brasileiras. De acordo com a Softex (2016) ele possui duas metas a alcançar a médio e longo prazos: meta técnica, que visa à criação e aprimoramento do Modelo MPS.br com resultados esperados; e meta de negócio, que visa à disseminação e à adoção do modelo em todas as regiões do país, em um intervalo de tempo determinado, a um custo razoável, tanto em micro, pequenas e médias empresas.

A Softex (2016) aborda cinco componentes que compõem o modelo MPS.br, são eles: Modelo de Referência MPS para Software, Modelo de Referência MPS para Serviços, Modelo de Referência MPS para Gestão de Pessoas, Método de Avaliação e Modelo de Negócio para Melhoria de Processo de Software e Serviços.

3 Modelo de referência da pesquisa

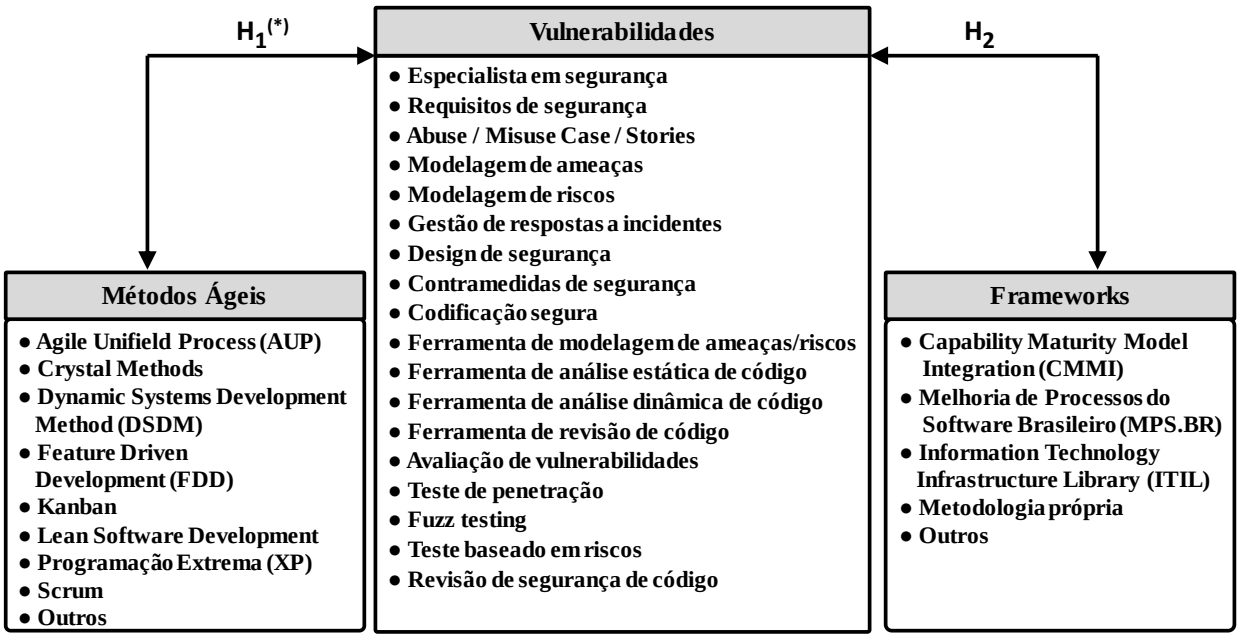
O modelo de referência para análise da relação entre o uso de métodos ágeis e *frameworks* de qualidade de software com o interesse em TFDV está apresentado na Figura 1 e foi concebido para atender ao objetivo desta pesquisa. As variáveis presentes no modelo de pesquisa foram criadas a partir da revisão bibliográfica realizada na Seção 2 deste trabalho. O modelo é composto por três conjuntos de variáveis:

- Métodos ágeis. Representam os diversos métodos ágeis descritos na literatura (Sommerville, 2011).
- Vulnerabilidades. Trata dos tipos de vulnerabilidade encontrados na literatura.
- *Frameworks*. Representa os principais *frameworks* relacionados à qualidade em software e serviços na área de sistemas de informação.

Com base no modelo de pesquisa foram enunciadas as seguintes hipóteses:

(H1) Relaciona a adoção de métodos ágeis com o interesse em TFDV e apresenta o seguinte enunciado: "equipes de desenvolvimento de software que adotam métodos ágeis têm interesse no uso de TFDV".

