Pós-Graduação em Ciência da Computação

NORTON JERZEWSKI NORO

SCUM-PSP-IF: UMA PROPOSTA DE INTEGRAÇÃO DE SCRUM E PSP

Universidade Federal de Pernambuco
posgraduacao@cin.ufpe.br
www.cin.ufpe.br/~posgraduacao

RECIFE

2017

Norton Jerzewski Noro

SCUM-PSP-IF: uma proposta de integração de SCUM e PSP

Este trabalho foi apresentado à Pós-Graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Mestre Profissional em Ciência da Computação.

ORIENTADOR: **Hermano Perrelli de Moura**

RECIFE

2017

Catalogação na fonte
Bibliotecária Monick Raquel Silvestre da S. Portes, CRB4-1217

N852s Noro, Norton Jerzewski
SCUM-PSP-IF: uma proposta de integração de Scrum e PSP / Norton Jerzewski Noro. – 2017.
209 f.: il., fig., tab.

Orientador: Hermano Perrelli de Moura.
Dissertação (Mestrado) – Universidade Federal de Pernambuco. CIn, Ciência da Computação, Recife, 2017.
Inclui referências, apêndices e anexos.

1. Engenharia de software. 2. Métodos ágeis. I. Moura, Hermano Perrelli de (orientador). II. Título.

005.1 CDD (23. ed.) UFPE- MEI 2018-086

Norton Jerzewski Noro

SCUM-PSP-IF: uma proposta de integração de Scrum e PSP

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Pernambuco, como requisito parcial para a obtenção do título de Mestre Profissional em 22 de agosto de 2017.

Aprovado em: 22/08/2017.

BANCA EXAMINADORA

Prof. Alexandre Marcos Lins de Vasconcelos

Centro de Informática / UFPE

Prof. Célio Andrade de Santana Júnior

Centro de Artes e Comunicação / UFPE

Prof. Hermano Perrelli de Moura

Centro de Informática / UFPE

(Orientador)

AGRADECIMENTOS

Ao Instituto Federal Farroupilha, por oportunizar, patrocinar e incentivar a qualificação dos seus servidores, não medindo esforços para contribuir com o andamento do mestrado profissional em ciência da computação.

À minha família, meu pai Ironi, minha mãe Solange, meus irmãos Marlon e Sharlon e à minha namorada, Ana Elise, por se manterem pacientes perante as minhas inquietações, me apoiando incondicionalmente.

Aos meus orientadores, professores Hermano e Wylliams, por não medirem esforços para me guiar pelos obstáculos que um mestrado profissional oferece, entendendo perfeitamente o contexto de um aluno que, concomitantemente, conduz seus estudos e seu trabalho.

Aos meus colegas de mestrado da turma de gestão de TI, de todas as partes do Brasil, que com suas marcantes personalidades, formaram um diferenciado grupo de amigos.

RESUMO

Pesquisas evidenciam que a qualidade de *software* ainda está muito abaixo do esperado pelos seus usuários. Atualmente, empresas que desenvolvem e mantêm sistemas têm adotado modelos de qualidade nos seus processos de desenvolvimento, juntamente com metodologias ágeis, com o objetivo de aperfeiçoar os processos e gerenciar com mais eficiência os projetos e as equipes. Porém, para alcançar melhores resultados, que realmente atendam às necessidades de seus clientes, muitas organizações estão apostando também na utilização de técnicas consideradas mais tradicionais e disciplinadas para capacitar, individualmente, seus engenheiros de *software*. Uma das principais técnicas é o *Personal Software Process.* Neste contexto, essa dissertação de mestrado apresenta um modelo, instanciado e aperfeiçoado de um modelo existente, que visa integrar duas estruturas distintas de desenvolvimento de *software*: uma metodologia dirigida a plano – PSP e um *framework* focado na agilidade – *Scrum*, na busca pela melhoria na qualidade do desenvolvimento ágil de *software*, através de eficientes técnicas de gerenciamento de defeitos do PSP. O trabalho também inclui um estudo de caso, que visa aplicar o modelo integrado SCRUM-PSP-IF em um ambiente real de projeto, no Instituto Federal Farroupilha, onde mudanças de requisitos de *software* ocorrem com frequência, o desenvolvimento deve ser realizado de forma rápida e as entregas têm prazos pouco negociáveis, geralmente regidos pela legislação federal. Cinco medidas de qualidade do PSP foram utilizadas para representar os resultados da pesquisa (PQI, A/FR, Densidade de Defeitos, Taxa de Revisão e Rendimento das Fases). O SCRUM-PSP-IF inseriu disciplina nas práticas ágeis de desenvolvimento de *software*, melhorando significativamente o desempenho dos programadores e reduzindo o número de erros presentes nos testes e no produto final.

Palavras-chave: *Personal software process*. *Scrum*. *M*étodos ágeis. Metodologias disciplinadas. Integração de processos de *software*. Qualidade de *software*.

ABSTRACT

Research evidences that the quality of software is still much lower than expected by its users. Currently, companies that develop and maintain systems have adopted quality models in their development processes, along with agile methodologies, with the objective of improving processes and more efficiently managing projects and teams. However, in order to achieve better results that truly meet the needs of their customers, many organizations are also betting on techniques that are considered more traditional and disciplined to individually empower their software engineers. One of the main techniques is the Personal Software Process. In this context, this masters dissertation presents an instantiated and improved model of an existing model, which aims to integrate two distinct software development structures: a methodology directed to the plan - PSP and a framework focused on agility - Scrum, in the search by improving the quality of agile software development through efficient PSP defect management techniques. The work also includes a case study, which aims to apply the SCRUM-PSP-IF integrated model in a real project environment, at the Instituto Federal Farroupilha, where changes in software requirements occur frequently, development should be carried out quickly and deliveries have little negotiable deadlines, usually governed by federal law. Five PSP quality measures were used to represent the results of the research (PQI, A / FR, Defect Density, Revision Rate and Phase Yield). SCRUM-PSP-IF has introduced discipline in agile software development practices, significantly improving programmers' performance and reducing the number of errors in the tests and the final product.

Keywords: *Personal software process. Scrum. Agile methods. Disciplined methodologies. Integration of software processes. Software quality*.

LISTA DE FIGURAS

LISTA DE QUADROS

SUMÁRIO

1 INTRODUÇÃO

Atualmente, de forma rápida e dinâmica, as organizações estão aumentando significativamente sua dependência das tecnologias da informação e comunicação. Dessa forma, todas as suas atividades e processos de negócio estão sendo direcionados por um conjunto cada vez maior de sistemas de informação. Portanto, quem se propuser a desenvolver *software*, deve ter a capacidade e os recursos necessários para criar um produto de qualidade.

Análises resumidas do relatório *Chaos* 2015 do *Standish Group* – que no período de 2011 a 2015 registrou o resultado de aproximadamente 50.000 projetos de *software* – representadas no artigo de Lynch (2015), demonstram que:

- 29% dos projetos foram bem sucedidos, entregues no tempo combinado, dentro do orçamento estipulado e com satisfação dos usuários;

- 52% dos projetos foram parcialmente bem sucedidos, entregues após o tempo previsto e/ou extrapolando o orçamento inicial e/ou com menos recursos que o combinado;

- 19% dos projetos foi um fracasso total, sendo cancelados antes da entrega ou entregues e nunca usados.

As porcentagens indicam, claramente, que ainda há muito esforço a ser feito para alcançar resultados mais bem sucedidos nos projetos de desenvolvimento de *software*, pois, até 2015, 71% dos projetos não apresentaram a qualidade desejada pelos usuários finais, o que ocasiona perda de tempo e dinheiro, além da frustração de adquirir um produto que não condiz com as necessidades de negócio.

Neste cenário, onde a grande maioria dos projetos é entregue com pouca qualidade, todas as variáveis envolvidas no processo de desenvolvimento de *software* têm um nível crescente de importância. Paralelo a isso, os riscos de mau funcionamento aumentam proporcionalmente, de acordo com a complexidade desse ambiente, tornando mais difícil produzir *softwares* com o nível de qualidade desejado pelos seus usuários.

Para padronizar e melhorar os níveis de qualidade nos processos de engenharia de *software* e, consequentemente, obter o sucesso nos projetos e a diminuição dos defeitos no produto, algumas normas e modelos foram criados, como por exemplos as normas ISO/IEC 25000,

ISO/IEC 12207 e ISO 33000, bem como as normas *Capability Maturity Model Integration – Software* (CMMI-SW), *Team Software Process* (TSP) e *Personal Software Process* (PSP).

Assim como existem esforços para criar normas e padrões para o desenvolvimento de *software*, como as citadas anteriormente, os métodos ágeis, como *Scrum*, *Feature Driven Development* (FDD), *eXtreme Programming* (XP) e *Dynamic System Development Model* (DSDM) vêm se disseminando de forma gradativa, com o objetivo de acelerar o desenvolvimento de *software*, fornecer respostas rápidas às mudanças de necessidades dos usuários, não exigir muito investimento financeiro e tempo de implantação, além de ser mais facilmente aplicável a pequenas e médias empresas.

1.1 Motivação

O cenário desafiador da indústria de *software* mostra que as empresas desse ramo devem investir mais recursos na ampliação de suas capacidades de desenvolvimento, primando por estratégias que envolvam os usuários nos processos de tomada de decisão e coleta de informações do projeto, através de comentários, revisão de requisitos, prototipagem ou qualquer outra forma de construção de consenso, sendo este um dos três fatores mais críticos de sucesso dos projetos de *software*. Os outros dois fatores são o suporte executivo (financeiro e emocional), fornecido pelas autoridades máximas da empresa, e a maturidade emocional dos participantes dos projetos. (LYNCH, 2015)

Possuir uma equipe bem estruturada e capacitada, bem como alocar corretamente os recursos tecnológicos para o trabalho contribuem para o sucesso de um projeto de *software*. Porém, melhores resultados podem ser alcançados utilizando metodologias que fomentem a agilidade (VERSIONONE, 2015, 2016). Nesse ambiente ágil, após a codificação de uma ou mais funcionalidades, um incremento do produto deve estar disponível para a demonstração ao cliente.

No entanto, dificilmente as respostas aos principais desafios do desenvolvimento moderno de *software* são encontradas utilizando apenas as técnicas de uma única metodologia ou processo, pois assim como um conjunto de técnicas possui vantagens e benefícios, também possui algumas limitações.

Um modelo de trabalho que faz uso intensivo de fases de planejamento, produzindo um grande número de documentos e mantendo histórico completo do projeto, pode ser mais

indicado para equipes que priorizem estimar, com mais precisão, o tamanho e o esforço necessários para a conclusão do trabalho, prevendo melhor o tempo de trabalho. Enquanto que um modelo de trabalho focado no desenvolvimento ágil, onde nem todos os requisitos podem ser conhecidos no início do projeto ou, ainda, podem se comportar de forma instável no decorrer do projeto pode ser mais aderente para equipes que respondam melhor a mudanças, onde a implementação das funcionalidades tem mais importância do que o uso de processos bem definidos.

Nesses últimos anos a integração entre diferentes processos de desenvolvimento de *software* é uma área de pesquisa em ascensão (SHEN et al., 2013), com destaque para os benefícios da integração entre métodos ágeis, especialmente o *Scrum*, e o PSP, que pode melhorar a capacidade individual de cada engenheiro de *software* (WENGRONG, SHEN, 2011).

Com base na revisão sistemática da literatura sobre integração de PSP e outros modelos feita por Shen (2013), podemos dizer que ainda se pesquisa muito sobre *Scrum* e *Personal Software Process* isoladamente, como se não pudessem ser utilizados de forma integrada. *Scrum* é o *framework* ágil para gestão de projetos e equipes mais utilizado na indústria de *software* (VERSION ONE, 2016). Já o PSP é uma metodologia utilizada para medir e analisar o trabalho individual do engenheiro de *software*, além de ter como base para a sua concepção um modelo de qualidade para processos de *software* consolidado mundialmente, o CMM (*Capability Maturity Model*).

O SCRUM-PSP-IF (IF se refere a Instituto Federal) é um esforço para demonstrar que, realmente, *Scrum* e *PSP* podem trabalhar conjuntamente no gerenciamento de defeitos e no aumento de qualidade no desenvolvimento de *software*.

1.2 Objetivos

O objetivo geral dessa dissertação é a proposição de um modelo integrado de processos para a melhoria da qualidade no desenvolvimento de *software*, através da combinação dos pontos fortes e compatíveis de duas metodologias, PSP e *Scrum*.

Os objetivos específicos são:

- Identificar boas práticas de automelhoria e planejamento do PSP e de agilidade e simplicidade do *Scrum*.

- Implantar e avaliar os processos da integração SCRUM-PSP-IF em unidades do Instituto Federal Farroupilha.
- Obter índices de qualidade satisfatórios, com base nos valores alvo do PSP, no desenvolvimento de *software*, com foco no gerenciamento de defeitos.

1.3 Justificativa

Enquanto *PSP* se caracteriza como uma metodologia que visa melhorar a capacidade de processo de cada engenheiro de *software*, *Scrum* é um *framework* de processo à nível de equipe, focando na cooperação da equipe e adaptabilidade ao ambiente de projeto.

Combinar o *Scrum* com outras metodologias, como o *TSP*, por exemplo, não pareceu viável, pois antes que os membros possam participar de uma equipe TSP, eles devem receber treinamento (e ter conhecimento) no PSP (HUMPHREY, 200b).

A proposta do trabalho se concentra em integrar os pontos fortes desses dois modelos, em uma organização que já utiliza processos ágeis em seus projetos de *software* e possui domínio nas tecnologias utilizadas para desenvolver sistemas. O SCRUM-PSP-IF deverá, com a correta replicação, aperfeiçoar o combate a defeitos, bem como controlar, de forma efetiva, a qualidade do processo e, consequentemente, melhorar a qualidade do produto.

1.4 Metodologia de pesquisa

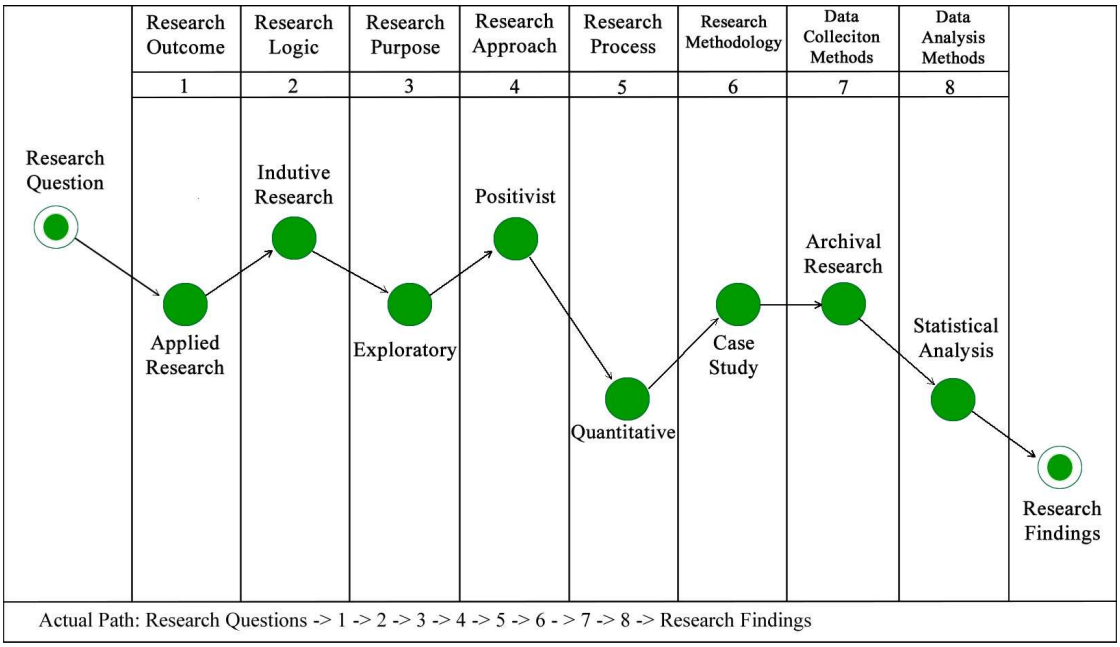
Pesquisas empíricas em engenharia de *software* têm crescido bastante nos últimos anos e uma atenção especial tem sido dada para as questões relativas à complexidade e diversidade das metodologias utilizadas. Neste trabalho, para orientar nas decisões de pesquisa será utilizada a estrutura de tomada de decisões de pesquisa de Wohlin e Aurum (2015).

A estrutura de tomada de decisões, que é formada por fases (estratégica, tática e operacional), tem início com a definição da questão (ou questões) de pesquisa, tendo como próximos passos a escolha das oito decisões de pesquisa: (i) resultado de pesquisa, (ii) lógica de pesquisa, (iii) propósito de pesquisa, (iv) abordagem de pesquisa, (v) processo de pesquisa, (vi) metodologia de pesquisa, (vii) métodos de coleta de dados e (viii) métodos de análise de dados. Os resultados (ou descobertas) devem ser atingidos ao final do último ponto de decisão. A Figura 01 mostra a estrutura de pontos de decisão desse trabalho.

1.4.1 Questão de pesquisa

É possível melhorar a qualidade no desenvolvimento ágil de *software* através da integração do *Personal Software Process* com o *Scrum*?

Figura 01 – Estrutura de tomada de decisões de pesquisa.



Fonte: Elaborado pelo autor, com base em (Wohlin e Aurum, 2015).

1.4.2 Pontos de decisão

A fase estratégica envolve os quatro primeiros pontos de decisão e visa prover subsídios ao pesquisador para a correta execução das fases tática e operacional da pesquisa, posicionando-a em relação a outras abordagens de pesquisa. O conhecimento do tema, bem como de cada ponto de decisão é fundamental para o uso de uma boa estratégia de pesquisa. (WOHLIN, AURUM, 2015).

(1) Resultado de pesquisa

Neste trabalho, o primeiro ponto de decisão corresponde à Pesquisa Aplicada, pois esse estudo é voltado para aplicação prática do conhecimento, através de um modelo de processos de *software*, visando solucionar um problema específico, que se refere à diminuição dos defeitos e a consequente melhoria da qualidade do produto de *software*. A pesquisa aplicada faz uso dos conhecimentos da pesquisa básica para, através de outros modelos, por exemplo, adaptá-los para serem aplicados em outro cenário.

(2) Lógica de pesquisa

O segundo ponto de decisão se refere à lógica de pesquisa, que indica a direção do conhecimento, se ele deve avançar da generalidade à especificidade ou o contrário. Para esse trabalho serão utilizados métodos de Pesquisa Indutiva. Parte-se da avaliação do desempenho individual do programador, até chegar a conclusões relacionadas à melhoria do projeto de desenvolvimento de *software* como um todo.

(3) Propósito de pesquisa

O terceiro ponto de decisão de pesquisa foi a Pesquisa Exploratória, que busca estudar e encontrar novos padrões e conceitos (que podem ser aplicados e replicados a um determinado problema). Uma metodologia tipicamente utilizada com a pesquisa exploratória é o estudo de caso e seus resultados podem ser qualitativos ou quantitativos (ou ambos).

(4) Abordagem de pesquisa

O quarto ponto de decisão diz respeito à abordagem Positivista, que separa o pesquisador do objeto pesquisado, focando no método de pesquisa (estudo de caso). Nesse contexto, acredita-se que a pesquisa é confiável se puder ser repetida por outro pesquisador e este conseguir chegar a conclusões semelhantes.

(5) Processo de pesquisa

O quinto ponto de decisão, bem como o sexto, constituem a fase tática da estrutura de tomada de decisões. Existem dois processos amplamente reconhecidos, pesquisa quantitativa e qualitativa. Combinados, é possível ter, ainda, um processo de abordagem mista. Esse trabalho faz uso de processo de pesquisa quantitativo.

(6) Metodologia de pesquisa

O sexto ponto de decisão selecionado foi o Estudo de Caso. Independente da inconsistência da literatura sobre os tipos de metodologias de pesquisa (WOHLIN, AURUM, 2015), a aplicação de um modelo integrado de processos de desenvolvimento de *software* deve ser realizada através de um Estudo de Caso, focando nas atividades de desenvolvimento ou na avaliação do uso de tecnologia, métodos e processos utilizados pelas partes interessadas. Além disso, as questões de pesquisa relacionadas com estas atividades devem ser adequadas para o estudo de caso, que fornece interpretação mais contextualizada e detalhada do fenômeno,

principalmente se este pode ser estudado a partir das perspectivas de mais de um participante (estudos de caso múltiplos).

No estudo de caso, a escolha de uma unidade adequada de análise (um projeto, uma equipe (ou múltiplas), um indivíduo, por exemplo) é fundamental para assegurar que a questão de pesquisa está adequadamente tratada (EASTERBROOK et al., 2008).

(7) Métodos de coleta de dados

A terceira e última fase da estrutura de tomada de decisões, a fase operacional, engloba a decisão sobre os métodos de coleta e de análise de dados. São atividades realizadas enquanto a pesquisa (estudo de caso) está sendo conduzida e após ser executada, procurando responder a questão de pesquisa.

O método de coleta de dados utilizado será a Pesquisa de Arquivos, pois representa a investigação de dados históricos arquivados por indivíduos (participantes do estudo de caso), após a aplicação do modelo integrado, envolvendo a coleta sistemática, a análise e interpretação dos dados (WOHLIN, AURUM, 2015).

(8) Métodos de análise de dados

Através do método de pesquisa de estudo de caso, os dados serão coletados de forma quantitativa em arquivos, fornecendo evidências do fenômeno estudado. Uma vez que os dados são coletados, o pesquisador precisa analisá-los, usando técnicas quantitativas (análise estatística). Para o oitavo ponto de decisão foi escolhida a análise estatística, utilizando dados numéricos. Caso seja necessário utilizar algum dado não numérico, pode-se, ainda, fazer uma conversão de qualitativo para quantitativo utilizando a análise estatística.

Quanto aos resultados da pesquisa, espera-se que o modelo integrado de processo atinja indicadores de qualidade de *software* confiáveis, garantindo a redução de defeitos nas fases finais de desenvolvimento, antes dos testes de *software* (através das revisões de projeto e código).

1.5 Organização da dissertação

Além deste capítulo introdutório, as demais partes desta dissertação estão organizadas nos seguintes capítulos:

No capítulo 2 é apresentado o referencial teórico, com os conceitos básicos para embasar e realizar a pesquisa, incluindo aspectos relacionados à integração dos modelos *PSP* e *Scrum*, buscando exemplos na academia e na indústria, visando elencar propostas similares ou complementares a esse trabalho.

No capítulo 3 é apresentado o modelo integrado SCRUM-PSP-IF, considerando o uso da metodologia *Scrum* e a personalização do *Personal Software Process* para torná-lo aderente às necessidades de uma equipe que utiliza métodos ágeis no desenvolvimento de *software*.