(H2) Relaciona a adoção de *frameworks* de qualidade de software com o interesse em TFDV e apresenta o seguinte enunciado: "equipes de desenvolvimento de software que adotam *frameworks* de qualidade de software têm interesse no uso de TFDV".



Legenda: (*) H_1 e H_2 são as hipóteses da pesquisa

Figura 1: Modelo da referência da pesquisa

Estas hipóteses foram estabelecidas com base na premissa de que membros de equipes que usam métodos ágeis e *frameworks* de qualidade de software têm maior preocupação com a qualidade do produto e, portanto, devem ter interesse por TFDV.

4 Método da pesquisa

Esta seção apresenta os procedimentos metodológicos que foram aplicados à pesquisa. A pesquisa proposta neste trabalho se caracteriza por ser um estudo descritivo (Wrightman, Cook, & Selltiz, 1976). Este tipo de estudo tem como objetivo verificar a frequência com que algo ocorre e descobrir ou verificar a ligação entre determinadas variáveis. Trata-se de uma pesquisa quantitativa e transversal do tipo único, pois a coleta de informação foi feita uma única vez.

4.1 Fases da pesquisa

Esta pesquisa foi desenvolvida em quatro fases. A primeira fase constituiu o levantamento bibliográfico apresentado na Seção 2. Esse levantamento apresentou e descreveu os métodos ágeis e as vulnerabilidades no desenvolvimento de software.

A segunda fase definiu um modelo de referência da pesquisa para análise do uso de TFDV. A construção desse modelo partiu dos conceitos apresentados na revisão bibliográfica e o modelo está apresentado na Seção 3.

Na terceira fase, foi realizado um *survey* com 111 profissionais que atuam em equipes de desenvolvimento de software. Para a realização do *survey* escolheram-se membros de equipes de desenvolvimento de software que utilizam métodos ágeis. Foram estabelecidas quatro funções diretamente envolvidas na implantação das TFDV no projeto: (1) analista de requisitos; (2) arquiteto de software; (3) desenvolvedor; e (4) testador. Algumas organizações da amostra possuem membros diferentes dos estabelecidos acima, ou ainda membros específicos de equipes ágeis, tais como especialistas em segurança, *scrum masters product owners* e *agile coaches*. Sendo assim, estas funções complementam as quatro funções inicialmente definidas.

Na quarta fase, os dados sobre as características dos respondentes foram apresentados, bem como as análises sobre o uso de TFDV. Os resultados obtidos e as conclusões da pesquisa estão apresentadas nas Seções 5 e 6, respectivamente.

4.2 População e amostra

As unidades de análise desta pesquisa são os membros das equipes de desenvolvimento de software pesquisados. As unidades de observação são o uso das técnicas ou ferramentas de detecção de vulnerabilidade, e o escopo da pesquisa abrangeu organizações brasileiras.

Optou-se por uma amostra não probabilística, com procedimento de amostragem por conveniência, pois essas características são adequadas para a obtenção de informações com custo menor (Aaker, Kumar, & Day, 2004). Obteve-se uma amostra com 111 profissionais que atuam em equipes de desenvolvimento de software.

4.3 Coleta e análise dos dados

Os dados coletados são do tipo primário, ou seja, são aqueles que não foram antes coletados. Adotou-se nesta pesquisa o questionário estruturado como instrumento de coleta de dados. A vantagem desse instrumento está no custo de aplicação e na uniformidade de mensuração. Além de ser a melhor forma de coletar informações de muitos respondentes (Aaker, Kumar, & Day, 2004). Os dados foram coletados no segundo semestre de 2017.

4.4 Análise e Tratamento dos Dados

A análise de dados foi realizada em duas etapas. Na primeira etapa foram utilizadas estatísticas descritivas. O objetivo dessa etapa foi descrever as características da amostra. Na segunda e última etapa utilizou-se a medida de associação Lambda e Gama, voltada para variáveis nominais e ordinais, respectivamente, de acordo com Kinnear e Taylor (1979), resumindo os procedimentos estatísticos para análise bivariada. Estas medidas de associação foram usadas com o objetivo de verificar as hipóteses de pesquisa e estão descritas a seguir:

- Medida de Associação Lambda ($\underline{\lambda}$). A estatística Lambda mostra a proporção pela qual se reduz o erro de predição da variável dependente quando se usa a variável independente. O cálculo é feito por meio da seguinte fórmula:

$$\lambda = \frac{(Erro\_1_{var1\text{-}var2} - Erro\_2_{var1\text{-}var2}) + (Erro\_1_{var2\text{-}var1} - Erro\_2_{var2\text{-}var1})}{(Erro\_1_{var1\text{-}var2}) + (Erro\_1_{var2\text{-}var1})}$$

Onde

$Erro\_1_{var1\text{-}var2}$ = Número de erros de predição da variável 2 sem ter informação da variável 1.

$Erro\_2_{var1\text{-}var2}$ = Número de erros de predição da variável 2 tendo informação da variável 1.

Erro_1 $_{var2\text{-}var1}$ = Número de erros de predição da variável 1 sem ter informação da variável 2.

Erro_2 $_{var2\text{-}var1}$ = Número de erros de predição da variável 1 tendo informação da variável 2.

- Medida de Associação Gama (γ). A estatística Gama não só indica a força de associação entre as variáveis, mas também sua direção. Sua mensuração baseia-se no cálculo de pares concordantes e discordantes para todos os pares de observação das variáveis. O cálculo é feito por meio da seguinte fórmula:

$$\gamma = \frac{(P - Q)}{(P + Q)}$$

Onde: P = número de pares concordantes

Q = número de pares discordantes

Para auxiliar a interpretação dessas estatísticas, Babbie, Halley e Zanino (2000) associaram intervalos de valores a uma medida de força da associação, conforme indicado na Tabela 1.

Força da associação	Medida de Associação
Nenhuma	0,00
Fraca	± 0,01 a 0,09
Moderada	± 0,10 a 0,29
Forte	± 0,30 a 0,99
Perfeita	1,00

Tabela 1. Interpretação de medidas de associação

Fonte: adaptada de Babbie *et al*. (2000, p. 233)

5 Análise e resultados

A análise dos dados e os resultados estão apresentados em quatro tópicos: (1) Características dos Respondentes; (2) Características do Ambiente; (3) Resultado das Hipóteses de Pesquisa; e (4) Análise dos Resultados. A amostra inicialmente obtida foi de 111 profissionais. Porém, na fase de verificação dos dados, 35 casos foram eliminados por não terem dados completos, ou não atenderem a critérios de qualidade, restando uma amostra válida de 76 profissionais.

5.1 Características dos respondentes

As características da amostra foram analisadas em relação aos respondentes e ao ambiente de desenvolvimento de software. As características dos respondentes estão apresentadas na Tabela 2. Em relação à função exercida pelos respondentes, a maioria é representada por desenvolvedores e *agile coaches*, que representam mais da metade da amostra (59,2% = 35,5%+23,7%). Profissionais ocupando mais de uma função representam 15,8%, o que permite inferir que algumas empresas são de pequeno porte e, por isso, os profissionais acabam acumulando funções. As demais funções foram pouco representativas na amostra.

Variáveis	Categorias ou níveis	Frequências	
		Absoluta	Relativa
Função	Especialista em segurança	5	6,6
	Analista de requisitos	2	2,6
	Arquiteto de software	2	2,6
	Desenvolvedor	27	35,5
	Testador	7	9,2
	Agile coach	18	23,7
	Outros	3	3,9
	Vários	12	15,8
	Total	76	100,0
Tamanho da equipe	0 a 5 anos	39	51,3
	6 a 10 anos	18	23,7
	Acima de 10 anos	19	25
	Total	76	100,0
Tempo na função	0 a 5 anos	39	51,3
	6 a 10 anos	31	40,8
	Acima de 10 anos	6	7,9
	Total	76	100,0

Tabela 2. Características do respondente e da equipe

Fonte: dados da pesquisa

As equipes de desenvolvimento de software são na maioria de pequeno porte (51,3%), ou seja, com até cinco integrantes. Isso é característico de pequenas empresas ou de equipes que adotam métodos ágeis. Por último, destaca-se a experiência na função. A maioria (51,3%) possui pouca experiência (até 5 anos) e apenas 7,9% da amostra possui muita experiência na função.

5.2 Características do ambiente

As características do ambiente de desenvolvimento de software estão apresentadas na Tabela 3. Em relação à tecnologia, a opção *web* foi a mais adotada. Ela esteve presente em praticamente todos os ambientes (93,4%). Em segundo lugar cabe destacar a opção mobile, que esteve presente em 39,4% (19,7%+19,7%) das empresas pesquisadas. Isso caracteriza os ambientes de desenvolvimento de software presentes na amostra com predomínio do uso de web e com recursos mobile. Em relação aos métodos ágeis, o Scrum foi o mais utilizado. Esteve presente em 93,5% das empresas. Por outro lado, os *frameworks* mais adotados pelo mercado em relação à qualidade de software e serviços estiveram pouco presentes nas empresas pesquisadas. Apenas 15,8% delas adota um dos *frameworks*. Mais de 40% não adotam nenhum e outros 38,2% adotam uma metodologia própria.