No capítulo 4 o modelo integrado é avaliado, através de um estudo de caso em um projeto de *software*, em um ambiente real de desenvolvimento, na Reitoria do Instituto Federal Farroupilha.

No capítulo 5, são discutidas as principais contribuições desta pesquisa e propostas de trabalhos futuros sobre o tema.

2 REVISÃO BIBLIOGRÁFICA

O objetivo desse capítulo é apresentar, através da revisão bibliográfica, a base de conhecimento necessária para o entendimento da proposta de pesquisa. Na seção 2.1 será relatado um breve histórico sobre qualidade de *software*, incluindo exemplos de modelos de qualidade de produto de *software*.

A seção 2.2 trará uma visão geral dos principais modelos de qualidade de processo de *software*. Na seção 2.3 será apresentado o PSP, seu funcionamento e níveis de processos. A seção 2.4 aborda a integração entre PSP com métodos ágeis de desenvolvimento de *software* (XP, RUP, DSDM e Scrum), além dos benefícios da combinação entre PSP e metodologias ágeis Finalizando, na seção 2.5 serão abordadas algumas metodologias ágeis, com destaque para a metodologia de maior uso mundialmente, *Scrum*.

2.1 Qualidade de *software*

Nas décadas de 50 e 60, a estratégia de qualidade da maioria das organizações industriais era baseada em testes. As equipes que trabalhavam nessa área buscavam encontrar e corrigir problemas depois da produção dos produtos(HUMPHREY, 2000a).

Nas décadas de 70 e 80, um esforço maior foi concentrado para melhorar a forma com que as pessoas realizavam seu trabalho, ao invés de concentrar os esforços nos produtos acabados. Este foco em processos de trabalho foi o responsável por importantes melhorias na qualidade da maioria dos produtos industrializados naquela época (HUMPHREY, 2000a).

Enquanto a maioria das organizações industriais já tinham adotado princípios mais modernos de qualidade, a comunidade de *software* continuou a confiar em testes como o principal método de gestão da qualidade. Porém, adicionando as inspeções de *software* nos projetos. Usando inspeções, as organizações conseguiram melhorar um pouco a qualidade dos seus produtos, mas ainda sofriam as dificuldades impostas pelo atrasado, comparado com as indústrias.

Um passo mais significativo na busca pela melhoria da qualidade de *software* foi dado com a concepção do *Capability Maturity Model* para *software* (SW-CMM), no final dos anos 80, que serviu de base para muitos modelos atuais. Um caminho não menos relevante foi trilhado com o *Personal Software Process*, a partir de 1995, assim como a versão inicial do *Team*

Software Process, criada no ano seguinte. Em 1998, o *Capability Maturity Model Integrated* (CMMI) foi concebido, propondo a ampliação do CMM.

A criação do SWEBOK (*Guide to the Software Engineering Body of Knowledge*), pelo IEEE, trouxe um razoável amadurecimento à área de engenharia de *software*, incluindo uma área de conhecimento chamada Qualidade de *Software*.

Com o objetivo de tornar viável a melhoria das capacidades de desenvolvimento de *software* no âmbito Brasileiro, a SOFTEX, em 2003, criou o programa MPS.BR (Melhoria de Processos do *Software* Brasileiro), visando consolidar um modelo de referência, onde as empresas podem implementar boas práticas de engenharia de *software* e assim, melhorar o desempenho e ganho no desenvolvimento de *software* (SOFTEX, 2012).

Para ser melhor entendida, caracterizada e contextualizada, a qualidade de *software* pode ser representada em dois grupos: modelos de qualidade de produto de *software* e modelos de qualidade de processo de *software* (SOMMERVILLE, 2011).

A ideia de modelo de qualidade para produto não faz parte do contexto desse trabalho. Porém, merece ser contextualizado, pois também faz parte da área de qualidade de *software*.

Conforme Koscianski e Soares (2007), com base em uma lista de características mensuráveis, a qualidade do produto de *software* é encontrada repetidas vezes na literatura, tendo início nas publicações de Boehm (1976) e McCall (1977). Posteriormente, diversos outros autores procuraram definir seus próprios modelos, existindo até modelos específicos para determinadas tecnologias e aplicações, como por exemplo, para *softwares* orientados a objetos ou programas educativos. Podem-se citar como exemplos desse modelo normas criadas pela ISO, em conjunto com a IEC: ISO/IEC 9126, ISO/IEC 14598 e ISO/IEC 25000 (SQuaRE).

2.2 Modelos de qualidade de processo de *software*

Para Pressman (2006) a qualidade de *software* é definida como “a conformidade a requisitos funcionais e de desempenho explicitamente declarados, a padrões de desenvolvimento claramente documentados e a características implícitas que são esperadas de todo *software* profissionalmente desenvolvido”.

Nesse contexto, a qualidade do *software* não depende apenas das técnicas e ferramentas utilizadas em seu desenvolvimento, uma vez que é possível um *software* que não apresenta

erros de codificação não ser um produto de qualidade por não atingir as expectativas do solicitante. Portanto, está ligada também ao aperfeiçoamento do processo de desenvolvimento. O processo de *software* é um conjunto de atividades, métodos, práticas e transformações que as pessoas utilizam para desenvolver e manter *software* e produtos relacionados (SEI, 2010).

Apresentaremos agora uma visão geral dos principais modelos de qualidade de processo criados pela ISO (*International Organization for Standardization*), pelo SEI (*Software Engineering Institute*) e pela SOFTEX, com maior nível de detalhamento para modelo PSP, por ser uma das partes do modelo integrado a ser criado. Assim, os modelos ISO serão resumidos em uma única seção e os modelos CMMI, MPS.BR tratados resumidamente.

2.2.1 Visão geral – normas ISO/IEC

Nessa seção serão abordadas, brevemente, três normas: ISO/IEC 12207, ISO 90003 e a ISO/IEC 33000.

A ISO/IEC 12207 estabelece uma estrutura comum para os processos de ciclo de vida de *software*. A estrutura contém processos, atividades e tarefas que servem para ser aplicadas durante a aquisição de um sistema ou serviço de *software*, e durante o fornecimento, desenvolvimento, operação e manutenção de produtos de *software* (ISO/IEC 12207). A norma brasileira correspondente é a NBR ISO 12207. Deve ser utilizada relacionando duas partes interessadas no produto, admitindo, inclusive, que as duas partes sejam da mesma organização ou por uma única parte, atribuindo tarefas a ela mesma.

A norma ISO 90003 surgiu como uma instância da ISO 9001 e fornece orientação para as organizações interessadas na aplicação para a aquisição, fornecimento, desenvolvimento, operação e manutenção de *software*. A segunda edição, de 2014, anula e substitui a primeira, de 2004. A norma brasileira correspondente é a NBR ISO/IEC 9000-3.

Já o conjunto de normas ISO/IEC 33000, publicado em 2015, tem o objetivo de fornecer uma abordagem estruturada para a avaliação de todos os processos organizacionais, não apenas os de *software*, com o objetivo de compreender o estado dos processos e tentar melhorá-los, os adequando aos requisitos da própria organização ou de outra organização (no caso de parcerias ou contratos). No contexto de desenvolvimento de *software*, o *framework* para a

avaliação abrange os processos empregados no desenvolvimento, manutenção e utilização de sistemas de TI e também os profissionais envolvidos nesse ambiente (ISO/IEC 33001).

2.2.2 CMMI

No início dos anos 90 foi criado o SW-CMM, um conjunto de melhores práticas para avaliação da maturidade de processos de desenvolvimento de *software*. Ainda nos anos 90, foi estendido para outras áreas, como gestão de recursos humanos (P-CMM), aquisição de *software* (SA-CMM) e engenharia de sistemas (SE-CMM). Com a finalidade de integrar essas estruturas, surge o CMMI (*Capability Maturity Model Integration*) ou Integração do Modelo de Maturidade em Capacitação.

O modelo CMMI consiste, basicamente, de uma coleção de melhores práticas que ajudam as organizações a melhorar seus processos. Também pode ser definido como um modelo de maturidade para melhoria de processo, destinado ao desenvolvimento de produtos e serviços (SEI, 2010).

A última versão do CMMI é a 1.3 (publicada em outubro de 2010) e apresenta três modelos: CMMI para Desenvolvimento (CMMI-DEV), CMMI para Aquisição (CMMI-ACQ) e CMMI para Serviços (CMMI-SVC). Atualmente, o SEI transferiu as atividades relacionadas ao CMMI para o Instituto CMMI, mantido pela Universidade Carnegie Mellon.

2.2.3 MPS.BR

Melhoria de Processo do *Software* Brasileiro (MPS.BR) é um programa criado para definir e aprimorar um modelo de melhoria e avaliação de processos de *software* e serviços de TI no Brasil (SOFTEX, 2012).

Constituído no final de 2003, pela Associação para Promoção da Excelência do *Software* Brasileiro (SOFTEX), busca oferecer um modelo de referência de qualidade viável e aderente à realidade das micro, pequenas e médias empresas (MPME) de *software*, incluindo as instituições que não são indústrias de *software,* mas possuem considerável demanda por sistemas computadorizados.

A base técnica para a construção da última versão do modelo, no ano de 2016, é formada pelas normas ISO/IEC 12207:2008, ISO/IEC 20000, ISO/IEC família 330xx, CMMI-SVC, P-

CMM, NBR ISO 9001:2008, PNQ e MoProSoft - NMX-I-059/2-NYCE-2011 (SOFTEX, 2016).

O programa MPS.BR possui cinco componentes principais: Modelo de Referência MPS para Software (MR-MPS-SW), Modelo de Referência MPS para Serviços (MR-MPS-SV), Modelo de Referência MPS para Gestão de Pessoas (MRMPS-RH), Método de Avaliação (MA-MPS) e Modelo de Negócio (MN-MPS). Cada componente é descrito por meio de guias e/ou documentos do Programa MPS.BR (SOFTEX, 2016).

2.3 Métodos ágeis

Os métodos ou *frameworks* ágeis surgiram como uma alternativa aos modelos tradicionais de desenvolvimento e estão sendo cada vez mais utilizadas na indústria de *software*, bem como nas empresas que necessitam desenvolver sistemas para suprir suas necessidades de negócios.

Os três principais benefícios sentidos pelas organizações a partir da adoção de abordagens ágeis são (VERSIONONE, 2015 e 2016):

- Habilidade para gerenciar mudança de prioridades;
- Aumento de produtividade da equipe;
- Melhoria da visibilidade do projeto.

Nesse contexto, o desenvolvimento ágil surgiu para acelerar a criação de *software* e responder adequadamente a qualquer tipo de mudança no projeto, utilizando estruturas iterativas de processos, promovendo entregas rápidas e a maior satisfação dos usuários.

No cenário ágil, nenhum método se destaca tanto quanto o *Scrum*, utilizado por 58% dos profissionais de TI entrevistados tanto pelo 10th quanto pelo 11th *Annual State of Agile Survey* (VERSIONONE, 2015 e 2016). *Scrum* será abordado com mais detalhes na seção 2.3.1. Curiosamente, o segundo método ágil mais utilizado, com 10%, também envolve o *Scrum*, formando um modelo híbrido com XP (VERSIONONE, 2015 e 2016). Portanto, aproximadamente 70% dos entrevistados utilizaram *Scrum* nesses dois últimos anos. Métodos ágeis populares como XP ocupam 1% na pesquisa, bem como FDD e DSDM, enquanto *Lean Development* tem 2% e *Kanban* 5%. Ainda, metodologias híbridas customizadas representam 8% da pesquisa (VERSIONONE, 2015 e 2016).

Comparando as versões de 2015 e 2016 do *State of Agile* pode-se perceber, claramente, como a abordagem ágil está cada vez mais presente nas organizações, não só na TI, mas também em setores como *marketing* e RH, que através dos fundamentos e técnicas ágeis, estão conseguindo alcançar melhores resultados e aumento de valor de negócio da unidade envolvida.

2.3.1 Scrum

O nome *Scrum* se originou da estratégia cooperativa do jogo de *Rugby*. Nesse jogo, *Scrum* é uma estratégia na qual as equipes avançam contra o adversário passando a bola para trás e para frente em sequências coordenadas. O fundamento desta técnica baseia-se no conceito de inspecionar e adaptar-se, e não se restringe a uma metodologia de projetos de *software* apenas, por isso é considerada pelos especialistas como uma ferramenta de gerenciamento ágil de projetos de qualquer natureza.

Papéis

No *Scrum* existem três papéis distintos, o do *Product Owner* (proprietário do produto), que define os requisitos que compõem o *Product Backlog* e os prioriza nas *Sprint Planning Meetings*. O papel de *Scrum Master* (mestre *Scrum*), representado pelo gerente de projeto ou alguém da equipe que tenha conhecimento amplo das atividades que serão desenvolvidas, pois tem a função de controlar e obter métricas de avanço do processo. O terceiro papel, *Scrum Development Team* (time de desenvolvimento), é composta por todos os profissionais que irão executar as tarefas das *Sprints*. O conjunto desses três papéis forma o *Scrum Team* (equipe *Scrum*).

Etapas

São cinco as etapas ou fases da metodologia *Scrum*, descritas a seguir:

Scrum Cerimonials ou *Sprints*: são os ciclos do projeto, normalmente de duas a quatro semanas. Representam um espaço de tempo, onde um conjunto de atividades deve ser executado.

Daily Scrum: diariamente, preferencialmente pela manhã, a equipe faz reuniões com o objetivo de responder três questões: O que eu fiz ontem? O que farei hoje? Existe algum problema ou impedimento que me impeça de realizar minhas tarefas? Concentrando-se nas respostas a estas perguntas, a equipe obtém uma visão ampla do trabalho já realizado e do trabalho a ser realizado, além dos possíveis impeditivos para o êxito das tarefas. Qualquer problema ou impedimento identificado nesta fase do processo *Scrum* deve ser tratado, com eficiência, pelo *Scrum Master*.

Sprint Planning Meeting: é uma reunião onde participam o *Product Owner*, *Scrum Master* e todo o *Scrum Team*, além de outras pessoas interessadas no projeto. Nesta reunião, o proprietário do produto descreve todas as funcionalidades de maior prioridade para a equipe. Os membros da equipe podem fazer perguntas que os permitam subdividir estas funcionalidades em tarefas de cunho técnico, e estas irão originar o *Sprint Backlog*. A boa relação entre o tamanho da lista e a velocidade de trabalho da equipe permite que sejam descritos apenas os itens prioritários, e que os itens de menor prioridade sejam tratados adequadamente na próxima *Sprint Planning Meeting*. Todos os progressos propostos na *Sprint* que forem alcançados serão avaliados, posteriormente, na *Sprint Review Meeting*. Ao final da *Sprint Planning Meeting*, toda o *Scrum Team* se reúne, de maneira isolada, para tratar os assuntos que foram abordados e definir quanto cada um pode contribuir na *Sprint* que se inicia.

Sprint Review Meeting: é uma reunião que acontece quando uma *Sprint* é encerrada. Nesta oportunidade, o time de desenvolvimento apresenta o que foi conseguido durante a *Sprint* e, normalmente, é apresentada uma versão de demonstração das funcionalidades que foram implementadas neste ciclo. Os membros que participam da *Sprint Review Meeting*, tipicamente, são: *product owner*, *development team*, *scrum master*, clientes e alta gerência da organização solicitante. Durante essa reunião, o projeto é avaliado quanto ao cumprimento dos objetivos que foram estabelecidos durante a *Sprint Planning Meeting*.

Sprint Retrospective: ocorre ao final de uma *Sprint* e tem por objetivo permitir que todos os envolvidos identifiquem o que funcionou, o que deve ser melhorado e quais as ações a serem tomadas para alavancar os próximos passos no processo.

Artefatos

Os principais artefatos utilizados na metodologia *Scrum* serão descritos a seguir:

Product Backlog: é uma lista contendo todas as funcionalidades desejadas para o produto alvo do projeto. É escrita pelo *Product Owner* e não precisa estar completa desde o início do projeto. É possível iniciar com todos os requisitos óbvios e com o passar do tempo ir incluindo novas funcionalidades, na proporção em que a equipe e os futuros usuários vão aperfeiçoando os seus conhecimentos sobre o projeto. Na *Sprint Planning Meeting*, o *Product Owner* elenca as prioridades entre os itens do *Product Backlog* e os descreve para a equipe. A equipe então determina quais os itens que serão capazes de completar durante a *Sprint* que está por começar. Tais itens são então transferidos do *Product Backlog* para o *Sprint Backlog*. Em seguida os itens são decompostos em uma ou mais tarefas do *Sprint Backlog* e isso ajuda a dividir o trabalho entre os membros da equipe. Tarefas técnicas ou atividades relacionadas com os requisitos exigidos no projeto podem fazer parte do *Product Backlog*.

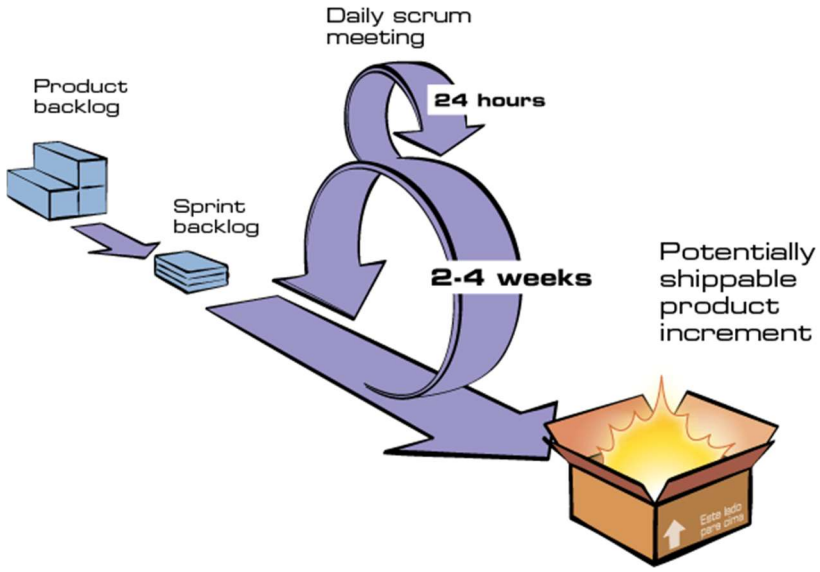
Sprint Backlog: é uma lista de tarefas que a equipe *Scrum* se compromete a realizar, em um ciclo *Sprint*. Todos os itens da *Sprint Backlog* são retirados do *Product Backlog*, pela equipe, seguindo sempre as prioridades estabelecidas pelo *Product Owner*. Também estão relacionadas nesta lista a estimativa de tempo que será necessário para completar cada funcionalidade exigida no projeto. Uma tarefa da equipe é determinar quantos itens do *Product Backlog* serão incorporados na *Sprint Backlog*. O *Scrum Master* mantém o *Sprint Backlog* atualizado para que reflita as tarefas que foram concluídas e qual a estimativa de tempo que a equipe determinou para completar as tarefas que ainda estão pendentes. Esta estimativa é calculada diariamente e colocada em um gráfico que resulta num *Sprint Burndown Chart*.

Release Burndown: durante o andamento de um projeto *Scrum*, ao final de cada *Sprint*, a equipe monitora o progresso das atividades, através da atualização de um *Release Burndown Chart*. No eixo horizontal são exibidos *Sprints* e no eixo vertical a quantidade de trabalho que ainda deve ser executado no início de cada *Sprint*. As atividades que ainda estão pendentes podem ser exibidas nas unidades preferenciais da equipe *Scrum*: *story points* ou *team days*, por exemplo.

No ciclo do processo *Scrum* (figura 02), o *Product Owner* constrói o *Backlog* (requisitos do projeto), que são listados por prioridade. A equipe que planeja a *Sprint* define alguns itens

prioritários e estabelece um prazo para cada um deles. Reuniões diárias permitem ao *Scrum Master* acompanhar todo o progresso e dão subsídios para que ele oriente toda a equipe, com foco principal na conclusão da *Sprint*. É feita então uma revisão detalhada da *Sprint* finalizada que irá preparar a equipe para o trabalho na próxima *Sprint*. Este ciclo continua até que a equipe consiga concluir tudo aquilo que esteja pendente, se esgote o orçamento ou acabe o prazo pré-determinado para a conclusão do projeto.

Figura 02 – Ciclo do processo Scrum.



Fonte: http://www.desenvolvimentoagil.com.br/scrum

Com uma breve descrição dos conceitos, dos papeis, das etapas e dos artefatos *Scrum*, abordados nessa seção, se pôde ter uma ideia das características e capacidade desse importante método ágil de gerência de projetos.

2.4 Personal software process

O PSP foi criado em 1989, por Watts Humphrey, pesquisador do SEI. Porém, foi apresentado, detalhadamente, no livro *A Discipline for Software Engineering* (HUMPRHREY, 1995). Dois anos depois, uma versão simplificada foi apresentada no livro *Introduction to the Personal Software Process* (HUMPRHREY, 1997). Em seguida, foi criado um relatório técnico (HUMPRHREY, 2000), fornecendo uma visão geral do processo. Outro livro, mais recente, que merece destaque é *A Self-Improvement Process for Software Engineers* (HUMPRHREY,

2005), que apresenta práticas industriais mais modernas, com gerenciamento de defeitos mais preciso.

Segundo a definição de seu próprio criador, Humprhrey, (1995), o *Personal Software Process* é um processo de auto-melhoria, projetado para ajudar os desenvolvedores a controlar, administrar e aperfeiçoar sua competência para produzir *software* de qualidade. Também pode ser visto como uma estrutura composta por um conjunto de *scripts*, formulários, *logs*, padrões e procedimentos para o desenvolvimento de *softwares* de qualidade.

O propósito do PSP é ajudar o desenvolvedor a realizar um trabalho seguindo padrões de qualidade e a ter maior previsibilidade e produtividade, de acordo com os conceitos clássicos de qualidade. Entre os mais importantes conceitos, encontra-se o de garantia de qualidade, como oposto ao controle de qualidade. Enquanto o segundo procura encontrar defeitos no produto acabado, o primeiro procura garantir que, em cada etapa da fabricação do produto, defeitos não sejam injetados.

Para o PSP, a competência de uma organização em desenvolver sistemas de acordo com as necessidades acordadas no início do projeto está diretamente ligada às habilidades individuais dos desenvolvedores em construir um produto de qualidade. Segundo SEI (2000), o PSP geralmente é aplicado individualmente, em uma equipe com poucos componentes.

O PSP se baseia no princípio de melhorias contínuas no processo individual de desenvolvimento de *software*, com foco na minimização da frequência de erros cometidos individualmente (HUMPHREY, 1995).

2.4.1 Estrutura do PSP

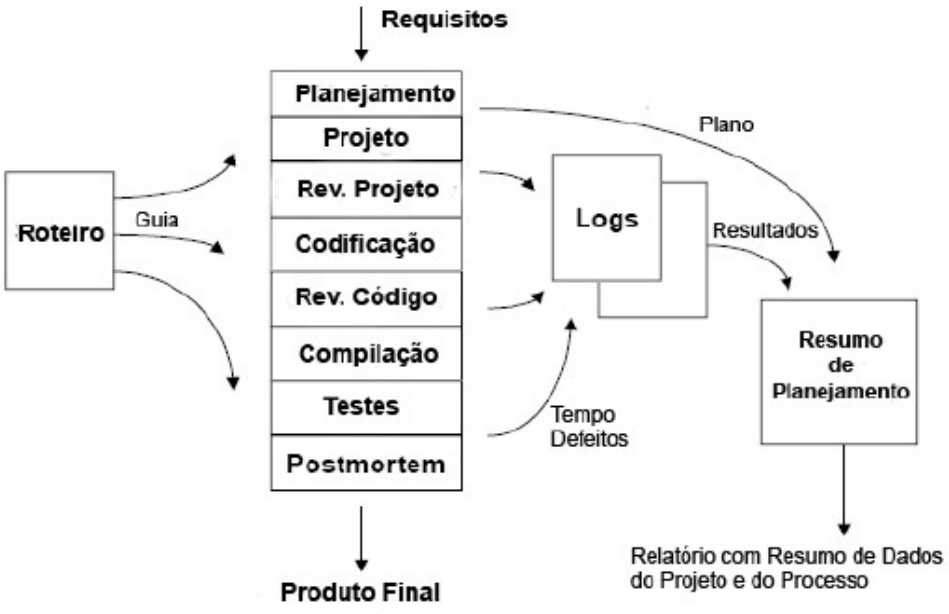
O PSP é um processo que se divide, basicamente, em oito fases: planejamento, projeto, revisão do projeto, codificação, revisão da codificação, compilação, testes e *postmortem*. Durante todas essas fases são utilizados quatro elementos básicos (POMEROY-HUFF et al., 2009):

1. Scripts: são descrições que guiam os usuários na execução de um processo. Documentam o objetivo do processo, os critérios de entrada, as diretrizes gerais, os passos a serem executados, as medições e os critérios de qualidade do processo e suas condições de saída.

2. Formulários: são a forma consistente de coletar e armazenar informações necessárias para execução dos processos. Planos resumidos e *checklists* de revisão são exemplos desses formulários.
3. Medições: são as formas de quantificar o processo e o produto e habilitar o engenheiro de *software* a manter um perfil de dados, com base em seus projetos anteriores, os quais podem ser usados em outros projetos. Além disso, as medições possibilitam monitorar a produtividade pessoal.
4. Padrões: são as definições que guiam o trabalho e a coleta e uso dos dados, tornando possível aplicar as mesmas medições em vários projetos para obtenção de dados que reflitam corretamente a realidade e auxiliem na melhoria do processo de desenvolvimento.

Com base em um roteiro de guias, a estrutura do processo PSP (figura 03) se inicia com a declaração dos Requisitos, tendo como primeiro passo o Planejamento. Existem roteiros (*scripts*) que guiam o trabalho e um formulário de totais para armazenar os dados, além dos log de registros de tempo e defeitos e o conjunto de formulários.

Figura 03 – Estrutura do PSP.



Fonte: The Personal Software Process (HUMPRHREY, 2000).

Basicamente, a fase de Planejamento é baseada no desempenho prévio individual de cada engenheiro, considerando sua participação (e registros) dos últimos projetos. Dessa forma é possível planejar cada atividade e controlar o próprio trabalho.

A fase de Projeto é direcionada à modelagem do programa, prevendo sua estrutura e funcionamento antes de começar a escrever o código. Existem formulários *templates* que auxiliam essas tarefas. Na próxima fase é realizada a Revisão do Projeto, onde a modelagem do sistema é revista, seguindo *checklists*.

Na fase de Codificação, o programador deve transformar o projeto revisado em um módulo ou parte do sistema, utilizando uma linguagem de programação consistente, de preferência seguindo padrões de codificação consolidados. A próxima fase consiste em revisar a codificação, seguindo, da mesma forma que a revisão de projeto, uma lista de verificação pré definida. O PSP concentra, claramente, mais esforços nas fases de revisões no que nas fases que as antecedem, pois são tentativas eficientes de diminuir a ocorrência de erros nas fases de compilação e testes.

A fase de Compilação (ou interpretação, dependendo da linguagem de programação) destina-se a executar o programa até que não existam mais mensagens de erro. Finalmente, a fase de Testes é executada e documentada até ter certeza que o projeto atende a todos os requisitos e projeto, além de não conter erros.

Ao longo do projeto de *software*, cada desenvolvedor deve registrar dados de tempo e defeitos. Esses dados são totalizados na fase de *Postmortem*, onde o tamanho do *software* é medido (e comparado com o planejado), o formulário de resumo de projeto completado e o produto final entregue (HUMPRHREY, 2000).

A implementação completa do PSP também pode ser estruturada em quatro níveis de competência. No nível de Medição Pessoal (PSP0 e PSP0.1) são registrados o tempo gasto em cada etapa do ciclo do desenvolvimento e os defeitos encontrados. O nível de Planejamento Pessoal (PSP1 e PSP1.1) é destinado a estimativas do tempo necessário para realizar uma tarefa com base em medições de tarefas semelhantes já realizadas. No nível de Qualidade Pessoal (PSP2 e PSP2.1) os erros do projeto são tratados a partir dos dados já coletados. A última etapa do PSP (PSP3), Processo Cíclico Pessoal é a extensão do modelo para projetos maiores.

2.4.2 Materiais de referência do PSP

Os materiais que compõem o PSP (quadro 01), separados pelo nível do processo (PSP 0 até o PSP 2.1), englobam roteiros (*scripts*), resumos do plano, formulários, modelos (*templates*), padrões e instruções de uso.

Quadro 01 – Materiais do PSP.

Heading						
Process Version	PSP0	PSP0.1	PSP1	PSP1.1	PSP2	PSP2.1
Process Scripts and Summaries						
PSP Process Script	X	X	X	X	X	X
PSP Planning Script	X	X	X	X	X	X
PSP Development Script	X	X	X	X	X	X
PSP Design Review Script					X	X
PSP Code Review Script					X	X
PSP Postmortem Script	X	X	X	X	X	X
Project Plan Summary and Instructions	X	X	X	X	X	X
PROBE Estimating Script			X	X	X	X
Forms, Templates, Standards, and Instructions						
Time Recording Log	X	X	X	X	X	X
Defect Recording Log	X	X	X	X	X	X
Defect Type Standard	X	X	X	X	X	X
PIP		X	X	X	X	X
Coding Standard		X	X	X	X	X
Test Report Template			X	X	X	X
Size Estimating Template			X	X	X	X
Task Planning Template				X	X	X
Schedule Planning Template				X	X	X
Design Review Checklist					58	69
Code Review Checklist					X	X
Use Case Specification Template						X
Functional Specification Template						X
State Specification Template						X
Logic Specification Template						X
Expanded Defect Type Standard						
Process Development Process Script (PDP)						
PSP3.0 Development Script						
Prototype Experimental Process Script (PEP)						
Product Maintenance Process Script (PMP)						

Fonte: PSP Material SEI, 2006.

Analisando as informações do quadro 01, foi possível constar que:

• Todos os níveis possuem *scripts* de Processo, Planejamento, Desenvolvimento (projeto, codificação, compilação e testes) e *Postmortem*, além de um Plano de Resumo de Projeto.

• Os níveis PSP2 e PSP2.1 possuem também *scripts* de Revisão de Projeto e de Revisão de Código.

• A partir do nível PSP1, todos possuem *script* de Estimativa de Tamanho PROBE (*Proxy-Based Estimating*), para melhor planejar seu trabalho.

• Todos os níveis possuem *log* de Registro de Tempo, Registro de Defeitos e um padrão para Tipos de Defeitos (também existe um padrão de tipos de defeitos estendido).

• A partir do nível PSP0.1, todos os níveis podem utilizar o formulário de Proposta de Melhoria de Processo (PIP) e Padrão de Codificação.

• A partir do nível PSP1, todos os níveis possuem *Templates* de Relatório de Teste e Estimativa de Tamanho.

• A partir do nível PSP1.1, todos os níveis possuem *Templates* para Planejamento de Tarefa e Planejamento de Cronograma.

• Os níveis PSP2 e PSP2.1 possuem *Checklist* de Revisão de Código e *Checklist* de Revisão de Código.

• O último nível, PSP2.1 possui *Templates* para Especificação Operacional, Especificação Funcional, Especificação de Estado e Especificação Lógica, para registrar o projeto do programa.

O SEI ministra treinamentos sobre o PSP, ofertados na forma de curso fundamental e curso avançado. Também existem formações para instrutor PSP. Em seu site (https://www.sei.cmu.edu/training/p19b.cfm), o SEI oferece materiais de auto-estudo para *download*, complementares ao livro *A Self-Improvement Process for Software Engineers*, em versões tanto para alunos quanto para educadores.

2.4.3 Níveis do PSP

Nesta seção serão abordados, detalhadamente, todos os níveis do modelo de qualidade PSP, exceto o último, PSP3, que tem o foco nas equipes e não no indivíduo, estendendo o PSP para projetos maiores.

2.4.4 PSP de nível 0

O Processo de Medição Pessoal, composto pelos níveis PSP0 e PSP0.1, define as medidas e formatos de relatórios, fornecendo a base sobre a qual será implantada a melhoria contínua pessoal. Os objetivos do nível 0 do PSP são (SEI, 2006):

- Demonstrar a utilização de um processo definido para escrever pequenos programas;
- Incorporar medidas básicas no processo de desenvolvimento de *software*;
- Exigir mudanças mínimas nas práticas pessoais do desenvolvedor.

O PSP0 possui um *script* para guiar o processo de desenvolvimento de pequenos programas ou pequenos módulos de *software*. Esse nível procura propor poucas mudanças no atual processo de *software* do programador, fazendo com que este siga as fases do modelo PSP, que contempla o planejamento, desenvolvimento e *postmortem*. Essas fases também possuem *scripts* próprios para guiar suas atividades.

A fase de planejamento necessita da descrição do problema do programa, além de artefatos como o formulário de resumo de plano de projeto e *log* de registro de tempo. É composta por duas atividades: especificação dos requisitos do programa e estimativa de recursos. Para contemplar a primeira atividade o programador deve obter (ou produzir) a especificação de requisitos do programa. A segunda atividade representa a estimativa de tempo, em minutos, que o programador deve levar para realizar o seu trabalho. Na fase de desenvolvimento são utilizados dois formulários: o *log* de registro de tempo (*Time Recording Log*) e o *log* de registro de defeitos (*Defect Recording Log*).

O formulário *Time Recording Log* registra o tempo gasto pelo programador em cada fase do PSP (planejamento, projeto, revisão do projeto, codificação, revisão da codificação, compilação, testes e *postmortem*). Isso inclui o tempo, em minutos, de início e fim da tarefa, bem como o tempo de interrupção nessas atividades. Estes formulários serão melhor explicados quando o modelo integrado SCRUM-PSP-IF for descrito, no terceiro capítulo desse trabalho.

O formulário *Defect Recording Log* registra dados sobre os defeitos e o tempo para sua remoção durante a execução de cada fase do PSP. Para isso, o programador deve registrar o nome do projeto, a data em que o defeito foi encontrado, um número identificador único para

o defeito, um tipo (com base em uma tabela de categorias de defeitos), a fase em que o defeito foi injetado, a fase em que o defeito foi removido e o tempo para a remoção.

O padrão de tipos de defeitos contém um conjunto de dez tipos de defeitos que podem ser encontrados nas atividades do PSP (ANEXO A). Também existe um conjunto estendido de padrões de defeitos.

Na fase de *postmortem*, os dados coletados através dos formulários *Time Recording Log* e *Defect Recording Log* devem ser registrados no *Project Plan Summary* do PSP0 (estes formulários serão exibidos quando o modelo integrado SCRUM-PSP-IF for descrito, no terceiro capítulo desse trabalho).

Com os registros contidos no Resumo pode ser realizada a comparação entre o tempo planejado e realmente utilizado, considerando o tempo gasto em cada fase do PSP. Para isso existem os registros do tempo real gasto em um determinado programa, os registros do tempo acumulados nas fases, considerando todos os programas desenvolvidos utilizando o PSP, e a distribuição percentual do tempo.

Com esses registros também se pode diagnosticar quais fases do processo estão demandando mais tempo e inserindo mais defeitos, possibilitando melhores estimativas e ações específicas para a melhoria contínua do processo.

A partir do conteúdo apresentado nesta seção, podemos identificar que os principais artefatos do PSP 0 são:

- Quatro *scripts*: um para orientar os desenvolvedores na utilização do PSP 0 (*script* de processo) e outros três para orientar em cada fase do PSP - Planejamento, Desenvolvimento e *Postmortem*;

- Três formulários: um de resumo de plano de projeto, outro com o *log* de registro de tempo e outro com o *log* de registro de defeitos;

- Dois tipos de Medições e um Padrão: medição de tempo e de defeitos e um padrão de tipo de defeitos.

2.4.5 PSP de nível 0.1

O PSP0.1 objetiva definir, de forma consistente, como aferir o tamanho do *software* ao término de seu desenvolvimento. Portanto, uma estimativa deve ser feita com base no tamanho esperado do programa para, após o desenvolvimento, medir o tamanho real do programa. O PSP adotou como padrão para medir o tamanho dos programas o número de Linhas de Código (LOC- *Lines of Code*). Nesse nível, também são utilizados os *scripts* Processo, Planejamento, Desenvolvimento e *Postmortem*.

O PSP 0.1 tem os seguintes objetivos (HUMPHREY, 2005 e SEI, 2006):

- Incentivar a criação e uso de padrões de codificação;
- Medir e contabilizar, com exatidão, o tamanho dos programas produzidos;
- Incentivar a melhoria dos processos para o desenvolvimento de *software*.

No nível 0.1 existem dois novos elementos do processo, o formulário de Proposta de Melhoria de Processo (PIP - *Process Improvement Proposal*) e os Padrões de codificação e contagem de tamanho. O reflexo desses novos elementos se dá apenas no formulário de resumo do plano de projeto, que foi expandido com a seção Resumo do Tamanho do Programa. Ao se registrar o tamanho dos programas, é possível criar um histórico de tamanho dos programas desenvolvidos. As fases de planejamento e desenvolvimento não sofreram alterações.

O PIP é um formulário de registro de problemas ou sugestões de melhoria durante o processo. Embora seja possível lembrar-se posteriormente de algum passo ou tarefa que foi um problema, provavelmente os detalhes serão esquecidos se não forem registrados prontamente. O formulário também pode ser usado para registrar comentários, ideias, melhorias ou notas sobre o que foi aprendido em cada programa, como mostra o quadro 02.

Quadro 02 – Process Improvement Proposal (PIP).

Desenvolvedor	Desenvolvedor 1	Data	05/01/2016
Programa	Programa 1	Programa #	1
Instrutor	Norton Jerzewski Noro	Linguagem	PHP

Descrição do Problema
Descrever brevemente os problemas encontrados.
Erros de digitação dos dados de defeitos.
Descrição da Proposta
Descrever brevemente as melhorias de processo propostas.
Melhorar a legibilidade do código, utilizar comentários mais descritivos.
Outras Notas e Comentários
Quaisquer notas, outros comentários ou observações que descrevem as experiências ou ideias de melhoria.
Utilizar o PIP.

Fonte: HUYMPHREY, 2005.

Para medir o tamanho do programa, em LOC, o PSP0.1 indica, primeiramente, a utilização de um padrão de codificação. Os padrões existem para que, além de correto, o código fonte seja legível e bem estruturado, para auxiliar também os testes, modificações e reuso. Comentários claros e precisos ajudam outras pessoas que atuam no desenvolvimento do sistema.

O padrão de codificação adotado deve ser seguido em todos os programas desenvolvidos na mesma linguagem de programação. Existem padrões de codificação para Java (ALBUQUERQUE, MEIRA, 1997) e C++ (HUMPHREY, 1995), por exemplo.

Após definir e utilizar o padrão de codificação, a contagem e classificação das linhas de código produzidas deve ser realizada, ao final do desenvolvimento do programa. As categorias são (HUMPHREY, 2005): base (B), adicionadas (A), modificadas (M), deletadas (D), reutilizadas (R), adicionadas e modificadas (A&M) e novas reutilizadas (NR). Após a adequada classificação das linhas, pode-se calcular o total de linhas de código do programa construído: $T = B + A + R – D$.

Os resultados das contagens de LOC possibilitam aos engenheiros de *software* estimar com mais precisão o tamanho dos próximos programas, bem como obter seu próprio desempenho, em um determinado período de tempo, além de fornecer dados para que as métricas de qualidade de um programa sejam aplicadas. Por exemplo, o número de defeitos para um determinado número de linhas ou o número de defeitos em uma determinada fase do projeto.

Na fase de *postmortem*, as medições (base, adicionadas, modificadas, deletadas, reutilizadas, adicionadas e modificadas e novas reutilizadas) devem ser registradas no Resumo do Plano de Projeto, na seção Resumo para tamanho de programa (acrescido no PSP0.1), bem como os dados devem ser coletados através dos formulários *Time Recording Log* e *Defect Recording Log*.

A partir do conteúdo apresentado nesta seção, podemos identificar que os principais artefatos do PSP 0.1 são:

• Quatro *scripts*: um para guiar o processo do PSP 0.1 e os outros três para guiar no planejamento, desenvolvimento e *postmortem*;

• Dois formulários: um Resumo do Plano de Projeto e um *Process Improvement Proposal* (PIP);

• Um tipo de Medição e um tipo de Padrão: uma medição do tamanho dos programas desenvolvidos e um padrão de codificação.

2.4.6 PSP de nível 1

O Processo de Planejamento Pessoal engloba os níveis 1 e 1.1 do PSP. Além dos objetivos do nível anterior, o PSP1 tem como objetivo (SEI, 2006):

- Estabelecer um procedimento ordenado e repetível para o desenvolvimento de estimativas de tamanho de *software* e para o registro de informações sobre os testes de *software* realizados.

Nesse nível de processo, existem três novos elementos: o PROBE (*Proxy-Based Estimating*), o *template* de Estimativa de Tamanho e o *template* de Relatório de Teste. O Resumo do Plano de Projeto do PSP1 foi expandido, na seção de Resumo do tamanho do programa, para registrar o tamanho planejado para todas as categorias de tamanho, incluindo linhas de código novas, reutilizadas, deletadas ou alteradas. Os formulários serão exibidos na apresentação do modelo SCRUM-PSP-IF, no capítulo três desse trabalho.

PROBE é o método de estimativa de tamanho utilizado pelo PSP1, principalmente na fase de planejamento. Esse método é baseado em técnicas estatísticas e seu critério de medida é o *Proxy*, que pode ser definido como uma unidade de medida que relaciona o tamanho do

produto com funções que podem ser facilmente visualizadas. São exemplos: classes, objetos, tabelas, campos ou telas.

Antes de realmente utilizar o PROBE, se faz necessário criar uma tabela de tamanho relativo para definir o número médio de LOCs para os *proxies* nas cinco faixas de tamanho: muito grande (VL), grande (L), médio (M), pequeno (S), e muito pequeno (VS), além de elencar as categorias. Não são necessárias mais que cinco ou seis categorias para definir o *proxy* (HUMPHREY, 2005).

No quadro 03 pode ser visto um exemplo de tabela de tamanho relativo, usando classe como *proxi*, na linguagem C++. Os cálculos detalhados podem ser vistos no livro *Self-Improvement Process for Software Engineers* (HUMPRHREY, 2005).

Quadro 03 – Exemplo de Tabela de Tamanho Relativo.

Tamanho de classe em LOCs - Linguagem C++					
Categoria	Muito pequena	Pequena	Média	Grande	Muito grande
Cálculo	2,34	5,13	11,25	24,66	54,04
Dados	2,60	4,79	8,84	16,31	30,09
Entrada/Saída	9,01	12,06	16,15	21,62	28,93
Lógica	7,55	10,98	15,98	23,25	33,83
Configuração	3,88	5,04	6,56	8,53	11,09
Texto	3,75	8,00	17,07	36,41	77,66

Fonte: Extraído de HUMPHREY (2005).

As LOCs do programa devem ser registradas, por categoria, no *Template* de Estimativa de tamanho (*Size Estimating Template*). Neste formulário, devem ser estimadas as partes base, adicionadas e reutilizadas. LOC base são aquelas que já existiam antes da construção do programa. LOC adicionadas são as criadas no decorrer da construção do programa. LOC reutilizadas são aquelas retiradas integralmente da biblioteca de reuso. Além dos registros de LOC estimadas, devem-se registrar os dados reais sobre o tamanho do programa.

Além do método PROBE e de seu formulário de estimativa de tamanho, o nível 1 do PSP indica também o uso do *Template* de Relatório de Teste (*Test Report Template*) para manter um registro detalhados sobre os dados dos testes realizados nos programas, como por exemplo, descrição, condições de teste, resultados esperados e resultados reais. Essas e outras informações possibilitam a repetição desses testes, alcançando os mesmos resultados.

Assim como nos níveis 0 e 0.1 do PSP, o PSP 1 estabelece o Resumo do Plano de Projeto do PSP 1, o qual acrescenta a produtividade real, planejada e o histórico de produtividade, no formato de LOC/hora. Também expande a seção de resumo de tamanho do programa para

representar os dados planejados sobre o tamanho em todas as categorias de LOC. Todos os valores, exceto o valor total real do tamanho do programa devem ser calculados.

A partir do conteúdo apresentado nesta seção, podemos identificar que os principais artefatos do PSP 1 são:

• Cinco *scripts*: um para guiar o processo do PSP 1, outros três para guiar no planejamento, desenvolvimento e *postmortem* e o *script* de estimativa com o método PROBE;

• Três formulários: um formulário de Resumo do Plano de Projeto, outro de Estimativa de Tamanho (*Size Estimating Template*) e um de Relatório de Teste (*Test Report Template*).

2.4.7 PSP de nível 1.1

O PSP de nível 1.1 incorpora atividades às fases de planejamento e desenvolvimento, não agregando nenhuma mudança à fase de *postmortem*. Mantendo os objetivos do PSP1, o PSP1.1 tem como objetivos específicos introduzir métodos para:

- Criar o planejamento de recursos, tarefas e cronograma e acompanhar esses planos;
- Acompanhar o progresso do que foi planejado com relação ao realizado.