Características adotadas no ambiente	Categorias ou níveis	Frequências	
		Absoluta	Relativa
Tecnologia	Web	33	43,4
	Web + mobile	15	19,7
	Web + ecommerce	8	10,5
	Web + mobile + ecommerce	15	19,7
	Outro	5	6,6
	Total	76	100,0
Método ágil	Só Scrum	23	30,3
	Scrum + Outro	48	63,2
	Só XP	2	2,6
	Só kanban	2	2,6
	Outro	1	1,3
	Total	76	100,0

Características adotadas no ambiente	Categorias ou níveis	Frequências	
		Absoluta	Relativa
Framework	Nenhum	31	40,8
	Metodologia própria	29	38,2
	CMMI ou MPS.Br ou ITIL	12	15,8
	Outro	2	2,6
	Sem resposta	2	2,6
	Total	76	100,0

Tabela 3. Características do ambiente de desenvolvimento de software

Fonte: dados da pesquisa

5.3 Resultado das hipóteses de pesquisa

A análise da relação entre adoção de métodos ágeis e *frameworks* de qualidade de software com TFDV foi feita por meio da análise das hipóteses de pesquisa H1 e H2. As hipóteses foram verificadas pela aplicação da técnica estatística *Fisher´s Exact Test*, e só foram consideradas as relações com nível de significância estatística menor ou igual a 5%. Os resultados da análise estão sumarizados na Tabela 4.

Técnicas e ferramentas de detecção de vulnerabilidades	Método ágil adotado (gama)	*Frameworks* Adotado (lambda)
Especialistas em segurança	-0,169	0,244 *
Requisitos de segurança	-0,140	0,145
Abuse / Misuse Case / Stories	-0,051	0,193 *
Modelagem de ameaças	-0,128	0,170
Modelagem de riscos	-0,104	0,209 *
Gestão de respostas a incidentes	-0,210	0,175 **
Design de segurança	-0,111	0,159
Contramedidas de segurança	-0,167	0,222 *
Codificação segura	-0,255	0,057
Ferramenta de modelagem de ameaças / riscos	-0,187	0,159
Ferramenta de análise estática de código	-0,128	0,013
Ferramenta de análise dinâmica de código	-0,149	0,027
Ferramenta de revisão de código	-0,271 **	0,028
Avaliação de vulnerabilidades	-0,127	0,093
Teste de penetração	-0,163	0,123
Fuzz testing	-0,334 *	0,205 *
Teste baseado em riscos	-0,297 *	0,184 **
Revisão de segurança de código	-0,276 **	0,146

(*) p value < 005; (**) p value < 0,10

Tabela 4. Medidas de associação Gama e Lambda

Fonte: tratamento de dados da pesquisa

A hipótese H1 relaciona a adoção de métodos ágeis com o interesse em TFDV. O interesse por TFDV teve pouca relação com o uso de métodos ágeis. Somente em quatro das 18 técnicas houve uma correlação estatisticamente significativa. Além disso, todas as medidas de associação (gama) foram negativas, ou seja, quanto mais métodos ágeis adotados pela equipe menor o interesse por TFDV.

A hipótese H2 relaciona a adoção de *frameworks* de qualidade de software com o interesse em TFDV. No caso dos *frameworks* ocorreu o contrário da hipótese H1, ou seja, o interesse por TFDV teve correlação positiva com sete das 18 técnicas. A adoção de *frameworks* de qualidade de software está mais correlacionada com o interesse em TFDV do que com a adoção de métodos ágeis.

5.4 Análise dos resultados

A análise dos resultados está apresentada em duas partes: a primeira trata das relações com os métodos ágeis (H1); e a segunda trata das relações com os *frameworks* de qualidade de software (H2). Foram observadas quatro relações estatisticamente significativas entre métodos ágeis e TFDV. Todas as quatro relações se mostraram inversas, ou seja, quanto mais métodos ágeis adotados pela equipe menor o interesse em TFDV. A seguir a descrição das relações que estão sumarizadas na Tabela 5.