Existem dois novos elementos do processo, o *template* de planejamento de tarefas e o *template* de planejamento de cronograma, geralmente utilizados em projetos com várias semanas de duração. Nesse contexto, o resumo do plano de projeto deve ser expandido para incluir as estatísticas básicas do processo. (HUMPHREY, 2005)

O plano de tarefas visa estimar o tempo de desenvolvimento e a data de conclusão de cada tarefa, proporcionando uma base para rastrear o progresso do cronograma do programa. Para criar o planejamento de tarefas, o programador deve, inicialmente, utilizar o método PROBE para estimar o tempo total de cada fase do PSP para desenvolver os programas do projeto (com base no campo "*To Date %*"). No formulário de tarefas cada atividade deve ser estimada separadamente, mesmo sendo pertencente à mesma fase (respeitando o total de atividades na fase).

As informações citadas anteriormente, bem como os dados reais das tarefas, devem ser registradas no *Task Planning Template*, considerando dados sobre o nome da tarefa, fase do PSP a que ela pertence, duração planejada (em horas), total de horas acumuladas nas tarefas anteriores até a última tarefa, semana em que se estima concluir a tarefa e semana em que a tarefa foi realmente concluída, valor planejado (VP) para cada tarefa, VP acumulado (soma dos VP anteriores), horas gastas para cada tarefa, valor agregado (equivale ao VP se a tarefa for concluída com êxito) e valor agregado acumulado (soma dos anteriores).

O plano de cronograma tem o objetivo de registrar as horas reais e estimadas despendidas nas tarefas para um determinado período de calendário (geralmente medido em semanas), conforme disponibilidade do programador, considerando a quantidade de horas total do projeto.

Os seguintes atributos são utilizados e devem ser registrados no *Schedule Planning Template*: identificador numérico da semana (se o projeto for muito pequeno pode-se usar dias), data de início da semana, horas disponíveis para o trabalho na semana (considerando dias não trabalhados também, como feriados e reuniões), horas acumuladas até determinada semana, horas reais gastas em cada semana, horas reais acumuladas até determinada semana, PV e EV de cada semana (somatório dos PVs e EVs das tarefas que deverão ser concluídas em cada semana), PV e EV acumulado até determinada semana.

Com os planos de tarefas e cronograma, cada desenvolvedor tem condições de planejar suas atividades, de acordo com as semanas disponíveis para a execução do projeto. Além disso, pode-se acompanhar e comparar o progresso real com o planejado (EV acumulado comparado com PV acumulado). Se o EV acumulado for maior ou igual ao PV acumulado, o progresso do projeto está dentro do esperado. Caso contrário, o planejamento das tarefas e cronograma deve ser revisado.

Assim como nos níveis anteriores do PSP, o nível 1.1 possui seu próprio Resumo do Plano de Projeto do PSP 1.1, o qual foi expandido para acrescentar as seguintes informações:

- Tempo planejado, real e acumulado para a construção do programa;
- *Cost Performance Index* consiste na relação: tempo planejado *To-Date* / tempo real de trabalho *To-Date*;

- %Reutilizada (%*Reused*) se refere à porcentagem do tamanho reutilizado para determinado projeto, representado pela fórmula: LOC Reutilizadas / Total LOC * 100.

- %Novo reutilizável (%*New Reusable*) representa a percentagem de código novo reutilizável desenvolvido para determinado projeto, calculado pela fórmula: Novas reutilizáveis / adicionadas e modificadas * 100.

Em resumo, a partir do conteúdo apresentado podemos identificar que os principais que os elementos do PSP 1.1 são:

• Quatro *scripts*: um para guiar os desenvolvedores na utilização do PSP1.1 e três para guiá-los nas fases de planejamento, desenvolvimento e *postmortem* do PSP;

• Três formulários: Sumário do Plano de Projeto, *Task Planning Template* e *Schedule Planning Template*.

2.4.8 PSP de nível 2

Os níveis 2 e 2.1 do PSP representam os processo de Gestão de Qualidade Pessoal. Além dos objetivos vindos do nível anterior, os objetivos específicos do PSP2 são (SEI, 2006):

- Introduzir revisões de projeto e de código;

- Introduzir métodos para avaliar e melhorar a qualidade das revisões, adicionando, assim, a capacidade de planejamento da qualidade;

- Estimar o número de defeitos nas fases.

Nesse nível, existem dois novos elementos do processo: a *Checklist* de Revisão de Projeto e a *Checklist* de Revisão de Código e com isso, as capacidades de revisar projeto e código e de realizar o planejamento da qualidade do programa. O Resumo do plano de projeto foi expandido para suportar estes novos recursos, através de registros de dados de tempo e defeito (injetados e removidos) sobre as fases de Revisão de Projeto (DLDR) e Revisão de Código (CR).

O PSP2 introduz o planejamento da qualidade, englobando estimativas do número total de defeitos injetados e removidos em cada fase do processo, estimativas da quantidade de tempo

necessário para as revisões de projeto e de código e ajustes desses parâmetros para garantir um resultado de maior qualidade.

Ambas as revisões possuem *script* para orientação do trabalho, que busca encontrar e corrigir defeitos nas fases de projeto e código, antes da compilação e dos testes (unitário, integração e sistema), pois é menos custoso e mais rápido corrigir os defeitos nas fases iniciais do desenvolvimento de *software* do que nas finais (PRESSMAN, 2006). Enquanto a *checklist* de revisão do projeto é específica, a *checklist* de revisão do código varia de acordo com a linguagem de programação e o seu padrão de codificação. Obviamente, a revisão de projeto deve ser realizada logo após a criação do projeto e antes da fase de codificação, enquanto a revisão do código deve ser realizada após a codificação e antes da compilação.

As *checklists* não são necessárias apenas para remover defeitos, mas servem também como lembretes das boas práticas de desenvolvimento que devem ser seguidas e aplicadas pelos engenheiros de *software*.

Inicialmente, nas fases de revisão, não estão disponíveis dados históricos para o planejamento de defeitos injetados e removidos. Até que se tenham esses dados, o número de defeitos injetados é zero (0).

Com relação ao planejamento de qualidade, o nível 2 do PSP possui muitas formas de medir e avaliar a qualidade. Este planejamento envolve estimativa do número total dos defeitos injetados, estimativa do número de defeitos injetados e removidos em cada fase e estimativas da quantidade de tempo necessário para as revisões de projeto e de código.

Além das estimativas citadas anteriormente, o PSP2 fornece outras medidas de qualidade, como a eficiência de remoção de defeitos (*defect removal efficiency*), *defect removal leverage*, defeitos de teste por KLOC (*test defects per KLOC*), total de defeitos por KLOC (*total defects per KLOC*) e rendimento (*yield*) de fase ou total.

Eficiência de remoção de defeitos mostra o número de defeitos por hora removidos durante a revisão do projeto, revisão de código, compilação e teste. É expressa pela fórmula:

Eficiência de remoção de defeitos = 60 x defeitos removidos na fase / tempo na fase (minutos)

Defect removal leverage compara a eficiência na remoção de defeitos da fase de teste unitário com outras três fases, revisão de projeto, revisão de código e compilação. É expressa pela fórmula:

Defect removal leverage = defeitos removidos por hora para revisão do projeto ou revisão do código ou compilação / defeitos removidos por hora para teste unitário

Defeito de teste por KLOC indica a qualidade do programa que está sendo testado, sendo medido pela fórmula:

Defeitos Teste/KLOC = 1000 x defeitos removidos nos testes / total LOC adicionadas e modificadas.

Total de defeitos por KLOC representa a medida do total de defeitos injetados durante o processo.

Total defeitos/KLOC = 1000 x total de defeitos removidos / total LOC adicionadas e modificadas

Rendimento total: representa a porcentagem de defeitos injetados e removidos antes da primeira compilação. É expressa pela fórmula:

Yield (total) = 100 x defeitos removidos antes da compilação / defeitos injetados antes da compilação

Rendimento por fase: representa a porcentagem de defeitos removidos e não removidos em determinada fase. É expressa pela fórmula:

Yield (fase N) = 100 x defeitos removidos na fase / defeitos não removidos na fase

Outras medidas, como o Process Quality Index (PQI), serão abordadas, detalhadamente, no capítulo três e quatro dessa dissertação.

Como nos demais níveis do PSP, o PSP 2 possui seu próprio Resumo do Plano de Projeto do PSP 2, o qual acrescenta ao PSP 1.1:

- Defeitos injetados e removidos nas fases de revisão de projeto e revisão do código, bem como do tempo, em minutos, nas fases;
- Defeitos por KLOC por fase e total (planejado, real e até a data);

- Rendimento do processo;
- Dados sobre a eficiência na remoção de defeitos.

Em resumo, a partir do conteúdo apresentado, podemos identificar que os principais elementos do PSP 2 são:

- Seis *scripts*: um para guiar os desenvolvedores na utilização do PSP2, três para guiar as fases do PSP (planejamento, desenvolvimento e *postmortem*), um para orientar as revisões de projeto e outro para orientar as revisões de código;
- Três formulários: um Resumo do Plano de Projeto, uma *Checklist* para revisões de projeto e um *Checklist* para revisões de código.

2.4.9 PSP de nível 2.1

O foco do PSP2.1 está na qualidade de projeto do PSP, incluindo padrões para essa fase. Pode ser considerado o último nível pessoal do modelo, pois PSP3 é um nível legado que foi substituído pelo TSP (KHAN, 2012) e por isso não será abordado nesse trabalho. O nível 2 do PSP tem como objetivos específicos (SEI, 2006):

- Reduzir o número de defeitos nos projetos e oferecer critérios para determinar se um projeto está completo;
- Introduzir medidas adicionais para a gestão do processo de qualidade;
- Introduzir modelos de projeto que forneçam uma estrutura ordenada e um formato para gravação de dados de projetos.

O PSP2.1 acrescenta ao processo o *Script* e *Checklist* de Revisão de Projeto PSP2.1 mais detalhados e quatro Modelos (*templates*) de projetos. Os modelos de especificação de projeto do PSP ajudam a registrar e documentar o projeto, facilitando a revisão:

- *Template* de especificação operacional – mantém as descrições dos prováveis cenários operacionais seguidos durante o uso do programa, garantindo que todas as questões significativas de uso sejam consideradas durante o projeto do programa. Também auxilia na especificação dos cenários de teste. Este modelo pode ser representado pelo

diagrama de casos de uso da *Unified Modeling Language* (UML), pois este mostra cenários com as funcionalidades do sistema, na perspectiva dos usuários;

- *Template* de especificação funcional – mantém as especificações funcionais (*interfaces* e lógica dos métodos de cada classe, por exemplo) de uma parte do programa ou do programa inteiro. Pode ser representado pelo diagrama de classes da UML, que descreve os objetos do sistema e seus relacionamentos;
- *Template* de especificação de estados – mantém o estado e as especificações de transição de estado para um sistema, classe ou programa, dando suporte às análises de máquinas de estado durante o projeto, bem como nas revisões e inspeções de projeto. Esse modelo é similar ao diagrama de estados da UML, que mostra comportamento e estado dos objetos (e suas transições);
- *Template* de especificação lógica – mantém os pseudocódigos de um programa ou componente e, dessa forma, fornece subsídios para uma implementação mais precisa e completa.

No Resumo do plano de projeto foram adicionados os seguintes itens:

- Intervalo superior de previsão (UPI – *upper prediction interval*) e Intervalo inferior de previsão (LPI – *lower prediction interval*), valores calculados apenas se o desenvolvedor usar o método A ou B do PROBE;

- O item % *Appraisal Cost of Quality* se refere à percentagem de tempo de desenvolvimento gasto na revisão de projeto e revisão de código. Representado pela fórmula:

% *Appraisal COQ = 100 x tempo de revisão de projeto + tempo de revisão do código / tempo total de desenvolvimento.*

- O item % *Failure Cost of Quality* é a percentagem de tempo de desenvolvimento gasto em compilar e testar. Representado pela fórmula:

% *Failure COQ = 100 x tempo de compilação + tempo de teste / tempo total de desenvolvimento.*

- O item % *COQ A/F Ratio* representa a proporção de custos de avaliação para os custos de falhas. Representado pela fórmula:

% *COQ A/F Ratio = % Appraisal COQ / % Failure COQ*

A partir do conteúdo apresentado nessa seção pode-se concluir que os principais elementos do PSP 2.1 são:

- Cinco *scripts*: um para orientar os desenvolvedores na utilização do PSP 2.1, três para orientá-los nas fases de planejamento, desenvolvimento e *postmortem* e um para orientar as revisões de projeto.
- Seis formulários: um Sumário do Plano de Projeto do PSP2.1, uma *checklist* para revisão de projeto e quatro modelos de especificação de projeto.

2.5 Integração entre PSP e métodos ágeis de desenvolvimento de software

O cenário atual da indústria de *software*, considerando a concorrência mercadológica cada vez mais acirrada e o anseio por janelas de tempo cada vez menores para a entrega do produto faz com que a maioria das empresas desenvolvedoras tente encontrar maneiras de criar *software* com maior qualidade, em menor tempo e com menos custos. Dentre os fatores que influenciam no alcance desses objetivos, o Processo é um fator chave de sucesso, além das Pessoas envolvidas e das Ferramentas utilizadas. Infelizmente, ainda não há um processo ou um roteiro pré-definido que alcance integralmente esses objetivos, mas existem alguns esforços para atender às necessidades mais modernas de desenvolvimento e obter resultados mais satisfatórios, por meio da integração de processos de desenvolvimento de *software*.

Os estudos referenciados nessa seção mostram alguns indicativos de que PSP pode ser integrado a alguns métodos ágeis no desenvolvimento de *software*. Turk (2002) desenvolveu um trabalho identificando as limitações de muitos processos ágeis publicados, relacionado ao esforço dos desenvolvedores e gerentes para adequar os processos ágeis a seu ambiente de desenvolvimento.

Sobre integração de PSP com processos ágeis, algumas pesquisas tentam demonstrar a possibilidade do PSP ser integrado com outros modelos (SHEN et al., 2013): *Extreme Programming* (XP), *Feature Driven Development* (FDD), *Adaptive Software Development* (ASD), *Dynamic Systems Development Method* (DSDM), *Lean Software Development* (LSD), *Crystal*, *Evolutionary Project Management* (EVO) e *Scrum*. Porém.... somente x e y que apresentam...

Especificamente sobre a integração do PSP com *Scrum*, uma pesquisa se destaca na constatação que esses dois métodos podem complementar um ao outro, onde o primeiro atua sobre a melhoria da capacidade do processo do engenheiro de *software*, enquanto o segundo fornece um *framework* de gerenciamento de processo de *software* no nível de equipe, se concentrando na cooperação e adaptabilidade ao ambiente de projeto. Em comparação com *Scrum* tradicional, esse modelo integrado visa fornecer práticas mais concretas, para controlar e gerenciar a qualidade, em pequenos ou médios projetos (RONG et al., 2010).

A seguir serão apresentadas as características de integração entre PSP e alguns métodos ágeis, obtidas a partir de uma revisão sistemática da literatura (SHEN et al., 2013), que mostra as características dos processos nas tentativas de integração com o *PSP*: *XP*, *DSDM* e *Scrum*. Esse mesmo estudo não encontrou propostas que promovessem a integração do *PSP* com os modelos como *Crystal*, *EVO* e *ASD*.

2.5.1 PSP e XP

Extreme Programming (XP) pode ser considerado como um *framework* ágil de desenvolvimento de *software*, indicado para pequenas e médias equipes, onde os projetos se caracterizam por sofrer frequentes mudanças de requisitos e exigir alta qualidade do produto, enquanto o PSP ajuda cada desenvolvedor a compreender e melhorar o seu próprio desempenho, com registros de dados e utilização de padrões.

PSP tem fortes indicativos que pode ser integrado com o XP, pois:

1. Como o PSP é formado por uma estrutura de *scripts*, pode fornecer ao XP orientações mais precisas para detalhar o processo de desenvolvimento e suportar, com mais eficiência, as atividades do modelo XP. Porém, a escolha dos *scripts* e formulários deve ser realizada de acordo com as especificidades do projeto (MIHAYLOV, 2003).

2. PSP pode fornecer métodos, como o PROBE, para estimar e planejar o trabalho, acrescendo medidas que o XP não possui. As estimativas devem ser realizadas de acordo com o registro de dados dos últimos projetos (SVENSSON, 2005).

3. PSP pode oferecer ao XP formulários de registro de defeitos. Com a análise dos defeitos o programador tem condições de evitar a repetição do erro ou aproveitar a solução do mesmo defeito em outro projeto (SVENSSON, 2005), além de oferecer um formulário de registro de

tempo, que auxilia na correta estimativa do esforço necessário para cumprir as atividades do projeto.

4. O atributo de Valor Agregado do PSP pode ser usado para acompanhar o progresso do projeto XP, vendo se este está sendo realizado dentro do esperado (SVENSSON, 2005).

5. Como o PSP está estruturado em níveis de maturidade de processo, as organizações podem escolher um determinado nível para ser implantado, de acordo com a própria maturidade (MIHAYLOV, 2003).

Complementarmente, XP tem fortes indicativos que pode ser integrado com o PSP, pois:

1. Engenheiros de *software* XP podem utilizar a análise de diagramas UML de alto nível, como Casos de Uso, para auxiliar o PSP na obtenção de requisito de *software* (WILLIAMS, 2001), pois o modelo de especificação operacional do PSP2.1 registra informações similares às presentes nesse diagrama. Ainda, existem outros diagramas da UML que podem auxiliar, como o diagrama de classes e de estados, que podem ser utilizados para complementar os formulários *templates* de especificação funcional e especificação de estados, respectivamente.

2. Considerando que a programação em pares é uma revisão contínua, as revisões de projeto e código do PSP podem ser utilizadas individualmente e também em pares (WILLIAMS, 2001).

3. O XP possui práticas de desenvolvimento orientadas a testes, o que significa que testes podem ser realizados mesmo antes de começar a codificar o *software*. Se esta prática for utilizada juntamente com os testes do PSP, existe uma grande possibilidade de aumentar a qualidade do desenvolvimento e reduzir o tempo (DZHUROV, 2009; ILIEVA, STEFANOVA 2002), além de reduzir os defeitos no produto final.

4. A integração contínua do XP inclui práticas de controle de código e controle de versões, que pode acelerar o progresso do desenvolvimento do PSP (DZHUROV, 2009; ILIEVA, STEFANOVA 2002).

2.5.2 PSP e DSDM

A integração do PSP com *Dynamic Systems Development Method* (DSDM) tem o objetivo de desenvolver *software* em pequenas empresas (COLEMAN, VERBRUGGEN, 1998). DSDM é um método ágil iterativo e incremental, que enfatiza o envolvimento constante do usuário.

PSP pode ser integrado ao DSDM (COLEMAN, VERBRUGGEN, 1998):

1. Testes unitários e de integração do PSP podem ser integrados dentro de cada um dos ciclos de desenvolvimento DSDM.

2. O PSP sugere o uso de *proxies* como unidade de medida de tamanho (presente no método de estimativa de tamanho PROBE) e o DSDM promove o uso de *timeboxes*. Este mecanismo de coleta e registro de dados do PSP pode fazer com que as pessoas obtenham os dados históricos para determinar quantas LOC podem ser desenvolvidas dentro de um determinado *timebox*.

3. PSP pode sugerir que revisões sejam realizadas em cada uma das fases de prototipagem do DSDM ou que cada desenvolvedor realize revisão de projeto e código, enquanto outro realiza os testes. Adicionalmente, realizar os testes em pares pode aumentar a confiança dos desenvolvedores.

4. Algumas documentações fornecidas pelo PSP poderiam ser incluídas em um plano de qualidade DSDM, como por exemplo, o *script* de processo, a *checklist* de projeto, *checklist* de código e o formulário PIP, que fornece uma maneira de documentar sugestões de melhorias no processo.

5. DSDM não recomenda nenhuma abordagem específica para a coleta e análise de métricas de defeitos, produtividade, tamanho, tempo e cronograma. Porém, todas estas métricas são bem definidas no PSP.

2.5.3 PSP e Scrum

Scrum é composto de um *framework* de gestão de processos ágeis para equipes, incluindo as de *software*, enquanto PSP é uma metodologia que fornece as habilidades e disciplinas que devem ser usadas por cada programador para planejar, estimar e gerenciar o seu trabalho, com foco na redução de defeitos. A integração desses dois modelos pode satisfazer as necessidades de previsibilidade e de adaptação a um ambiente de desenvolvimento moderno de *software*.

PSP tem fortes indicativos que pode integrar *Scrum*, pois:

1. No PSP o trabalho pode ser dividido de forma que os dados históricos tenham um tamanho relativamente pequeno para que estimativas mais precisas possam ser feitas (RONG, 2010 e SANDE, 2011).

2. O plano de tarefas pode ser determinado pelos resultados das estimativas de tamanho e de recursos da equipe. Se o cronograma não coincidir com o prazo real do projeto, mudanças podem ocorrer para alterar a situação, realocando novos recursos (RONG, 2010; SUPHAK, WERASAK, 2011 e SANDE, 2011).

3. No plano de qualidade, alguns dos principais índices de qualidade do PSP são o *Yield* (rendimento), *review rate* (taxa de revisão), PQI e A/FR. O rastreamento desses índices é realizado com base nas metas de qualidade. Caso esses índices alcancem determinados patamares, pode-se assegurar, praticamente, que o produto entregue ao usuário estará livre de erros (RONG, 2010).

4. O campo valor agregado do PSP é usado para acompanhar o andamento do projeto. Basicamente, esse valor é calculado utilizando a percentagem do total planejado e uma regra de 0-100 é aplicada para calcular o valor cumulativo. O PSP possibilita aos membros do projeto obter informações, a qualquer momento, a respeito do *status* e datas do projeto, possibilitando a efetivação de ações corretivas, em um intervalo menor de tempo (RONG, 2010 e SANDE, 2011).

Scrum tem indicativos que pode integrar PSP, pois:

1. Na reunião de retrospectiva, os membros da equipe se reúnem para discutir o *status* do projeto, o desvio das estimativas iniciais, o *status* de qualidade e os riscos, dentre ouras questões. *Scrum* oportuniza à equipe identificar as necessidades de melhoria (RONG, 2010; SUPHAK, 2011 e SANDE, 2011). Além disso, existem reuniões diárias que podem eliminar possíveis obstáculos que impedem um bom andamento do trabalho.

2. *Scrum* define tanto os papéis quanto as responsabilidades, sendo adequado para pequenos e médios projetos e para equipes com poucos participantes (SUPHAK, WERASAK, 2011 e SANDE, 2011).

3. O plano de coleta de dados do PSP define o procedimento para coletar e armazenar dados, como o tempo gasto em cada fase, o tamanho do programa e os defeitos injetados e removidos

em todas as fases. Isto padroniza o trabalho de coleta de dados e faz com que os dados recolhidos sejam mais confiáveis (GUOPING, 2010 e SUPHAK, WERASAK, 2011).