Técnicas e ferramentas de detecção de vulnerabilidades	**Níveis**	**Métodos ágeis**			
		Só um	**Até 2**	**Até 3**	**> 3**
Ferramenta de revisão de código	Baixo	37,5	6,3	25,0	31,3
	Médio	7,7	61,5 *	23,1	7,7
	Alto	44,7 **	27,7	10,6	17,0
Fuzz testing	Baixo	27,8	27,8	16,7	27,8 *
	Médio	33,3	33,3	25,0	8,3
	Alto	50,0 **	28,6	10,7	10,7
Teste baseado em riscos	Baixo	24,1	27,6	17,2	31,0 *
	Médio	47,1	23,5	23,5	5,9
	Alto	43,3	33,3	10,0	13,3
Revisão de segurança de código	Baixo	28,0	20,0	20,0	32,0 *
	Médio	37,5	37,5	12,5	12,5
	Alto	42,9	31,4	14,3	11,4
Total da amostra		41,9	39,2	16,2	2,7

(*) p value < 005; (**) p value < 0,10

Tabela 5. Relação entre TFDV e os métodos ágeis

Fonte: tratamento de dados da pesquisa

- Ferramenta de revisão de código. Equipes que têm interesse em TFDV adotam com mais frequência apenas um método ágil. No caso da amostra de pesquisa, este método ágil é o Scrum. Por outro lado, as equipes que têm um interesse médio em TFDV adotam com mais frequência dois métodos ágeis. Isso mostra que quanto maior o uso de diversos métodos ágeis menor é a frequência de interesse por esta TFDV.
- *Fuzz testing*. De forma similar ao item anterior, equipes que têm interesse em TFDV adotam com mais frequência apenas um método ágil. Além disso, equipes que têm pouco interesse por esta TFDV adotam com mais frequência vários métodos ágeis. Neste caso também, o resultado mostra que quanto maior o uso de diversos métodos ágeis menor a frequência de interesse por *fuzz testing*.
- Teste baseado em riscos. Equipes que têm pouco interesse por esta TFDV adotam com mais frequência vários métodos ágeis.
- Revisão de segurança de código. De forma similar ao item anterior, equipes que têm pouco interesse por esta TFDV adotam com mais frequência vários métodos ágeis.

Foram observadas sete relações estatisticamente significativas entre *frameworks* de qualidade de software e TFDV. Todas as sete relações apresentaram uma correlação direta, ou seja, quanto mais *frameworks* adotados pela equipe maior o interesse em TFDV.

- Observa-se pela Tabela 6 que as equipes que não usam *frameworks* de qualidade de software têm menor interesse por TFDV.
- O interesse por TFDV se relacionou mais com o uso de *frameworks* de qualidade de software do que com o uso de métodos ágeis.
- As equipes de desenvolvimento de software que têm mais interesse no uso de TFDV adotaram mais *frameworks* próprios ou outros *frameworks* do que os destacados na literatura.

- Outras metodologias foram fortemente relacionadas a um alto grau de interesse por TFDV. Ao se analisar quais são essas outras metodologias, verificou-se que se tratava de duas: OWASP *Testing Guide e Fortify*. Ou seja, metodologias voltadas para o teste de software. Isso mostra a coerência dos resultados, pois as equipes que relataram um maior interesse por TFDV adotam métodos específicos para teste de software.

Técnicas e ferramentas de detecção de vulnerabilidades	**Níveis**	***Frameworks***			
		Nenhuma	**Método próprio**	**CMMI ou MPS.Br ou ITIL**	**Outros**
Especialista em segurança	Baixo	64,5 *	25,8	9,7	0,0
	Médio	28,6	57,1	14,3	0,0
	Alto	24,1	44,8	24,1	6,9 **
Abuse / Misuse Case / Stories	Baixo	61,8 *	29,4	8,8	0,0
	Médio	37,5	43,8	18,8	0,0
	Alto	16,7	50,0	25,0	8,3 *
Modelagem de riscos	Baixo	65,4 *	26,9	7,7	0,0
	Médio	41,2	41,2	17,6	0,0
	Alto	22,6	48,4	22,6	6,5 *
Gestão de respostas a incidentes	Baixo	60,9 *	30,4	8,7	0,0
	Médio	57,1	28,6	14,3	0,0
	Alto	24,3	48,6 **	21,6	5,4
Contramedidas de segurança	Baixo	60,0 *	33,3	6,7	0,0
	Médio	50,0	12,5	37,5 **	0,0
	Alto	20,0	50,0 **	19,4	5,6
Fuzz testing	Baixo	58,8 *	26,5	14,7	0,0
	Médio	33,3	33,3	25,0	8,3
	Alto	25,0	57,1 *	14,3	3,6
Teste baseado em riscos	Baixo	60,7	28,6	10,7	0,0
	Médio	31,3	37,5	31,3 **	0,0
	Alto	30,0	50,0	13,3	6,7 **
Total da amostra		41,9	39,2	16,2	2,7

(*) p value < 005; (**) p value < 0,10

Tabela 6. Relação entre TFDV e os métodos ágeis

Fonte: tratamento de dados da pesquisa

6 Conclusões

O objetivo deste trabalho foi identificar e descrever o interesse dos membros de equipes que usam métodos ágeis no desenvolvimento de software na utilização de técnicas e ferramentas para detecção de vulnerabilidades (TFDV). Para isso, foi realizado um *survey* envolvendo 76 profissionais que atuam em equipes de desenvolvimento de software. A pesquisa tem limitações da qual se destaca o fato de a amostra não ser aleatória e, portanto, os resultados não podem ser generalizados. As conclusões estão apresentadas em três tópicos:

- Interesse no uso de TFDV. O interesse no uso de cada um das 18 TFDV foi médio. Pode-se inferir que o interesse por TFDV não tem pautado o desenvolvimento de software no Brasil, criando oportunidades para melhoria na prevenção e detecção de vulnerabilidades.
- Relação entre o interesse por TFDV e a adoção de métodos ágeis. O maior interesse em TFDV ficou associado ao uso de Scrum. Porém, quanto mais método ágeis adotados pela equipe de desenvolvimento de software, menor o interesse por TFDV.
- Relação entre o interesse por TFDV e a adoção de *frameworks* de qualidade de software e serviços. Os *frameworks* de qualidade de software tiveram uma relação mais direta com o interesse em TFDV.

Equipes que adotaram *frameworks* tiveram maior interesse por TFDV e aquelas que adotaram *framework* proprietários ou relacionado a teste de software tiveram um nível de interesse maior ainda.

Esta pesquisa faz parte de um projeto iniciado em 2016 com o objetivo de analisar as relações entre o uso de TFDV e métodos ágeis e *frameworks* de qualidade de software. Os resultados desta pesquisa corroboram as relações sugeridas. Assim, sugerem-se futuras pesquisas que busquem confirmar, por meio de pesquisa explanatória as relações aqui identificadas.

7 Referências

Aaker, D. A., Kumar, V., & Day, G. S. (2004). *Marketing research 7ª edition*. New York: John Wileys & Sons.

Alexander, I. (2003). *Misuse cases: Use cases with hostile intent.* IEEE software, IEEE, 20(1), 58–66.

Axelos. (2018). *IT Service Management*. Disponível em: <https://www.axelos.com/best-practice-solutions/itil>. Acesso em 30/09/2017.

Babbie, E., Halley, F. & Zanino, J. (2000). *Adventures in Social Research*. California: Sage Publications.

Bach, J. (1999). Heuristic risk-based testing*. Software Testing and Quality Engineering Magazine*, 11(9), 1-10.

Beck, K. (2000). *Extreme programming explained* 2ª *edition.* São Paulo: Addison -Wesley.

CERT.br. (2012). *Cartilha de segurança para internet*: *Ataques na internet.* Disponível em: <https://cartilha.cert.br/ataques/>.

Cichonski, P., Millar, T., Grance, T.; Scarfone, K. (2012). Computer security incident handling guide*. NIST Special Publication*, 800, 61. Disponível em: <https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-61r2.pdf>.

Cigital. *Penetration testing*. (2016). Disponível em: < https://www.synopsys.com/software-integrity/security-testing/penetration-testing.html>.

CMMI. (2017). *What Is Capability Maturity Model Integration (CMMI)*®? Disponível em: <http://cmmiinstitute.com/capability-maturity-model-integration>.

Cristofoli, F., Prado, E. P. V., & Takaoka, H. (2012). Gestão da Terceirização da Tecnologia da Informação Baseada nas Práticas de Governança In: *International Conference on Information Systems and Technology Management, 2012, São Paulo*. Proceedings of the 9th International Conference on Information Systems and Technology Management.

Foster, J. S.; Fandrich, M.; Aiken, *A.* (1999). *A theory of type qualifiers*. ACM SIGPLAN Notices, 34(5), 192–203.