2.5.4 Benefícios da combinação entre PSP e metodologias ágeis

Entre os processos ágeis existentes, há indícios ou fortes argumentos para a integração entre *PSP* com *XP*, *DSDM*, *RUP* e *Scrum*, mostrando fortes evidências de que PSP favorece a implementação do desenvolvimento ágil de *software*. Quando o PSP é incorporado a métodos ágeis, o processo do PSP deve, necessariamente, ser adaptado às necessidades próprias do projeto e aplicado em conformidade com este.

Nesse contexto, pode-se listar alguns benefícios do uso de PSP para complementar as metodologias ágeis de desenvolvimento de software (RONG, 2010; WILLIAMS, 2001; STEFANOVA, 2002, SANDE, 2011 e WENGRONG, SHEN, 2011):

1. PSP oferece métricas e mecanismos de coleta de dados;

2. PSP provê estimativas mais razoáveis;

3. PSP ajuda a otimizar os planos;

4. PSP oferece suporte para produzir produtos de software utilizando metas de qualidade;

5. PSP fornece documentação adicional para os métodos ágeis;

6. PSP ajuda os engenheiros a melhorar o seu desempenho pessoal, gradativamente.

7. PSP ajuda os métodos ágeis a monitorar e acompanhar o processo de *software*, agregando valor aos membros da equipe, que podem acompanhar o andamento e o *status* do projeto.

2.6 Considerações finais do capítulo

Este capítulo apresentou uma revisão bibliográfica dos principais conceitos a serem utilizados como base para esta pesquisa. A revisão focou na integração do PSP com métodos ágeis (especialmente Scrum). Estudar o PSP, suas características, seus níveis de processos e seus materiais (artefatos), bem como a estrutura do Scrum e seu funcionamento, foi fundamental para definir o posicionamento do pesquisador com relação aos desafios da pesquisa, indicando como tais referências contribuíram para o trabalho.

A revisão também forneceu ideias mais precisas sobre os estudos atuais do tema e sobre as contribuições que o trabalho pode oferecer.

3 PROCESSO PROPOSTO: SCRUM-PSP-IF

O modelo integrado e iterativo de processo SCRUM-PSP-IF descreve as atividades necessárias para criar um produto de *software* livre de defeitos, integrando processo de desenvolvimento disciplinado e ágil. Foi instanciado a partir do modelo de Guoping Rong, Dong Shao e He Zhang, SCRUM-PSP (RONG et al., 2010), que propôs a combinação de dois modelos de processos, onde o *Scrum* forneceu o *framework* de gerenciamento de processos ágeis e o PSP proveu a cada membro da equipe a disciplina necessária para medir e estimar o seu trabalho. Portanto, utilizou atividades e artefatos de ambas as estruturas.

Para tornar o processo mais prático, adaptações ao modelo original foram necessárias. Portanto torna-se necessário destacar o que foi mantido do modelo original e o que foi modificado no modelo instanciado.

Pontos alterados do modelo original:

- Número de fases, quatro ao invés de cinco: lançamento, requisitos e plano, construção e *postmortem* (modelo instanciado). Lançamento, plano, requisitos e projeto, construção e *postmortem* (modelo original).
- Nível de detalhamento do modelo, com a clara definição de quais atividades e materiais do PSP o compõem (*templates*, formulários e *scripts*), associando, de forma mais concisa, aos eventos e artefatos *Scrum* (*backlog* do produto, *backlog* da *sprint* e reuniões).
- Aplicação do modelo apenas em projetos de *software* de pequeno porte, desconsiderando os de médio e grande porte.

Pontos mantidos do modelo original:

- Modelo projetado em duas camadas, de ciclo de vida e de iteração;
- Fases de lançamento, construção e *postmortem* mantidas.

Com a redução das fases, algumas atividades foram suprimidas, outras realocadas ou redefinidas. No decorrer da descrição do modelo todas as alterações serão devidamente referenciadas e contextualizadas, procurando salientar o que foi mantido do modelo original e o que foi alterado ou aprofundado no novo modelo.

O modelo instanciado visa estabelecer um alto nível de confiabilidade para gerenciar defeitos no desenvolvimento ágil de *software*, através da aplicação de alguns métodos do PSP (em

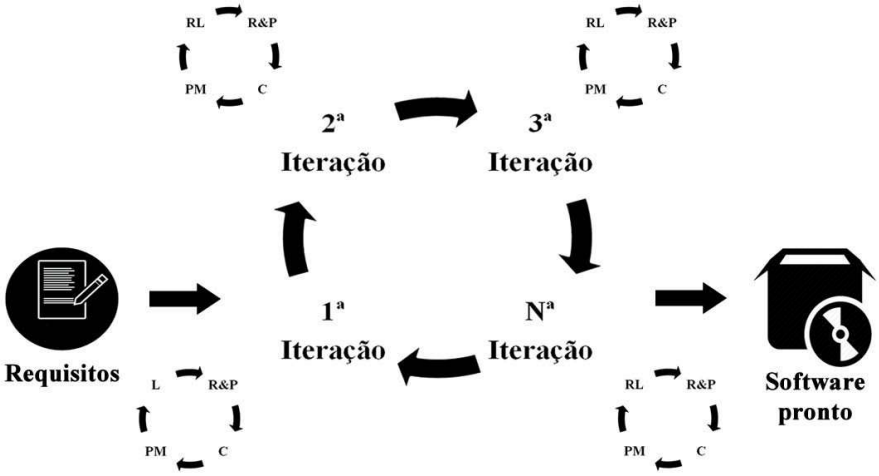
conformidade com as regras *Scrum*), com base em dados de tempo, tamanho e defeitos, coletados a partir das experiências individuais dos programadores nas iterações do desenvolvimento. Por isso é fundamental coletar dados sobre os defeitos injetados, contemplando as fases em que foram injetados, encontrados e removidos, bem como o tempo necessário para essas atividades.

Assim como no modelo base, o SCRUM-PSP-IF foi projetado em duas camadas: Ciclo de vida e Iteração (RONG et al., 2010). A primeira descreve a estrutura geral do processo, enquanto a segunda apresenta as fases que devem ser executadas para transformar as necessidades dos clientes em produtos de *software* de alta qualidade.

3.1 Ciclo de vida do SCRUM-PSP-IF

O SCRUM-PSP-IF apresenta ciclo de vida iterativo, onde uma parte dos requisitos do produto (refinados no decorrer dos ciclos), elencados pelo solicitante, fornece a entrada do processo. A partir daí, o trabalho é dividido em iterações (com prazo fixado entre duas e quatro semanas) e um incremento executável deve ser apresentado ao cliente no final de cada iteração. A figura 04 representa o modelo SCRUM-PSP-IF com N iterações.

Figura 04 – Ciclo de vida do SCRUM-PSP-IF.



Fonte: adaptado de RONG et al., 2010.

As quatro fases, Lançamento (RL), Requisitos e Plano (R&P), Construção (C) e *Postmortem* (PM), representam atividades especificas dentro de cada iteração. Nas fases RL, R&P e PM, as atividades são de responsabilidade da equipe do projeto, o Time *Scrum*. Na fase de

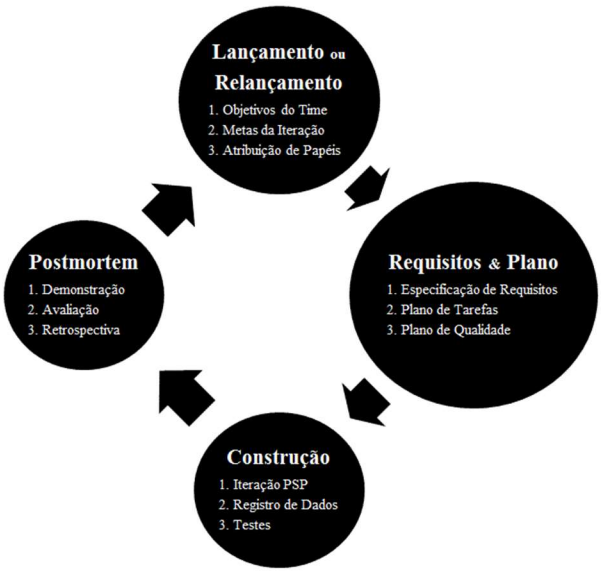
construção, onde a especificação dos requisitos é transformada em código de programação, um conjunto de guias, modelos e padrões do PSP deve ser utilizado para o desenvolvimento dos módulos do programa e registro dos dados. Essas atividades são de responsabilidade de cada desenvolvedor.

3.2 Iteração do SCRUM-PSP-IF

A camada de Iteração descreve os passos necessários para alcançaras metas de um determinado ciclo, que equivale a uma *sprint* do *Scrum*, que pode ser considerada como um *container* de tarefas realizadas em uma determinada janela de tempo.

Diferente do modelo original, composto por cinco fases (RONG et al., 2010), o SCRUM-PSP-IF é formado por quatro fases: Lançamento ou Relançamento (L ou RL), Requisitos & Plano (R&P), Construção (C) e *Postmortem* (PM). Cada fase possui suas próprias atividades, como pode ser visto na figura 05.

Figura 05 – Iteração do SCRUM-PSP-IF.



Fonte: adaptado de RONG et al., 2010.

Basicamente, a fase de Lançamento engloba atividades de formação do time *Scrum*. A atividade de especificação de requisitos, que no modelo original foi combinada com a atividade de projeto (fase de Requisitos e Projeto), foi combinada com a atividade de planejamento (fase de Requisitos e Plano), pois o objetivo da fase consiste em, a partir da obtenção dos requisitos do produto (ou grande parte deles), criar os planos de tarefas e de

qualidade, bem como gerar os *backlogs* do produto e das *sprints*. Já o modelo original considera os requisitos como insumos para um projeto de alto nível, criado pela equipe.

A fase de Projeto, que no modelo original fazia parte da fase de Requisitos e Projeto, foi removida, pois o *Scrum* não exige fase de projeto (ao contrário da fase de requisitos) ou, no mínimo, não salienta sua obrigatoriedade. Além do que, para o PSP, em sua estrutura básica, a fase de projeto pessoal, que poderia indicar a necessidade da fase suprimida, tem como entrada os requisitos e não projeto de alto nível da própria equipe, por exemplo. Isso significa que a criação do projeto de alto nível pela equipe, exigido pelo modelo original, pode ser descartado pelo novo modelo, pois o PSP não solicita como entrada (ou atividade) de nenhuma de suas fases o projeto feito pela equipe, mas sim o projeto pessoal de alto nível, criado por cada indivíduo.

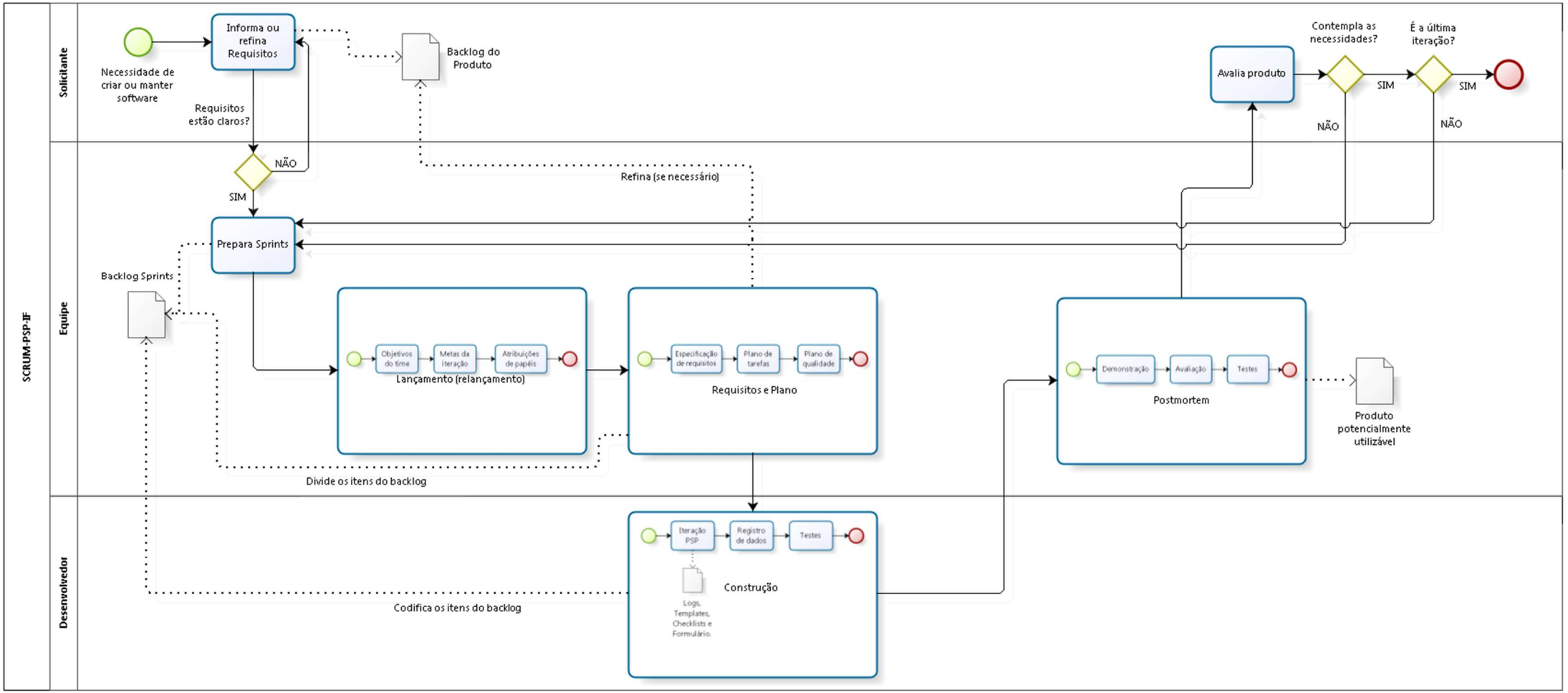
A figura 06 apresenta um modelo BPMN (*Business Process Model and Notation*) do ciclo de vida e das iterações do SCRUM-PSP-IF.

Os papéis envolvidos no processo modelado são: usuário solicitante, a equipe de desenvolvimento e os engenheiros de *software*. É o solicitante que realiza o evento de início, declarando os requisitos do produto. Se estes, representados pelo *Backlog* do Produto, não estiveram claros o suficiente, a equipe de desenvolvimento solicita ao usuário que refine os refine. Caso os requisitos estejam com um nível de detalhamento suficiente, a equipe de desenvolvimento pode preparar as *Sprints* e gerar os artefatos de *Backlog* das *Sprints*, passando pelas fases de Lançamento (ou relançamento, para as iterações que não forem a primeira) e Requisitos e Plano.

A partir das atividades de equipe citadas anteriormente, as atividades da fase de construção podem ser realizadas por cada desenvolvedor, utilizando os oito passos do SCRUM-PSP-IF (para a fase de construção). É muito importante o programador preencher os artefatos (*logs*, formulários e *checklists*), para cada iteração sob sua responsabilidade.

Por fim, o trabalho volta à equipe de desenvolvimento que, em conjunto com o solicitante avalia se o produto (ou parte dele) está de acordo com as necessidades dos usuários. Se o produto não contempla as necessidades, as suas funcionalidades são refeitas ou melhoradas nas próximas iterações. Caso contemple as necessidades, o produto é entregue ou o seu desenvolvimento pode ser continuado (caso não seja a última iteração).

Figura 06 – BPMN do modelo SCRUM-PSP-IF.



Fonte: elaborado pelo autor.

3.2.1 Fase 1: Lançamento ou relançamento

Cada iteração do SCRUM-PSP-IF começa, obrigatoriamente, com a fase de Lançamento (na primeira iteração) ou de Relançamento (a partir da segunda iteração). O principal objetivo da fase de lançamento é formar a equipe do projeto e discutir sobre a estratégia de desenvolvimento.

Atividades típicas de formação do time incluem a definição do objetivo da equipe e da meta da iteração, bem como a atribuição de papeis aos membros dessa equipe. Considerando que o *Scrum* atribui três tipos de papeis – *Product Owner* (proprietário ou solicitante do produto), *Scrum Master* (líder do projeto) e o *Development Team* (time de desenvolvimento) – o SCRUM-PSP-IF seguirá essa classificação para nomear seus participantes.

Nos lançamentos apresentados em iterações que não sejam a primeira – Relançamentos –não é obrigatório realizar todas as atividades da fase de Lançamento. Geralmente, os objetivos do time e a atribuição de papeis se mantêm inalterados. Porém, as metas da iteração tendem a mudar com frequência, devido, principalmente, aos refinamentos (detalhamentos) dos requisitos do produto informados pelo solicitante ou pela mudança de requisitos.

Por decisões de projeto, o SCRUM-PSP-IF utiliza apenas a estratégia de desenvolvimento de *software* com equipe interna, descartando as possibilidades de terceirização ou trabalho em equipe mista (terceirizada e interna). Outras decisões sobre a estratégia de desenvolvimento, como o número e o tempo das iterações, bem como a priorização dos requisitos já especificados podem ser discutidas durante a fase de Lançamento. Mesmo o R&L não exigindo nenhum artefato específico, é altamente recomendado que algum registro sobre essas atividades seja mantido pelo *Scrum Master*.

Principais atividades da fase:

- Definição dos objetivos do time e da iteração.

- Atribuição de papeis aos membros do time.

3.2.2 Fase 2: Requisitos & plano

A segunda fase da Iteração envolve todos os membros da equipe em atividades de especificação de requisitos e criação de alguns planos. Nas atividades de requisitos devem ser criados, priorizados e estimados todos os itens do *backlog* do produto, proporcionando maior clareza para o planejamento e execução das iterações.

A discussão sobre a estratégia de desenvolvimento, iniciada na fase de Lançamento, deve ser retomada nessa fase, pois agora se tem com maior clareza das necessidades dos usuários e, assim, se pode estimar com mais precisão o número de iterações e a prioridade entre os requisitos, além da possibilidade de utilizar novas tecnologias.

Atividades de planejamento envolvem a criação dos planos de estimativa de tarefas (considerando os recursos disponíveis) e plano de qualidade (utilizando os índices de qualidade do PSP presentes no SCRUM-PSP-IF).

Especificação dos requisitos de *software – Backlog* do produto

A especificação dos requisitos no SCRUM-PSP-IF tem como objetivo captar todas as necessidades do solicitante para um determinado produto de software, onde os requisitos são representados por uma lista ordenada, nomeada Backlog do Produto. Esse artefato Scrum contém as características, funções, melhorias e correções do produto. Sua criação, priorização e manutenção são de responsabilidade do Product Owner, guiado pelo Scrum Master e pelo Time de desenvolvimento.

Os itens do backlog devem possuir, no mínimo, atributos de descrição, ordem, estimativa e valor. O valor está relacionado com um intervalo numérico, que pode ser, por exemplo, de zero a cem, que ordena, de forma crescente, a importância do requisito. Portanto, quanto maior o valor, maior deve ser a prioridade e o nível de detalhes do item, pois estes serão desenvolvidos nas primeiras Sprints.

Os itens que podem ser entregues pela equipe são considerados como “Preparados” para seleção na reunião de planejamento da sprint, que transforma esses itens em tarefas. Como é de responsabilidade do time de desenvolvimento fazer a estimativa final, o solicitante do produto deve influenciar o time, ajudando no entendimento e nas decisões conflituosas.

Na tarefa de estimar o tamanho dos itens da lista de requisitos, através de técnicas de divisão do trabalho em unidades menores, que possam ser mais facilmente estimadas, Rong et al. (2010) sugere uma estratégia similar ao desenvolvimento de uma Estrutura Analítica de Projetos (*WBS - Work Breakdown Structure*), subdividindo o trabalho e as entregas em partes menores (da mais geral para mais específica). Porém, isso não está diretamente relacionado com as práticas do *Scrum*, como o SCRUM-PSP-IF sugere.

Por mais que o *Scrum* não descreva técnicas específicas de estimativa, algumas técnicas de comparação são indicadas e comumente utilizadas por times ágeis. O mecanismo genérico consiste, basicamente, em comparar itens similares no *backlog* do produto, que já tenham sido implementados anteriormente.

A medida mais utilizada no *Scrum* é o *Story Point* (ponto de história), uma medida aleatória usada pelas equipes ágeis para medir o esforço necessário na implantação de uma *User Story* (história de usuário), que representa um ou mais requisitos (itens). A história de usuário é uma narrativa com foco nos objetivos do usuário e na maneira com que o sistema pode ajudá-lo na busca desses objetivos.

O *backlog* do produto não necessita estar completo desde o início do projeto e, na prática, dificilmente estará, mas com o tempo, ele vai sendo refinado até estar completo. Seus itens são priorizados durante a reunião de planejamento da *sprint*, pelo solicitante do produto, juntamente com a equipe. Cada item que pode ser desenvolvido na próxima iteração, elencado pelo time de desenvolvimento, é então transferido para o *backlog* da *sprint*, representado por uma ou mais tarefas, que serão divididas entre os programadores do time.

Plano de tarefas – *Backlog* da *sprint*

O plano de tarefas do SCRUM-PSP-IF tem como objetivo separar, de forma concreta, as ações necessárias para cada programador ter condições de utilizar os artefatos e técnicas da fase de Construção para desenvolver os módulos do sistema e atingir os objetivos das iterações. Para isso, fornece os insumos e o tempo necessário para a realização das tarefas, alocando esses recursos na *backlog da sprint*, um artefato *Scrum* que representa o conjunto de itens selecionados, pelo time de desenvolvimento, diretamente do *backlog* do produto, para ser desenvolvido na próxima iteração.

As tarefas, que podem ser técnicas, por exemplo, codificar uma interface, ou não técnicas, por exemplo, criar um manual *online*, devem ser distribuídas durante os dias da janela de tempo da iteração. Ao utilizar os roteiros da fase de Construção, as tarefas do *backlog* da *sprint* devem ser atualizadas, contendo o detalhamento das atividades da Iteração PSP, onde os requisitos são transformados em código fonte do programa.

Nesse momento pode ser usado algum *software* de gerenciamento de tarefas, que forneça recursos para registrar e, principalmente, acompanhar as tarefas da equipe, que serão atribuídas, futuramente, aos membros da equipe, fazendo parte do seu planejamento pessoal.

Plano de qualidade

O foco de qualidade do SCRUM-PSP-IF é encontrar e corrigir defeitos, considerando que esse termo pode se referir a qualquer coisa que prejudique a capacidade do programa de atender completa e efetivamente as necessidades do usuário (HUMPHREY, 2000a). Esses defeitos podem ser erros técnicos ou não técnicos, encontrados em qualquer fase do desenvolvimento de *software*, não só nas fases de projeto e codificação.

Nesse contexto, o plano de qualidade tem o objetivo de medir, avaliar e gerenciar a qualidade de um programa, utilizando um conjunto de medidas do PSP para auxiliar o programador a analisar a qualidade de seus programas, sob várias perspectivas. Embora nenhuma medida, isoladamente, possa indicar a qualidade global de um programa, agregando algumas medidas do PSP pode-se chegar a indicadores de qualidade confiáveis, pois é possível avaliar a qualidade do trabalho sob mais de uma perspectiva.

No SCRUM-PSP-IF, as atividades do plano de qualidade devem ser realizadas utilizando algumas métricas que servirão de referência para os desenvolvedores. A mais importante e obrigatória consiste no Índice de Qualidade do Processo (PQI - *Process Quality Process*), que agrega cinco medidas de qualidade (*Design/Code Time*, *Design Review Time*, *Code Review Time*, *Compile Defects/KLOC* e *Unit Test Defects/KLOC*), para avaliar o processo de desenvolvimento de *software*. O PQI é o produto dessas cinco métricas. Porém, o SCRUM-PSP-IF não irá utilizar a medida *Compile Defects/KLOC*, pois os ambientes integrados de desenvolvimento (IDE) da atualidade, praticamente não permitem que um código seja executado com erros sintáticos e semânticos, realizando um trabalho de revisão de código nesses programas.

A partir de agora, serão citadas e descritas algumas das principais métricas do PSP utilizadas pelo SCRUM-PSP-IF.

$PQI_{DesignCodeTime}$= DesignTime/CodeTime – representa a qualidade do projeto, sendo expressa pela divisão entre o tempo de projeto e o tempo de codificação. É recomendado que o tempo gasto na fase de projeto seja maior que o tempo gasto na fase de codificação. Em ambientes ágeis de desenvolvimento, dificilmente esse índice seria alcançado.

$PQI_{DesignReviewTime}$ = 2 . DesignReviewTime/DesignTime – representa a qualidade de revisão de projeto, sendo expressa pela divisão entre o dobro do tempo de revisão de projeto e o tempo de projeto. É recomendado que o tempo gasto na fase de revisão de projeto seja, no mínimo, metade do tempo gasto na fase de projeto.

$PQI_{CodeReviewTime}$ = 2 . CodeReviewTime/CodeTime – representa a qualidade de revisão de código, sendo expressa pela divisão entre o dobro do tempo de revisão de código e o tempo de codificação. É recomendado que o tempo gasto na fase de revisão de código seja, no mínimo, a metade do tempo gasto na fase de codificação.

$PQI_{UnitTestDefects}$ = 10/5+UnitTestDefectsPerKLOC – representa a qualidade do programa, sendo expressa pela divisão de 10 pela soma do número de defeitos (encontrados nos testes unitários a cada 1000 linhas de código) com 5, que representa o valor alvo para essa métrica, ou seja, a densidade de defeitos da fase de testes unitários não deve ser superior a 5 defeitos/KLOC.

Os componentes PQI foram normalizados para [0, 1] no PSP, sendo que zero (0) indica baixa qualidade no processo, enquanto um (1) representa alta qualidade no processo. Isso significa que os valores das medidas descritas anteriormente devem estar dentro desse intervalo. Segundo relatório técnico do SEI (HUMPHREY, 2000a), o valor global de PQI consiste no produto das cinco medidas e deve estar acima de 0,4 para apresentar um índice de qualidade aceitável.

No SCRUM-PSP-IF também podem ser calculadas, desde que se tenham os registros necessários de cada programador, a Densidade de defeitos (*Defect density*), a Taxa de revisão (*Review rate*), as Relações de defeito (*Defect ratios*), o Rendimento (*Yield*) e a Relação entre avaliação e falha (*A/FR - Appraisal to Failure Ratio*). Todas essas medidas são oriundas do PSP.

Densidade de Defeito (*Defect density*): se refere aos defeitos encontrados em um programa a cada mil linhas de código (KLOC), considerando LOC novas e alteradas. Por exemplo, se um programa de 200 LOC tem 19 defeitos, a densidade de defeito será 1000 * 19/200 = 95 defeitos / KLOC, ou seja, 95 defeitos a cada 1000 linhas de código. Conforme (HUMPHREY, 2000a), a densidade de defeitos pode ser medida para todo o processo de desenvolvimento e para fases específicas de processo.

No PSP, um programa com cinco ou menos defeitos por KLOC, no teste unitário é considerado de boa qualidade. Para engenheiros de *software* que não tenham sido treinados e não utilizem PSP, os níveis típicos de defeitos de teste unitário variam de 20 a 40 ou mais defeitos a cada mil linhas de código (HUMPHREY, 2000a).

Taxa de revisão (*Review rate*): no PSP, tanto no projeto quanto no código, os engenheiros revisam seus programas, com base em *checklists*. Dados do PSP mostram que quando a revisão de projeto ou código é mais rápida do que cerca de 150 a 200 linhas de código novas ou alteradas por hora, mais defeitos tendem a passar despercebidos (HUMPHREY, 2000a). Kemerer e Paulk (2009) também sugerem uma taxa de inspeção de código inferior a 200 LOC por hora e, ainda, indicam uma taxa inferior a quatro (4) páginas por hora quando a inspeção for realizada na documentação do projeto do *software*.

Considerando as taxas de revisão citadas anteriormente, pode-se coletar os dados das revisões e determinar a rapidez com que essas atividades devem ser feitas para encontrar o maior número de defeitos possível. A taxa de revisão é uma medida que atua como uma espécie de guia, orientando cada programador quanto ao tempo que deve gastar nas revisões e inspeções de código e projeto.

Relações de Defeitos (*Defect ratios*): as relações de defeitos comparam os defeitos encontrados em uma fase com os defeitos encontrados em outra fase. As principais relações são entre os defeitos encontrados na fase de revisão de código e defeitos encontrados na fase de compilação e entre os defeitos encontrados na fase de revisão de projeto e defeitos encontrados na fase de teste unitário.

Uma regra razoável é que os engenheiros devem encontrar pelo menos o dobro de defeitos revisando o código do que compilando esse mesmo código (HUMPHREY, 2000a). O número de defeitos encontrados durante a compilação é uma medida objetiva da qualidade do código. Quando são encontrados mais do que o dobro de defeitos na revisão do código que na

compilação, indica que o programador fez uma revisão eficiente de código ou (em uma perspectiva negativa) que não registrou todos os defeitos de compilação.

Os dados também sugerem que os engenheiros devem encontrar pelo menos o dobro de defeitos revisando o projeto do que realizando teste unitário, isso indica que eles provavelmente fizeram revisões de projeto aceitáveis (HUMPHREY, 2000a).

Rendimento (*Yield*): o rendimento é medido de duas maneiras, rendimento de fase e rendimento de processo (total). O rendimento de fase mede a percentagem dos defeitos totais que são encontrados e removidos em uma fase. Por exemplo, se um programa entrou na fase de teste unitário com 17 defeitos e os testes de unidade encontrarem/removerem 10, o rendimento da fase de teste unitário será de aproximadamente 59%. Da mesma forma, se um programa entrou na revisão de código com 40 defeitos e essa revisão encontrou/removeu 14, o rendimento da fase de revisão de código será 35%. O rendimento do processo representa a percentagem dos defeitos removidos antes da primeira compilação e teste unitário.

Relação Avaliação/Falha (A/FR - Appraisal to Failure Ratio): essa relação mede o custo da qualidade do processo de engenharia, utilizando parâmetros de custo de qualidade (HUMPHREY, 2000a). O A em A/FR representa o custo de qualidade de avaliação, formado pelo total de tempo gasto nas revisões de projeto e código (multiplicado por 100), incluindo o tempo gasto reparando os defeitos encontrados nessas revisões, dividido pelo tempo total de desenvolvimento. O F em A/FR significa o custo de qualidade de falha, formado pelo total de tempo gasto na compilação e testes unitários (multiplicado por 100), incluindo o tempo gasto para encontrar, corrigir, recompilar e testar novamente os defeitos encontrados. A relação, representada pelo R, é formada pela divisão do resultado do custo de avaliação pelo resultado do custo de falha.

A medida A/FR fornece uma forma útil de avaliar a qualidade, tanto para um programa como para comparar a qualidade dos processos de desenvolvimento utilizados para vários programas. Com base em experiências anteriores com PSP, o SEI indicou o valor alvo para a relação A/F de cerca de 2,0, a fim de obter bom rendimento, a um custo global razoável de qualidade (HUMPHREY, 2000a).

O plano de qualidade não possui um *template* específico, pois esta contido no formulário de resumo de plano de projeto. Porém, é recomendado que todos os envolvidos no projeto tenham ciência dos valores base para rastrear a qualidade e recebam treinamento em PSP,

para conseguirem extrair e registrar os dados de tempo, tamanho, defeitos e qualidade. O *Scrum Master* geralmente tem o papel de ser o instrutor e orientar os programadores sobre o uso dessas informações.

Outros planos

Existem outros planos que podem ser usados, conforme as necessidades específicas dos projetos de *software*. Eles não são obrigatórios, mas podem suprir alguma necessidade não tão coberta pelos demais artefatos ou uma particularidade do projeto. O plano de riscos, configuração e mudança, coleta de dados, cronograma e monitoramento de projeto (RONG et al., 2010) são exemplos desses planos.

Um plano para gerenciar riscos descreve os riscos que podem impactar na realização do projeto ou de suas ações. Geralmente são avaliados de acordo com a probabilidade, impacto e severidade. Quanto maior for a probabilidade, o impacto e a severidade, maior é a necessidade de rastrear o risco. Avaliar coerentemente as respostas que devem ser dadas para mitigar ou eliminar os riscos contribui para a correta tomada de decisões no desenvolvimento de *software*.

O plano de configuração e mudança pode ser importante para a equipe de desenvolvimento, com foco na manutenção da integridade do *software* durante o projeto, além do controle sistemático das mudanças.

O plano de coleta de dados faz uso de técnicas para coletar os dados sobre o tempo gasto em cada fase do projeto, o tamanho do *software* e os defeitos injetados e removidos em todas as fases. Esse plano também deve definira forma de armazenamento dessas informações, pois elas servem de base para orientar a equipe.

Por fim, o plano de monitoramento do projeto, se for utilizado, faz uso de reunião diária (*daily meeting*) e valor agregado (EV-*Earned Value*), ambos oriundos do *Scrum*, para rastrear o andamento do projeto. Basicamente, o valor agregado possibilita medir e avaliar o avanço do projeto quanto a custos e prazos, com base no trabalho realizado relacionado com o planejado.

Para organizar de forma sistemática as tarefas, ferramentas *online* de gestão de projetos podem ser utilizadas, pois oferecem suporte para os registros e monitoramento das atividades.

Principais atividades da fase:

- Especificação dos requisitos do produto.

- Criação do plano de tarefas.

- Definição do plano de qualidade (medidas utilizadas).

- Criação de outros planos (opcional).

Principais artefatos da fase:

- *Backlog* do produto.

- *Backlog* da *sprint*.

3.2.3 Fase 3: Construção

A fase de construção do SCRUM-PSP-IF tem como objetivo codificar o *software*, transformando a especificação de requisitos, juntamente com o projeto pessoal (que contempla o preenchimento dos dois *templates* de projeto – especificação funcional e operacional), em um produto testado que atenda as necessidades dos usuários, além de ter os dados do processo registrados. Os itens do *backlog* do produto selecionados para a implementação são representados por um conjunto de tarefas específicas (*backlog* da *sprint*), considerando os recursos disponíveis e a produtividade da equipe. A principal atividade da fase de construção é a Iteração PSP.

O modelo de Rong et al. (2010) sugere que cada programador utilize um processo PSP2.1, composto por oito fases: planejamento, projeto, revisão de projeto, código, revisão de código, compilação, teste e *postmortem*. Porém, não sinaliza, especificamente, quais os *logs*, formulários, *scripts*, *templates* e padrões devem ser utilizados, o que limita o entendimento e, principalmente, a aplicabilidade do modelo.

Cabe ressaltar que o nível de processo PSP2.1 utiliza, além de seus próprios artefatos, adicionados apenas nesse nível, artefatos de todos os cinco níveis que o antecedem. Por isso é necessário ter clareza quando se propõe o uso de uma metodologia extremamente disciplinada.

Já o modelo instanciado SCRUM-PSP-IF procura definir, precisamente, quais documentos e métodos do PSP serão utilizados, sua ordem e forma de utilização (integralmente, em partes ou adaptados). Aplicar os materiais PSP em sua totalidade inviabilizaria, certamente, a agilidade procurada pelo novo modelo e, muito provavelmente, não seria utilizado integralmente pelos engenheiros de *software*.

Basicamente, o nível de processo PSP2.1 é composto por 23 documentos, presentes no material de auto estudo do PSP (SEI,2008):

- Sete *Scripts*: Processo, Planejamento, Desenvolvimento, Revisão de Projeto, Revisão de Código, *Postmortem* e Estimativa PROBE. O último *script* é utilizado a partir do processo PSP1, os demais são utilizadas em todos os níveis do PSP.
- Dois *Logs*: Registro de Tempo e Registro Defeitos. Todos os níveis de processo utilizam esses registros. São a base da coleta de dados do PSP.
- Dois Padrões: Tipos de Defeitos e Codificação. Adicionalmente, existe um terceiro padrão, Tipos de Defeitos Estendidos, que não está especificado em nenhum nível PSP, mas pode ser usado juntamente com o padrão de tipos de defeitos, pois utiliza também subcategorias de defeitos (enquanto o primeiro somente categorias). Todos os níveis do PSP utilizam padrão de tipo de defeitos. O padrão de codificação é introduzido a partir do PSP0.1.
- Oito *Templates* (modelos): Relatório de Teste e Estimativa de Tamanho (PSP1), Planejamento de Tarefas e Planejamento de Cronograma (PSP 1.1), Especificação de Operacional, Especificação Funcional, Especificação de Estado e Especificação Lógica (PSP2.1).
- Duas *Checklists* (listas de verificação): Revisão de Projeto e Revisão de Código (PSP2).
- Uma Proposta de melhoria de processo (PIP - *Process Improvement Proposal*). Utilizada a partir do PSP0.1.
- Um Resumo de Plano de Projeto. Todos os níveis de processos PSP possuem o seu próprio resumo de plano, incrementado ao longo dos níveis.

São atividades complementares à Iteração PSP (principal atividade), na fase de Construção, o Registro de Dados e os Testes de Sistema e de Integração. Essas atividades serão abordadas posteriormente.

Iteração PSP

A Iteração PSP do SCRUM-PSP-IF é baseada nas oito fases da estrutura do processo PSP: planejamento, projeto, revisão de projeto, código, revisão de código, compilação, teste e *postmortem*.

No PSP, há um *script* específico (ou uma atividade dentro do *script* de desenvolvimento) para orientar o trabalho nas oito (8) fases. Enquanto o programador segue as instruções dos *scripts* para fazer o trabalho, registra informações sobre o tempo gasto nas fases e informa dados sobre os defeitos, nos respectivos *logs*. No final do trabalho, durante a fase de *postmortem*, o programador resume os dados de tempo e defeitos dos *logs*, mede o tamanho do programa e registra essas informações no formulário de resumo de plano de projeto. É neste momento que o produto (ou parte dele) deve ser demonstrado e entregue ao cliente.

O *script* de Processo do PSP2.1, presente no material original de auto estudo do PSP (SEI, 2006), pode ser utilizado como primeira leitura para orientar, de maneira geral, o desenvolvimento de pequenos módulos de programas. Ele é composto por um resumo das atividades das fases de planejamento, desenvolvimento e *postmortem*. O desenvolvimento é composto, ainda, por outras seis fases, o que totaliza as oito fases que formam a estrutura de processo do SCRUM-PSP-IF.

Planejamento (preparação)

O planejamento pessoal, orientado pelo *script* de Planejamento PSP2.1, presente no material original do PSP (SEI, 2006), é a primeira fase da Iteração PSP e visa preparar o desenvolvedor para as demais atividades da Iteração. Originalmente descrito na forma de cinco atividades: requisitos do programa, estimativa de tamanho, estimativa de recursos, estimativa de defeitos e planejamento de tarefas e cronograma, não deve ser confundido com o planejamento de equipe, presente na fase de Requisitos e Plano.

No SCRUM-PSP-IF, as três atividades de estimativas (tamanho, recursos e defeitos) não serão utilizadas, nem os *templates* de planejamento de tarefas e cronograma (o que não impede o programador de registrar as tarefas, no *backlog* da *sprint*). Estimar o tamanho, os recursos e os defeitos, em nível pessoal, tornaria o processo pouco ágil, pelo excesso de

atividades e documentos a serem preenchidos. Portanto, essa fase está focada apenas na obtenção dos requisitos de *software*. A estimativa de tamanho, em LOC (*Lines of Code*), não faz parte do escopo do trabalho. Porém, o registro do tamanho real dos programas será realizado em um formulário especifico (resumo de projeto).

A falta de dados históricos, ao menos dos últimos três projetos (HUMPHREY, 2000a), inviabiliza a criação de uma estimativa de tamanho realista, além de fugir dos objetivos desse trabalho, que consiste na melhoria da qualidade de *software* através da correta gestão e redução de defeitos, utilizando *checklists*, *templates* e guias, com base em dados reais (atuais, não planejados) de projetos de desenvolvimento de *software*.

A estimativa de recursos individuais foi suprimida pelos mesmos motivos da atividade de estimativa de tamanho. Porém, a estimativa de recursos no nível de equipe, realizadas na fase anterior para montar os *backlogs* das *sprints*, pode fornecer ao programador os recursos necessários para realizar o seu trabalho. Com os registros dos primeiros projetos, pode-se ter a base necessária para estimar os recursos pessoais.

A estimativa de defeitos também foi suprimida, pelos mesmos motivos das atividades de estimativa de tamanho e de recursos. Porém, os registros dos dados reais de defeitos do projeto, bem como o tempo e as fases de injeção e remoção, são extremamente necessários para disciplinar os desenvolvedores e para utilizar o SCRUM-PSP-IF,

O planejamento de tarefas e cronograma do PSP não será utilizado, pois estimar essas atividades não é um dos focos desse trabalho, além de deixar o processo mais pesado (trabalhoso), pois mais dois templates deveriam ser preenchidos para essa atividade. Porém, os programadores podem registrar dados reais sobre as tarefas (nome da tarefa, fase que a tarefa representa, horas gastas na realização da tarefa) em qualquer ferramenta *online* de gerenciamento de projetos, inclusive complementando (atualizando) o *backlog* da *sprint* com essas atividades. Esse tipo de registro pode fornecer uma boa base para o programador acompanhar o progresso do cronograma do projeto, mantendo uma espécie de mural geral das tarefas da *sprint*.

Como critérios de entrada, a fase de preparação do SCRUM-PSP-IF solicita a descrição geral do problema, representada pela documentação inicial do projeto (escopo ou oficialização da demanda, por exemplo), o *log* de registro de tempo, o *backlog* do produto e o *backlog* da *sprint*. Apesar do *script* de Planejamento PSP2.1 não referenciar o *log* registro de defeitos

como critério de entrada (nem como critério de saída), ele deve ser inserido no processo desde o principio, conforme (SEI, 2006), em sua tabela de referência cruzada PSP. Esses são os insumos necessários para realizar a atividade descrita a seguir.

Definição de Requisitos do Programa: é a única atividade da fase de planejamento do SCRUM-PSP-IF. Consiste na obtenção da especificação dos requisitos do programa (*backlog* do produto) e das tarefas do *backlog* da *sprint*, garantindo clareza e não ambiguidade a essa especificação. Ainda pode ser necessário refinar alguns requisitos criados pela equipe, com o envolvimento do solicitante do produto, produzindo novas especificações. Nesse caso os *backlogs* de produto e da *sprint* devem ser alterados, refletindo as novas necessidades. O tempo gasto para essa atividade deve ser informado no *Log* de Registro de Tempo (quadro 04) e os defeitos encontrados no *Log* de Registro de Defeitos (quadro 05), traduzidos do material original do PSP (SEI, 2006). É pouco provável algum defeito ser encontrado na fase de planejamento, pois esta não apresenta nenhuma lista de verificação para requisitos, por exemplo.

Quadro 04 – Log de registro de Tempo do PSP.

Programador _____ Data _____
Programa _____ Programa # _____
Instrutor _____ Linguagem _____

Projeto	Fase	Data e Tempo de Inicio	Tempo de Interrupção	Data e Tempo de Fim	Tempo Delta	Comentários

Fonte: traduzido do material de auto estudo de SEI, 2006.

O registro de tempo é relativamente simples, porém deve ser realizada com atenção e, principalmente, regularidade, para refletir a realidade do trabalho de cada engenheiro de *software* em cada fase. Conforme o módulo ou parte do programa vai sendo projetado, construído e testado, o programador informa o nome da fase, a data e tempo de início das atividades naquela fase, o tempo de interrupção (um telefonema, um intervalo, uma saída da sala de trabalho ou até mesmo outras tarefas), a data e o tempo do término das atividades da fase. O tempo delta representa o tempo total de trabalho, por fase, subtraindo o tempo de interrupção.

Quadro 05 – Log de Registro de Defeitos do PSP.

Programador _____ Data _____
Programa _____ Programa # _____
Instrutor _____ Linguagem _____

Projeto	Data	Número	Tipo	Injetado	Removido	Tempo correção	Ref.
Descrição:							
Projeto	Data	Número	Tipo	Injetado	Removido	Tempo correção	Ref.
Descrição:							

Fonte: traduzido do material de auto estudo de SEI, 2006.

O preenchimento do *log* de registro de defeitos também pode ser considerado simples. Ao passar pelas fases de construção do programa (planejamento, projeto, revisão de projeto, código, revisão do código, compilação e teste), se qualquer erro for encontrado, o desenvolvedor deve utilizar o *log* de defeitos, informando a data, o número do defeito (sequencial), o tipo do defeito (conforme padrão de tipo de defeitos e padrão de tipos de defeitos expandido, a fase em que o defeito foi injetado, a fase em que o defeito foi removido e o tempo necessário para essa correção). Se um defeito foi injetado enquanto se consertava outro defeito, o número desse último deve constar no campo Ref.

Para orientar o registro dos defeitos, existe o Padrão de Tipos de Defeitos (ANEXO A) e, complementarmente, o Padrão Estendido de Tipos de Defeitos, que pode ser acessado no material original de auto estudo do PSP (SEI, 2006).

Como critério de saída da fase de planejamento se espera apenas uma especificação de requisitos devidamente documentada (ela já pode estar nesses moldes antes de chegar ao planejamento) e a atualização (se necessária) do quadro de tarefas (*backlog* da *sprint*), além do registro do tempo e dos defeitos, no *log* de tempo e *log* de defeitos, respectivamente.