Freitez, W. R. J.; Mammar, A.; Cavalli, A. R. (2009). Software vulnerabilities, prevention, and detection methods: a review*.* In: *SEC-MDA: Security in Model Driven Architecture*, pp. 1–11.

Group, I. (2010*). IEEE Standard classification for software anomalies*. IEEE, New York.

Gtieskanp, W.; Gurevich, Y.; Schulte, W.; Veanes, M. (2002). Generating finite state machines*. ACM SIGSOFT*, pp. 112–122.

Howard, M., & Lipner, S. (2006). *The security development lifecycle*. Microsoft Press.

ISTQB. (2012). *Advanced Level Syllabus Test Manager*. Disponível em: <https://www.istqb.org/send/10-advanced-level-syllabus-2012/54-advanced-level-syllabus-2012-test-manager.html>.

Jones, C. (2013). *Software defect origins.* International Function Point Users Group, pp. 1–31.

Kinnear, T. C., & Taylor, J. R. (1979). *Marketing Research: an applied approach*. International Student Edition. Tokyo: McGraw-Hill.

Kim, H., Choi, T.-H., Jung, S.-C., Kim, H.-C., Lee, O., & Doh, K.-G. (2008). Applying dataflow analysis to detecting software vulnerability. In: *IEEE 10th International Conference on Advanced Communication Technology*, 1, pp. 255–258.

Leffingwell, D. (2007). *Scaling software agility: Best practices for large enterprises, 1st edition.* Pearson Education, Boston, MA.

Lindvall, M., Muthig, D., Dagnino, A., Wallin C., Stupperich, M., Kueffer, D., May, J., & Kahkonen, T. (2004). Agile software development in large organizations. *IEEE Computer*, 37(12), 26–34.

Mcdermott, J., & Fox, C. (1999). Using abuse case models for security requirements analysis. In: *IEEE Proceedings of 15th Annual Computer Security Applications Conference*, pp. 55–64.

Mcgraw, G., & Morrisett, G. (2000). Attacking malicious code. In: *IEEE Computer Society*, pp. 1–11.

Microsoft. (2018a). *Definition of a Security Vulnerability*. Disponível em: <https://msdn.microsoft.com/en-us/library/cc751383.aspx>.

Microsoft. (2018b). *Diretrizes de codificação segura*. Disponível em: <https://msdn.microsoft.com/pt-br/library/8a3x2b7f(v=vs.110).aspx>.

Microsoft. (2018c). *Respondendo a incidentes de segurança de TI.* Disponível em: < https://technet.microsoft.com/pt-br/library/cc700825.aspx>.

Miller, B. (2014). University of Wisconsin–Madison: *Fuzz testing of application reliability*. 2008. Disponível em: <http://pages.cs.wisc.edu/~bart/fuzz/>.

Mirek, J., Robert, K., Martin, P., Tomás, C., Stephen, W., Ruesva, Y., & Svoboda, M. (2014). *A guide to securing red hat enterprise Linux. RED*, v. 1.

NIST. (2009). *National vulnerability database.* Disponível em:<https://www.nist.gov/programs-projects/national-vulnerability-database-nvd>.

Northcutt, S., Shenk, J., Shackleford, D., Rosenberg, T., Siles, R., & Mancini, S. (2006). Penetration testing: Assessing your overall security before attackers do. *SANS Institute Information Security Reading Room*, 3(6), 4–7.

Ouni, A., Kula, R. G., Inoue, K. (2016). Search-based peer reviewer recommendation in modern code review. In: *IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pp. 367–377.

OWASP. (2010). *OWASP Top Ten.* Disponível em: <https://www.owasp.org/Category>.

OWASP. (2015). *Application threat modeling.* Disponível em: <https://www.owasp.org/index.php/Application_Threat_Modeling>.

PCI. *Penetration test guidance.* (2015). Disponível em: <https://www.pcisecuritystandards.org/>.

Petukhov, A., & Kozlov, D. (2008). *Detecting security vulnerabilities in web applications using dynamic analysis with penetration testing.* Computing Systems Lab, Department of Computer Science, Moscow State University, OWASP - Application Security Conference, pp. 1–16.

Pinheiro, J. M. dos S. (2011). *Ameaças e ataques aos sistemas de informação.* UniFOA, pp. 1-11.

Pressmann, R. (2009). *Engenharia de Software 7th edition.* McGraw Hill, Brasil.