Projeto

A segunda fase da Iteração PSP, projeto, não possui um *script* próprio, mas pode ser orientada pela atividade de Projeto, presente no *script* de Desenvolvimento PSP2.1, do material original do PSP (SEI, 2006).

Os passos da atividade de projeto do SCRUM-PSP-IF envolvem a revisão da declaração de requisitos e produção de um projeto para atender a esses requisitos. Para isso é necessário produzir e registrar o projeto detalhado nos formulários *templates* de Especificação Operacional (quadro 06) e Especificação Funcional (quadro 07), traduzidos do material original do PSP (SEI, 2006). Os templates de especificação de estado e de especificação lógica não serão utilizados pelo modelo, pois além de serem considerados menos relevantes que os outros dois templates da fase, tornariam o processo do SCRUM-PSP-IF muito pesado, demandando muito tempo para o seu preenchimento.

Os defeitos encontrados e corrigidos na fase de projeto devem ser registrados no *log* de Registro de Defeitos (quadro 05) e o tempo gasto nessa fase deve ser registrado no *log* de registro de tempo (quadro 04).

Quadro 06 – Template de Especificação Operacional do PSP.

Programador _____ Data _____
Programa _____ Programa # _____
Instrutor _____ Linguagem _____

Número do cenário		**Objetivo do usuário**	
Objetivo do cenário			
Fonte (Source)	**Etapa**	**Ação**	**Comentários**

Fonte: traduzido do material de auto estudo de SEI, 2006.

O objetivo do *template* operacional é apresentar as descrições dos prováveis cenários operacionais de uso do programa, assegurando que todos os problemas significativos de uso sejam considerados durante o projeto do programa. Além de especificar os cenários de teste.

Pode ser usado tanto para módulos de sistemas quanto para sistemas completos, agrupando vários cenários em um único modelo, desde que sejam claramente distinguidos e tenham

objetivos relacionados. Ao final da implementação e dos testes, esse modelo pode ser atualizado para refletir o produto implementado.

Conforme as instruções do SEI (2006) para especificação operacional, o número do cenário representa a referência única do cenário. O campo com o objetivo do usuário diz respeito ao propósito dos usuários para o cenário, como por exemplo, fazer *login* no sistema. O campo de objetivo do cenário representa o propósito do cenário, como por exemplo, definir os fluxos as condições e exceções para realizar o *login* no sistema.

A coluna fonte representa a origem da ação do cenário. Por exemplo, usuário, programa ou sistema. A coluna etapa fornece uma sequência de números para as etapas de cenário, facilitando as revisões e inspeções. Enquanto ação descreve as ações tomadas. Por exemplo, prover uma mensagem de erro. Por fim, os comentários servem para listar informações significativas relacionadas à ação. Por exemplo, o usuário insere um valor incorreto para ocorrer um erro.

De acordo com o item 2.3.8, PSP de nível 2.1, este modelo pode ser representado pelo diagrama de casos de uso da (UML), pois este mostra os cenários e as funcionalidades do sistema, incluindo usuários.

Quadro 07 – Template de Especificação Funcional do PSP.

Programador _____ Data _____
Programa _____ Programa # _____
Instrutor _____ Linguagem _____

Nome da Classe	
Classe Pai	

Atributos	
Declaração	**Descrição**

Itens	
Declaração	**Descrição**

Fonte: traduzido do material de auto estudo de SEI, 2006.

O *template* de especificação funcional de projeto tem como objetivo manter as especificações funcionais, descrevendo as classes, os módulos de programas ou programas inteiros. Da mesma forma que no projeto operacional, após a implementação e testes, as especificações funcionais podem ser atualizadas para refletir o produto implementado.

De acordo com as instruções do PSP SEI (2006), o nome da classe e das classes herdadas deve ser preenchido, podendo até ser listada a hierarquia completa de herança. Os atributos fornecem a declaração e descrição de cada variável ou parâmetro. A coluna itens fornece a declaração e descrição de cada item, informando precisamente as condições que regem os valores de retorno de cada item. São exemplos de itens métodos de classe, procedimentos, funções ou consultas de banco de dados.

De acordo com a seção 2.3.8, PSP de nível 2.1, este modelo pode ser representado pelo diagrama de classes da UML, que representa os objetos do sistema e seus relacionamentos.

Revisão de projeto

A terceira fase da Iteração PSP, revisão do projeto, é guiada pelo *script* de Revisão de Projeto do PSP2.1, presente no material original do PSP (SEI, 2006). Está dividida em três atividades, que englobam a preparação, a própria revisão e a verificação e correção do que foi revisado.

O principal critério de entrada dessa fase é o projeto do programa documentado nos dois *templates* de projeto, juntamente com a *Checklist* de Revisão do Projeto do PSP2.1 (quadro 08), traduzida do material original do PSP (SEI, 2006). Os materiais do PSP, com exceção dos *scripts* (específicos dos níveis), são inseridos em determinado nível de processo (do PSP0 ao PSP2.1) e servem, integralmente, para os níveis posteriores (*logs*, *templates* e *checklist* de código). Porém, a *checklist* de revisão de projeto, inserida no nível 2 do PSP, apresenta uma coluna a mais no nível PSP 2.1 (análise de estado). É essa lista de verificação que será utilizada no SCRUM-PSP-IF. Com esses artefatos, somados aos *logs* de registros de tempo e defeitos, o codificador tem condições para corrigir e registrar um maior número de defeitos encontrados no projeto e gravar o tempo dessas atividades, pois a *checklist* fornece os itens que devem ser verificados.

Quadro 08 – Checklist de Revisão de Projeto PSP2.

Programador _____ Data _____
Programa _____ Programa # _____
Instrutor _____ Linguagem _____

Objetivo	Para guiá-lo na realização de uma revisão de projeto efetiva.
Geral	- Revise todo o projeto do programa para cada categoria da *checklist*. Não tente revisar mais de uma categoria por vez! - Ao concluir cada etapa de revisão, marque esse item na caixa à direita. - Complete a *checklist* para um programa ou unidade de programa antes de rever o próximo.

V significa que o item de revisão não encontrou erros. X significa que o item de revisão encontrou um ou mais erros. Os números representam as revisões.		1	2	3	4
Completo	- Verifique se o projeto abrange todos os requisitos aplicáveis. • Todas as saídas especificadas são produzidas. • Todas as entradas necessárias são fornecidas. • Todas as exigências são incluídas.				
Lógica	- Verifique se o sequenciamento lógico do programa é apropriado. - Examine cada declaração condicional, verificando todos os casos.				
Análise de Estado	Para cada máquina de estado, verifique se as transições de estado estão completas e ortogonais.				
Casos especiais	- Verifique todos os casos especiais. - Assegure o funcionamento adequado com valores vazios, completos, mínimos, máximos, negativos para todas as variáveis. - Proteja contra condições de fora de limites, condições de *overflow* e *underflow*. - Assegure que condições "impossíveis" são absolutamente impossíveis. - Manipule todas as possíveis condições incorretas ou de erro.				
Uso Funcional	- Verifique se todos os atributos, funções, procedimentos ou métodos são totalmente compreendidos e usados corretamente. - Verifique se todas as abstrações referenciadas externamente são definidas com precisão.				
Considerações sobre o sistema	- Verifique se o programa não faz com que os limites do sistema sejam excedidos. - Verifique se todos os dados sensíveis à segurança são de fontes confiáveis. - Verifique se todas as condições de segurança estão em conformidade com as especificações de segurança.				
Nomes	Verifique se: - Todos os nomes são claros, definidos e autenticados. - Os escopos de todas as variáveis e parâmetros são auto evidentes ou definidos. - Todos os itens nomeados são usados dentro de seus escopos declarados.				
Padrões	- Certifique-se de que o projeto está em conformidade com todos os padrões de projeto aplicáveis.				
Uso Operacional (adicionado pelo autor)	Verifique se o modelo operacional: - Define claramente o fluxo principal e alternativo(s). - Define claramente as exceções e saídas. - Define claramente as mensagens e telas do sistema.				

Fonte: traduzido e adaptado do material de auto estudo de SEI, 2006.

Preparação: o próprio projeto e a *checklist* de revisão fornecem uma boa base para o programador decidir sobre qual estratégia de revisão utilizar. Ele deve examinar e tentar identificar no projeto do programa os *loops* internos, variáveis e limites do sistema. Pode, também, usar uma tabela de rastreamento ou qualquer outro método analítico para verificar a corretude do projeto.

Revisão: para cumprir essa atividade o programador deve seguir, cuidadosamente, a *checklist* de revisão do projeto (quadro 08), conferindo o projeto inteiro para cada categoria da *checklist*, marcando cada item nas colunas a esquerda dessa lista de verificação, corrigindo e registrando os erros encontrados (*log* de defeitos). Uma nova *checklist* deve ser utilizada para cada novo módulo do produto. O tempo gasto nas revisões também deve ser registrado (*log* de tempo).

Correção e verificação: o objetivo dessa atividade é verificar se os defeitos corrigidos na revisão estão realmente corretos, reexaminando todas as alterações e registrando todos os novos defeitos encontrados e corrigidos nos respectivos *logs*.

Como critério de saída dessa fase espera-se ter um projeto do programa devidamente revisado, por uma ou mais *checklists*, com os resultados documentados, com todos os defeitos identificados e corrigidos e as correções verificadas. Como nas outras fases, o tempo alocado para todas as atividades de ser registrado no *log* de tempo.

Código

Assim como a fase de projeto, a fase de código do SCRUM-PSP-IF não possui um *script* próprio para orientá-la, mas utiliza instruções vindas da atividade de código, presente no *script* de Desenvolvimento PSP2.1, do material original do PSP (SEI, 2006).

O programador deve codificar o projeto do programa em uma determinada linguagem de programação, seguindo um padrão de codificação, atrelado a uma determinada linguagem de programação (C++, por exemplo). Deve considerar, ainda, as demais tecnologias necessárias para essas tarefas, como tipo de banco de dados, IDE (*Integrated Development Environment* ou Ambiente de Desenvolvimento Integrado), *framework* de desenvolvimento, ferramentas de testes e integrações, entre outras. O programador deve registrar todos os defeitos encontrados nessa fase no *log* de registro de defeitos e o tempo gasto no *log* de registro de tempo.

Revisão de Código

A quinta fase da Iteração PSP do SCRUM-PSP-IF é orientada pelo *script* de Revisão de Código, presente no material original do PSP (SEI, 2006) e tem o objetivo de revisar o código desenvolvido na fase anterior, utilizando a *checklist* de Revisão de Código (quadro 09), traduzida do material original do PSP (SEI, 2006). É composta por três atividades principais: revisar, corrigir e verificar.

Revisar: para realizar essa atividade o programador deve seguir, cuidadosamente, a *checklist*, revisando o programa inteiro para cada categoria da lista e marcando os itens, conforme a sua conclusão. Dependendo do contexto, podem ser utilizadas mais de uma *checklist* de revisão de código por programa ou procedimento.

Corrigir: nessa atividade o programador vai corrigir todos os defeitos encontrados no código e registrar no *log* de defeitos. Se a correção não puder ter sido concluída, a revisão deve ser anulada e o programador deve voltar à fase anterior.

Verificar: na terceira atividade o programador verifica a correção, reexaminando cada alteração e registra nos *logs* os dados sobre os reparos e o tempo gasto para tais tarefas.

Como critérios de saída da fase de revisão de código tem-se o código fonte totalmente revisado, a *checklist* de revisão do código devidamente executada e os *logs* de tempo e defeitos atualizados.

Quadro 09 – Checklist de Revisão de Código do PSP.

Programador _____ Data _____
Programa _____ Programa # _____
Instrutor _____ Linguagem _____

Objetivo	Para guiá-lo na condução de uma revisão de código efetiva.
Geral	- Rever o programa inteiro para cada categoria de lista de verificação. Não tente revisar mais de uma categoria de cada vez! - Ao concluir cada etapa de revisão, marque esse item na caixa à direita. - Complete a lista de verificação para um programa ou unidade de programa antes de rever o próximo.

V significa que o item de revisão não encontrou erros. X significa que o item de revisão encontrou um ou mais erros. Os números representam as revisões.		1	2	3	4
Completo	- Verifique se o código abrange todo o projeto.				
Includes	- Verifique se os *includes* estão corretos.				

Inicialização	- Verifique a inicialização de variáveis e parâmetros. • No início do programa. • No início de cada loop. • Na entrada da classe / função / procedimento.				
Chamadas	- Verifique os formatos de chamada de função. • Ponteiros. • Parâmetros. • Uso de '&'.				
Nomes	- Verifique a ortografia e o uso do nome. • É consistente? • Está dentro do escopo declarado?				
Strings	- Verifique se todas as strings estão: • Identificadas por ponteiros. • Terminadas por NULL.				
Ponteiros	- Verifique se: • Ponteiros são inicializados por NULL. • Ponteiros são deletados somente após o novo. • Novos ponteiros são sempre apagados após o uso.				
Formato de saída	- Verifique o formato de saída. • "*Line stepping*" é apropriado. • O espaçamento é apropriado.				
() Pares	- Certifique-se de que () são adequados e correspondentes.				
Operadores Lógicos	- Verifique o uso correto de ==, =, ||, e assim por diante. - Verifique cada função lógica para ().				
Verificação linha-a-linha	- Verifique cada linha de código para: • Sintaxe da instrução. • Pontuação adequada.				
Padrões	Certifique-se de que o código está em conformidade com os padrões de codificação (adicionado pelo autor, conforme Guia do Codificador do PDS do IFFAR): - Certifique-se que o arquivo PHP: • Não tenha mais de 2000 linhas. • Inicie com um comentário descritivo. • Não tenha mais do que 80 caracteres em uma mesma linha. • Tenha comentários nos parâmetros e no retorno dos métodos, bem como nos atributos (variáveis). • Tenha Métodos separados por uma linha em branco (no máximo duas). • Não tenha mais de uma única instrução simples. • Não contenha parênteses para include, require, include_once e require_once.				
Abrir e Fechar Arquivo	- Verifique se todos os arquivos são: • Devidamente declarados. • Abertos. • Fechados.				

Fonte: traduzido e adaptado do material de auto estudo de SEI, 2006.

Compilação

A fase de compilação, assim como a fase de projeto e a fase de código, não possui um *script* próprio para guiá-la, mas sim uma atividade dentro do *script* de Desenvolvimento PSP2.1, do material original do PSP (SEI, 2006). Dependendo da linguagem de programação, o programador deve compilar ou interpretar o programa até que não exista nenhum erro na compilação ou interpretação, ou seja, ele deve corrigir os defeitos encontrados, mantendo atualizados os *logs* de defeitos e tempo. Não é objetivo desse trabalho entrar em detalhes à respeito dos tipos de linguagens de programação, desde que o programa seja devidamente compilado ou interpretado. Aliás, o SCRUM-PSP-IF não deve oferecer barreiras tecnológicas para a sua aplicação.

Testes

A fase de testes (unitários), assim como a de projeto, código e compilação não possui um *script* próprio, sendo orientada pela atividade de mesmo nome do *script* de Desenvolvimento PSP2.1, do material original do PSP (SEI, 2006). Tem como objetivo verificar se o código executável satisfaz os requisitos e projeto previamente elencados. Portanto, os cenários para esses testes devem partir da especificação de requisitos, considerando o projeto do programa, principalmente as partes que tratam dos cenários de uso (especificação operacional).

O programador deve realizar os testes unitários no módulo desenvolvido e preencher o *template* de Relatório de Testes (quadro 10), traduzida do material original do PSP (SEI, 2006), até que não encontre mais erros. Os defeitos encontrados e o tempo devem ser registrados nos *logs* de defeitos e tempo, respectivamente.

Quadro 10 – Template de Relatório de Teste do PSP.

Programador		Data	
Programa		Programa #	
Instrutor		Linguagem	

Nome/Número do Teste	
Objetivo do Teste	
Descrição do Teste	
Condições do Teste	
Resultados esperados	
Resultados reais	

Fonte: traduzido do material de auto estudo de SEI, 2006.

O objetivo do relatório de teste consiste em manter um registro da execução dos testes e dos resultados obtidos. Dessa forma pode-se manter padrões de execuções para garantir que os mesmos resultados sejam obtidos, por qualquer programador, em outros momentos.

Conforme as instruções do SEI (2006), o nome e número do teste identificam de forma única cada teste, considerando os cenários e os dados. O programador deve informar, resumidamente, o objetivo do teste, além de descrever os dados e procedimentos de cada teste com detalhes suficientes para facilitar sua reutilização, se necessário. As condições de teste listam qualquer configuração especial, cronometragem ou outras condições do teste. Quando vários testes são executados em diferentes condições ou diferentes parâmetros, devem ser listados separadamente. Os resultados esperados listam os resultados que o teste deve produzir se funcionar corretamente, enquanto os resultados reais listam os resultados que foram realmente produzidos, mesmo que não sejam os esperados.

Postmortem

A última fase da Iteração PSP, *postmortem* pessoal, é orientada pelo *script* de *Postmortem* PSP2.1, presente no material original do PSP (SEI, 2006). No SCRUM-PSP-IF esta fase tem o objetivo de resumir e analisar os dados individuais das fases anteriores, considerando o valor real (atual) de tamanho, tempo, defeitos e qualidade.

Dividida em quatro atividades, contempla o registro de defeitos, a consistência desses registros, a contagem de tamanho do programa e a revisão do tempo (SEI, 2006). O SCRUM-PSP-IF acrescenta uma atividade a essa fase, o Registro da Qualidade da Iteração, que contém os índices de qualidade mensurados. Como não foi identificado, exatamente, o momento do registro dos dados qualidade no PSP, o SCRUM-PSP-IF solicita esse registro no momento de preenchimento do formulário de resumo do plano, pois nesse momento o programador já tem em mãos todos os dados de qualidade necessários, ao final da iteração.

Possui como critérios de entrada a descrição do problema, a especificação de requisitos, o formulário de resumo, os *templates* de projeto, o *template* de relatório de teste, as *checklists* de revisão de projeto e de revisão de código, os registros de *log* de tempo, *log* de defeitos e o programa utilizável de acordo com os padrões de codificação.

Registro de defeitos: nessa atividade o programador deve registrar e revisar o resumo de plano de projeto para verificar se todos os defeitos encontrados, em cada fase, foram devidamente

registrados. Se algum defeito não registrado conseguir ser resgatado da memória do programador, pode ser registrado posteriormente.

Consistência dos dados de defeitos: o programador deve verificar os dados do *log* de registros de defeitos para assegurar se estão completos e precisos. Deve também verificar se o número de defeitos injetados e removidos, por fase, está razoável, não fugindo da realidade do projeto.

Tamanho: nessa atividade, o programa concluído deve ser devidamente contado e mensurado pela quantidade de linhas de código (LOC), considerando apenas o total de LOC do programa, independentemente da origem do código. Esses dados devem ser atualizados no resumo de plano de projeto.

Tempo: o programador deve revisar o *log* de registro de tempo (quadro 04) para verificar se todas as informações estão completas e informadas corretamente. Se algum dado não registrado conseguir ser resgatado da memória do programador, pode ser registrado posteriormente.

O Resumo de Plano de Projeto do SCRUM-PSP-IF (tabela 01) é um formulário traduzido e adaptado do material original do PSP2.1 (SEI, 2006). Contém os resumos (totais), compostos de três partes principais, o tamanho do programa, o tempo nas oito fases de construção e o resumo dos dados de defeitos injetados e removidos. Deve ser utilizado ao final da fase *postmortem* pessoal. Se for considerado que os *logs* de tempo e defeitos já estão com os insumos necessários para completar o formulário de Resumo de Projeto, o maior esforço do programador será em contar o tamanho do programa.

Tabela 01– Resumo de Plano de Projeto PSP2.1

Programador		Data	
Programa		Programa #	
Instrutor		Linguagem	

Tamanho do Programa	**Real**	**Até a Data**
Tamanho Total (T)	(Medido)	

Tempo na Fase (min.)	**Real**	**Até a Data**
Planejamento		
Projeto		
Revisão de Projeto		
Código		
Revisão do Código		
Compilação		
Teste		
Postmortem		
Total		

Defeitos Injetados	**Real**	**Até a Data**
Planejamento		
Projeto		
Revisão de Projeto		
Código		
Revisão do Código		
Compilação		
Teste		
Total		

Defeitos Removidos	**Real**	**Até a Data**
Planejamento		
Projeto		
Revisão de Projeto		
Código		
Revisão do Código		
Compilação		
Teste		
Total		

Fonte: traduzido e adaptado do material de auto estudo de SEI, 2006.

Registro da Qualidade. As informações de Resumo (*Summary*) do formulário original do PSP2.1 (SEI, 2006) proveem o preenchimento (ou cálculo) dos seguintes atributos: *Size/Hour*, *Planned Time*, *Actual Time*, *CPI* (*Cost-Performance Index*), *% Reuse*, *% New Reusable*, *Test Defects/KLOC or equivalent*, *Total Defects/KLOC or equivalent*, *Yield %*, *% Appraisal COQ*, *% Failure COQ*, *COQ A/F Ratio* e *PQI*.

O SCRUM-PSP-IF, segundo o plano de qualidade, presente na seção 3.4, irá utilizar alguns desses atributos para refletir os objetivos do modelo. Pra isso, irá separar esse Resumo do Plano de Projeto e criar um Resumo de Qualidade para o programa (iteração). Esse resumo inclui a seção das medidas do PQI, que no SCRUM-PSP-IF é representado por quatro medidas.

Após a realização dessas atividades, é possível ter como resultados do SCRUM-PSP-IF um programa completamente testado e em conformidade com os padrões de codificação, com os *templates*, com as *checklists*, e com os registros de tempo e defeitos.

Registro de dados

No PSP, os programadores usam dados para monitorar e melhorar o seu trabalho. Para isso, coletam dados sobre o tempo que gastam em cada fase do processo, sobre os tamanhos dos produtos que constroem e sobre a qualidade desses produtos, considerando determinadas taxas de defeitos aceitáveis. O SCRUM-PSP-IF segue essa mesma lógica, orientando, dentro das próprias atividades, a forma e a periodicidade que devem ser realizados os registros, os categorizando e contextualizando o seu uso.

Registro de tempo

Para medir o tempo gasto em cada fase, o programador utiliza o *log* de registro de tempo (quadro 04). Neste *log*, observa o tempo de início de uma determinada tarefa, o tempo de término da tarefa e o tempo de interrupção do trabalho. Cabe salientar que uma interrupção diz respeito a qualquer parada no andamento das tarefas, como um telefonema, um breve intervalo ou alguém interrompendo para fazer uma pergunta. Dessa forma se pode, realmente, acompanhar o tempo de trabalho com precisão, dimensionando de forma mais adequada o esforço gasto nas tarefas do projeto.