Quelal, R.E., Villavicencio, M., & Mendoza, L.E. (2018) A survey of agile software development methodologies in Ecuador. *Iberian Conference on Information Systems and Technologies, CISTI*, 2018-June, pp. 1-6.

Richardson, T., & Thies, C. N. (2012). *Secure software design.* Jones & Bartlett Publishers.

Santos, L. R. *Técnicas de teste de software.* (2011). Disponível em: <http://www.inf.ufpr.br/lmperes/ci221/aula_tecnicas_teste_Luis_Renato.pdf>.

Schwaber, K. S. (2013). Um guia definitivo para o SCRUM – As regras do jogo. *Scrum.Org, Scrum Guide*, 26, 3–15.

Seacord, R. C., & Householder, A. D. (2005). *A Structured approach to classifying security vulnerabilities.* CMU SEI. Disponível em: <https://resources.sei.cmu.edu/asset_files/TechnicalNote/2005_004_001_14474.pdf>.

Sindre, G., & Opdahl, A. L. (2001). *Capturing security requirements through misuse cases.* Norsk Informatic Conference.

Softex. (2016). *Guia Geral de Software*. Disponível em: <https://www.softex.br/wp-content/uploads/2013/07/MPS.BR_Guia_Geral_Software_2016.pdf>.

Sommerville, I. (2011). *Software Engineering 9th edition*. Pearson Education, Boston, Massachusetts.

Stallings, W. (2012). *Computer security 2nd edition.* Pearson Education.

Su, Z., & Wassermann, G. (2006). The essence of command injection attacks in web applications. *ACM SIGPLAN Notices*. 41(1), 372–382.

Veracode. (2017). *CA Technologies - Code review tools.* Disponível em: <https://www.veracode.com/security/code-review>.

Verdon, D., & McGraw, G. (2004). *Risk analysis.* Security & Privacy, IEEE, 79–84.

Vianna, E. W., & Fernandes, J. H. C. (2015). *O gestor da segurança da informação no espaço cibernético governamental: Grandes desafios, novos perfis e procedimentos. Brazilian Journal of Information Science*, 9(1), 1–28.

Weber, T. S. (2003). *Tolerância a falhas: Conceitos e exemplos.* Porto Alegre: Programa de Pós-Graduação, Instituto de Informática-UFRGS.

Wiesmann N, A., Stock, A. V. D., Curphey, M., Stirbei, R., Kang, A., Russell, A., Klein, A., Greidanus, B., Todd, C., & Grundy, D. et al. (2005). *A guide to building secure web applications and web services. The Open Web Application Security Project*, 2(1), 31–57.

Wrightman, L. S., Cook, S. W., & Selltiz, C. (1976). *Research Methods in Social Relations*. Publisher: Holt, Rinehart & Winston.

Zhioua, Z., Short, S., & Roudier, Y. (2014). Static code analysis for software security verification: Problems and approaches. In: IEEE. 38th International on Computer Software and Applications Conference Workshops, pp. 102–109.

Lígia C. M. de C. Santos

Mestre em Sistemas de Informação pela Universidade de São Paulo com pesquisa voltada para o uso de técnicas e ferramentas para detecção de vulnerabilidades no desenvolvimento ágil de software. Atuou profissionalmente em multinacionais como Accenture e Cognizant. Ocupa atualmente a posição de Scrum Master em empresa de medicina diagnóstica na América Latina.

Edmir P. V. Prado.

Professor Assistente da Universidade de São Paulo. Tem mais de trinta anos de experiência em Administração e Sistemas de Informação na área acadêmica e em empresas multinacionais. Atua principalmente com gestão de projetos e processos, e sourcing da tecnologia de informação. Possui graduação em Engenharia Mecânica (EPUSP) pela Universidade de São Paulo (1985), mestrado em Administração pela Universidade de São Paulo (2000) e doutorado em Administração pela Universidade de São Paulo (2005).

Marcos Lordello Chaim

Possui graduação em Engenharia Elétrica (1987), mestrado (1991) e doutorado em Engenharia Elétrica (2001) pela Universidade Estadual de Campinas. Foi professor assistente da Universidade Estadual Paulista Júlio de Mesquita Filho (UNESP). Atualmente é professor da Escola de Artes, Ciências e Humanidades da Universidade de São Paulo. A linha principal de pesquisa é em Engenharia de Software, mais especificamente nos temas relacionados a teste e manutenção de software, métodos de desenvolvimento, e engenharia de software experimental.