Registro de tamanho

Considerando que o tempo para desenvolver um produto é bastante influenciado pelo tamanho desse artefato, é importante medir o tamanho dos produtos desenvolvidos, para se

criar uma base histórica de medida e, através do PSP, ter condições de melhorar a qualidade do trabalho dos indivíduos e da equipe.

No PSP, antes do produto ser desenvolvido, os programadores estimam o tamanho do produto (com base no tamanho de programas desenvolvidos anteriormente) e quando o produto está pronto para ser entregue, os programadores medem o seu tamanho. Dessa forma são fornecidos os dados (estimados e reais) de tamanho do programa. Mesmo que essa pesquisa esteja focada apenas nos dados reais dos projetos, desconsiderando as atividades de estimativas, é importante salientar que a medida de tamanho deve estar correlacionada com o tempo de desenvolvimento do produto, para que os dados reais sejam úteis.

A principal unidade de medida do PSP, utilizada pelo SCRUM-PSP-IF, são as linhas de código. Porém, outras unidades de medida de tamanho podem ser usadas, como padrões de *interface*, determinadas funções ou procedimentos, desde que forneçam uma correlação razoável entre o tempo de desenvolvimento e o tamanho do produto. Permitir a medição automatizada do tamanho real do produto é altamente recomendável para qualquer unidade de medida candidata.

Registro de qualidade

O principal foco de qualidade do PSP está relacionado com a gerência dos defeitos. Dessa forma, os programadores precisam de dados sobre os defeitos injetados, as fases em que foram injetados, as fases em que os encontraram, a fase em que o defeito foi corrigido e quanto tempo foi necessário para corrigi-lo. Estes dados são registrados no *log* de registro de defeitos (quadro 05).

O termo defeito refere-se a algo que está errado com um programa ou qualquer coisa que prejudique a capacidade do programa de atender completa e efetivamente as necessidades do usuário (HUMPRHREY, 2000). Pode ser um erro ortográfico, um erro de pontuação ou uma instrução de programa incorreta. Os defeitos podem estar nos programas, em projetos e até mesmo nos requisitos ou outra documentação, ou seja, em qualquer fase de criação do sistema. O padrão de erros do PSP deve servir de suporte para categorização dos defeitos.

Com dados sobre o tamanho, tempo e defeito, existem muitas maneiras de coletar, medir, avaliar e gerenciar a qualidade de um programa. O PSP fornece um conjunto de medidas que

ajuda os programadores a analisar a qualidade de seus programas, sob várias perspectivas. Algumas medidas de qualidade PSP, detalhadas no plano de qualidade do SCRUM-PSP-IF, são a densidade de defeitos, taxa de revisão, relações de defeito, rendimento, relação entre avaliação e falha e, principalmente, o Índice de Qualidade de Processo (PQI).

Teste

Mesmo com a fase de teste unitário, realizado por cada programador, presente na sétima atividade da Iteração PSP, existe, ainda, a possibilidade de realizar testes de integração e testes de sistema, dependendo das necessidades do projeto e da estratégia de desenvolvimento selecionada. Portanto, esses testes são opcionais, guiados por necessidades específicas da iteração ou projeto.

Testes de integração devem ser realizados quando existe a necessidade de testar mais de um módulo do sistema, interligado com outros módulos, com outras aplicações ou com banco de dados. Já os testes de sistema têm o objetivo de validar e verificar as funcionalidades do sistema como um todo.

As principais atividades desta fase são:

- Seguir os *scripts* de Processo, Planejamento, Desenvolvimento, Revisão de Projeto, Revisão de Código e *Postmortem*.

- Seguir as instruções do Formulário do Resumo de Plano de Projeto e as instruções dos *templates*.

- Seguir os padrões de Tipos de defeitos, Codificação e Tipos de defeitos estendidos.

- Preencher os artefatos citados abaixo e mantê-los atualizados sempre que possível.

Os principais artefatos desta fase são:

- *Backlog* do Produto e *Backlog* da *Sprint*.

- *Log* de Registro de Tempo e *Log* de Registro de Defeitos.

- *Template* de Relatório de Teste, *Template* de Especificação Operacional e *Template* de Especificação Funcional.

- *Checklist* de Revisão de Projeto e *checklist* de Revisão de Código.

- Formulário de Resumo de Plano de Projeto.

Para monitorar o progresso do trabalho no caminho das metas, práticas do SCRUM, como *burndown* e *burnup* são sugeridas. Através desses gráficos é possível medir a velocidade média de entrega da equipe (e relacionar com o prazo determinado), além de permitir identificar fatores que possam influenciar na alteração da produtividade (tarefas muito complexas, extensas ou mal planejadas).

3.2.4 Fase 4: *Postmortem*

Na última fase do SCRUM-PSP-IF, assim como na primeira, o trabalho novamente é atribuído à equipe. Nesse momento é possível analisar e, futuramente, comparar as informações registradas nos projetos de desenvolvimento de *software*.

É nesse momento também que surgem grandes oportunidades de melhoria do processo, que podem ser implementadas já na próxima iteração ou no próximo projeto. As principais atividades na fase de *postmortem* de equipe são a demonstração do produto da iteração, a avaliação desse produto e a retrospectiva do processo, através da reunião de retrospectiva da *sprint* do *Scrum*. Segue a mesma lógica do modelo original (RONG et al., 2010).

Demonstração – Reunião de revisão da *sprint*

Pode ser considerada a principal atividade para validar módulos do produto potencialmente utilizáveis. Equiparada à reunião de revisão da *sprint* do *Scrum,* pois ambas são realizadas após a iteração e reúnem os representantes do cliente (principalmente o requisitante), um ou mais gestores do negócio (do mais alto escalão), além de toda a equipe do projeto.

Nesse cenário, a equipe reitera os objetivos da iteração, seu desempenho e o produto de trabalho alcançado nessa *sprint*. Através da demonstração das novas funcionalidades do sistema real, os resultados podem ser reconhecidos pelo solicitante. A motivação da equipe ganha outros patamares, pois a participação de membros da gestão da organização aumenta a relevância do trabalho.

Avaliação

Durante a reunião de demonstração, os representantes do cliente devem avaliar o produto de *software* que está sendo demonstrado. A equipe coleta os *feedbacks* e sugestões dos representantes para entender melhor as suas expectativas. Além disso, a presença de altos gestores faz a equipe assumir os compromissos com mais responsabilidade.

Retrospectiva

Após a reunião de demonstração do produto, a reunião de retrospectiva da iteração é então realizada por toda a equipe do projeto. Em uma típica reunião de retrospectiva da *sprint*, o líder coordena a discussão da equipe sobre os *status* do projeto e da qualidade, proporcionando visibilidade a áreas que necessitam de melhorias ou novos resultados.

Neste momento pode ser usado o PIP (*Process Improvement Proposal* – Proposta de Melhoria do Processo) do PSP (SEI, 2006), representado no quadro 8, que fornece uma maneira efetiva de registrar os pontos positivos e negativos do processo e indicar ideias de melhoria. Dessa forma, pode-se estabelecer prioridades entre as propostas de melhorias, descrevendo lições aprendidas e condições incomuns no projeto.

As principais atividades desta fase são:

- Realizar reunião de demonstração (revisão da *Sprint*).

- Avaliação do produto.

- Verificar o *status* da qualidade.

- Melhorar processo.

Os principais artefatos desta fase são:

- Produto potencialmente utilizável.

- Proposta de melhoria de processo (PIP).

3.3 Características do ambiente de aplicação do SCRUM-PSP-IF

O modelo iterativo SCRUM-PSP-IF pode ser considerado como um esforço para unir componentes de processos ágeis e orientados a planos, agregando práticas na busca por um considerável aumento na qualidade do desenvolvimento de *software*. A integração buscou, através da clara descrição dos métodos PSP, melhorar as condições de gerenciamento de defeitos de uma metodologia focada na entrega rápida de produtos e na adaptação as mudanças das necessidades de negocio e projeto.

Para caracterizar o ambiente de aplicação do SCRUM-PSP-IF foi utilizado o estudo de Boehm e Turner (2004), que propõe formas para equilibrar essa disputa entre métodos ágeis e disciplinados, considerando também as constatações de Rong et al. (2010) a respeito das características do projeto para aplicação do modelo instanciado.

3.3.1 Característica do projeto

SCRUM-PSP-IF adéqua-se, preferencialmente, a pequenos projetos de *software*, cujos requisitos não sejam todos conhecidos no inicio do projeto (ou tenham alta probabilidade de mudar durante a linha de tempo do projeto) e que os prazos sejam pouco negociáveis.

A equipe deve ter comportamento proativo, atuando com alto grau de produtividade e respondendo rapidamente a mudanças nas necessidades do produto, bem como ser auto dirigida, gerenciando com coerência curtos tempos de entrega, pois o desempenho pessoal de cada participante é fundamental para o sucesso da equipe.

3.3.2 Característica de gerenciamento

SCRUM-PSP-IF exige participação efetiva dos envolvidos por parte do cliente, de preferência, nomeando um solicitante principal (*Product Owner*). Dessa forma, pode-se ter uma declaração de requisitos efetiva e rica em detalhes, além da disponibilidade do solicitante do produto para reuniões de refinamento e alinhamento das necessidades do produto e validação das entregas.

O modelo também destaca a atuação de um treinador de equipe, uma pessoa que tenha experiência com modelos de qualidade de *software* e gerência de projetos. Esse papel deve ser assumido pelo *Scrum Master*, que irá treinar e acompanhar o trabalho da equipe, sendo o

moderador de conflitos e o solucionador de qualquer problema que impeça que a equipe alcance os objetivos das iterações.

O projeto de trabalho não é livre da escolha de estratégias e processos de desenvolvimento. Funciona melhor com estratégias iterativas ou incrementais de desenvolvimento de *software*. Além disso, o uso de ferramentas de *software* para automatizar as coletas de dados e acompanhar o andamento das tarefas é altamente recomendado.

3.3.3 Característica de tecnologia

SCRUM-PSP-IF é livre de tecnologias, não especifica as ferramentas para criar os requisitos, nem para construir as especificações de projetos. Também não prioriza nenhuma linguagem de programação ou tecnologia para implantar e integrar os módulos do sistema, bem como não sugere automatizações de cenários de testes.

3.3.4 Características dos membros da equipe

SCRUM-PSP-IF considera que toda a equipe deve, obrigatoriamente, receber treinamento básico em PSP e no modelo integrado. O *Scrum Master* pode se apossar desse conhecimento para ser o treinador do time de desenvolvimento. A equipe não deve possuir mais que seis (6) membros, localizados, preferencialmente, no mesmo local de trabalho (o que facilita reuniões diárias e reuniões de alinhamento, por exemplo). O treinamento não precisa ser completo, desde que contemple as atividades e artefatos utilizados pelo modelo instanciado.

Os participantes devem ter experiência de uso nas tecnologias envolvidas no projeto, incluindo linguagem de programação, banco de dados e linguagem de modelagem de dados, pois esses treinamentos não devem ser realizados no decorrer do projeto. Se necessário, o time de desenvolvimento deve ser preparado para as novas tecnologias antes do início do projeto.

Quanto ao conhecimento com processo de melhoria de desenvolvimento de *software*, considera-se que os participantes devam ter um bom nível de conhecimento sobre o assunto. Um perfil ainda mais adequado indica pessoas com domínio em processos de medição e tomada de decisões.

3.4 Considerações finais do capítulo

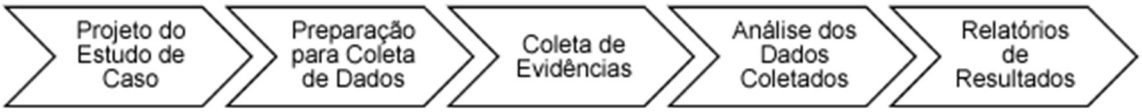
Este capítulo que abordou o modelo proposto, SCRUM-PSP-IF, instanciado do modelo original SCRUM-PSP (RONG et al., 2010). Inicialmente foi descrito o ciclo de vida para depois, detalhadamente, descrever o mecanismo de iteração do processo, composto pelas fases de Lançamento (ou Relançamento), Requisitos/Plano, Construção e Postmortem. Também descreveu quem são os responsáveis (equipe ou indivíduo) por cada atividade (dentro da fase) e quais são os materiais do PSP (formulários, templates e scripts) necessários para realizar essas atividades. Os materiais originais do PSP utilizados e, principalmente, as alterações necessárias para deixá-los mais enxutos (práticos) e coerentes com a lógica do SCRUM-PSP-IF, de fornecer uma estrutura unindo agilidade e disciplina, também foram devidamente detalhados nesse capítulo.

O SCRUM-PSP-IF buscou estabelecer um nível de confiabilidade aceitável, com base nos índices de qualidade, para gerenciar defeitos no desenvolvimento de software, através do uso de técnicas e métodos do PSP, com base em dados de tempo, tamanho e defeitos. Esses dados foram coletados por cada programador, individualmente, durante as iterações do desenvolvimento do produto. Ao final do capítulo foram abordadas as características (projeto, gerenciamento, tecnologia e membros da equipe) necessárias para a correta aplicação do SCRUM-PSP-IF em um ambiente real de desenvolvimento de software.

4 ESTUDO DE CASO – APLICAÇÃO DO MODELO INTEGRADO SCRUM-PSP-IF

A pesquisa de estudo de caso envolve várias etapas, incluindo o projeto, a realização do estudo de caso (coleta), a análise dos dados e o desenvolvimento das conclusões (YIN, 2002), sendo uma das metodologias mais utilizadas na pesquisa de sistemas de informação e engenharia de *software* (RUNESON e HOST, 2009). A figura 07 mostra o fluxo do processo do estudo de caso do presente trabalho.

Figura 07 – Processo para condução do estudo de caso.



Fonte: traduzido de RUNESON e HOST, 2009.

Os estudos de caso podem diferir com base em sua abordagem de pesquisa, período de tempo que é investigado (contemporâneo ou histórico), fontes de dados, unidade de análise, comprimento (período curto ou estudo longitudinal) e processo de pesquisa (RUNESON e HOST, 2009; YIN, 2002). Por isso é essencial que sejam explicados os motivos da seleção da metodologia de estudo de caso, pois existem várias maneiras de trabalhar com esse tipo de pesquisa.

Na pesquisa de estudo de caso, a escolha de uma unidade de análise apropriada, que no contexto desse trabalho pode ser um projeto, uma equipe de *software*, um participante individual, um tipo específico de trabalho ou a própria organização, é fundamental para garantir que a questão de pesquisa seja abordada adequadamente (EASTERBROOK et al., 2008).

A aplicação do modelo proposto, SCRUM-PSP-IF, foi realizada através de um estudo de caso exploratório, em um ambiente real de desenvolvimento de *software*, dentro de uma Instituição Pública de Ensino, entre janeiro e fevereiro de 2017. Os dados foram coletados através de arquivos, utilizando método de análise estatística para analisar esses dados, gerando os resultados (descobertas) da pesquisa.

O projeto, a realização e o relato do estudo de caso seguem algumas diretrizes propostas por Runeson e Host, em seu Guia de pesquisa de estudo de caso em engenharia de *software* (2009). Somados a isso estão os pontos de decisões da seção 1.4 que fornecem sucintas

orientações para a escolha de vários pontos de decisão de pesquisa, atuando como base para o detalhamento do estudo de caso e suas atividades (WHOLIN e AURUM, 2014). As atividades envolvem o projeto do estudo de caso, a coleta de dados, a análise dos dados coletados e os relatos de resultados.

4.1 Projeto do estudo de caso

Na metodologia de estudo de caso, o caso e as unidades de análise deverão ser escolhidas intencionalmente (RUNESON e HOST, 2009). A definição do projeto começou com escolha do Caso a ser estudado, representado por um único Projeto de Desenvolvimento de *Software*, na Reitoria do Instituto Federal Farroupilha, tendo como Unidade de Análise, dentro do caso, cada um dos três Engenheiros de *Software* que participaram do projeto. Segundo Yin (2002) esse estudo pode ser classificado como incorporado (embarcado), pois em um mesmo projeto de *software* (caso), participam mais de um indivíduo (unidade de análise).

4.1.1 Ambiente de aplicação do SCRUM-PSP-IF – IFFAR – Reitoria

Antes de falar sobre o caso a ser estudado, é necessário contextualizar o ambiente de aplicação do SCRUM-PSP-IF e a Diretoria de Tecnologia da Informação (DTI), especialmente a Coordenadoria de Análise e Desenvolvimento de Sistemas (CADS).

O Instituto Federal Farroupilha (IFFAR) não se caracteriza como uma "fábrica de *software*", mas sim como um Instituto Federal de Educação, cuja missão é "Promover a educação profissional, científica e tecnológica por meio do ensino, pesquisa e extensão. O foco da instituição é a formação de cidadãos críticos, autônomos e empreendedores, comprometidos com o desenvolvimento sustentável". É composto por onze *campi* e a Reitoria, em Santa Maria/RS.

A DTI, ligada à Pró-Reitoria de Desenvolvimento Institucional, é o segmento responsável pela gestão da área de Tecnologia da Informação e Comunicação do IFFAR. Cabe a ela prestar apoio técnico, administrativo e operacional, fornecendo suporte de *hardware*, *software* e serviços de TI. Está subdividida, em três áreas (coordenadorias): Suporte e Manutenção; Rede, Segurança e Conectividade; Análise e Desenvolvimento de Sistemas. A mesma lógica de estruturação das áreas se aplica aos *campi*.

Na CADS, as tarefas são realizadas pelos próprios servidores, sendo dois Analistas de TI e dois Técnicos em TI, na reitoria. O total de servidores de TI em toda a instituição ultrapassa os cinquenta, somados os cargos. Aproximadamente um terço desses trabalha com desenvolvimento de *software*. Nem todas as equipes, das unidades, utilizam um processo de desenvolvimento, ao contrário da Reitoria e do *campus* São Vicente do Sul, por exemplo. Esses padrões foram formalizados para serem utilizados com o método ágil *Scrum*.

4.1.2 Caso – Sistema integrado de processos seletivos

O Caso, um projeto de software para integrar os processos seletivos, está de acordo com as características de aplicação do SCRUM-PSP-IF, presente na seção 3.3.

Características do Projeto – o sistema se caracteriza como um pequeno projeto de *software*, composto por alguns módulos. No início foi possível conhecer apenas os principais requisitos do *software* (com base nos sistemas legados), porém, algumas necessidades estão diretamente ligadas a editais ou normas governamentais e não puderam ser conhecidas inicialmente, tendo grande probabilidade de sofrer alterações no decorrer do projeto. O prazo para alguns módulos entrarem em produção é pouco negociável, portanto, a equipe deve ter alta produtividade e, ao mesmo tempo, responder rapidamente às mudanças nos requisitos do produto.

Características de Gerenciamento – os participantes do projeto, tanto a equipe técnica quanto os principais usuários, participaram ativamente do processo. Ressalta-se a importante participação do *Scrum Master*, que além de treinar os participantes, tem a função de gerenciar o projeto e solucionar os problemas encontrados. O projeto utilizou estratégias iterativas de desenvolvimento de *software*.

Características de Tecnologia – As tecnologias utilizadas tem como base o Processo de Desenvolvimento de *Software* (PDS) do IFFAR, que estabelece um processo básico de engenharia de *software*, baseado nos artefatos e eventos do *Scrum* e se aplica à Reitoria e a todas as suas unidades, desde que utilizem bases tecnológicas equivalentes. A meta do PDS é garantir a produção de sistemas de qualidade, que atendam aos requisitos e necessidades dos usuários. O processo é iterativo e dividido em 4 partes principais: abertura (oficialização da demanda, reunião com os interessados, formação da equipe e abertura do projeto), execução

(planejamento e criação dos *backlogs* do produto e da *sprint* e codificação), implementação (homologação e implantação) e manutenção do projeto (treinamento e melhorias).
As tecnologias sugeridas no PDS são típicas de um ambiente de desenvolvimento PHP:

- *Sublime Text* ou *PHP Storm* (Ambientes de Desenvolvimento Integrado PHP);
- *Bootstrap* (*Framework* de *Interface Web*);
- *FileZilla* (Gerenciador de arquivos *FTP* e *SFTP*);
- *Data Grip* (acesso ao Banco de Dados);
- *MySQL WorkBench* (modelagem de Banco de Dados);
- *Astah Community* (ferramenta para diagramação UML);
- *Pencil* 3.0 ou superior (ferramenta para prototipação de telas).

O PDS institucional não foi utilizado juntamente com o novo processo sugerido, mas serviu como base para padronizar as ferramentas tecnológicas utilizadas nos projetos de *software* e como utiliza *Scrum*, se fosse utilizado, não iria interferir na aplicação do SCRUM-PSP-IF, pois os artefatos e atividades do *Scrum* são as mesmas, em ambos os processos.

Características do sistema – o sistema tem o objetivo de fornecer uma visão única das seleções da Instituição, tanto de alunos quanto de servidores (técnicos administrativos e docentes). Os usuários do sistema serão os candidatos, administradores do sistema e alguns servidores que acompanham os processos seletivos, através de relatórios. Portanto, devem existir três perfis distintos de acessos. Dentre as funcionalidades que devem ser criadas ou alteradas estão: Manter Instituição, manter Processo Seletivo, manter Cursos, manter Inscrições, manter Pagamentos, manter Isenção, manter Homologação, manter Ensalamento (alocar devidamente os alunos nas respectivas salas), manter Correção, manter Classificação, manter Usuários, manter Logs e manter Relatórios.

Atualmente, existem muitos tipos de processos seletivos na Instituição, englobando cursos presenciais, à distância, seleção de professores, de pesquisadores, totalizando dezoito independentes.

Características dos membros da equipe – a equipe *Scrum* foi composta por três programadores e um líder de projeto, além de alguns usuários dos sistemas. Todos atuam no mesmo local de

trabalho. O líder de projeto e, principalmente, os programadores receberam treinamento em PSP e SCRUM-PSP-IF. O treinamento em PSP não foi completo, pois o objetivo dessa tarefa foi de contemplar as atividades e artefatos utilizados pelo modelo instanciado. Os membros da equipe não apresentaram dificuldades com as tecnologias envolvidas no projeto, mostrando também um pouco de conhecimento em processo de melhoria de desenvolvimento de *software*.

4.1.3 Unidades de análise – Engenheiros de *software*

As unidades de análise desse estudo de caso consistem de três engenheiros de *software*, que no decorrer do trabalho serão chamados de Programadores A, B e C. Considerando os papéis de uma equipe *Scrum*, ainda existe o *Scrum Master* e alguns solicitantes do produto (clientes). Porém, apenas os programadores serão as unidades de análise dentro do projeto, pois o SCRUM-PSP-IF tem o objetivo de melhorar a capacidade individual dos programadores em desenvolver *software*. Os dados gerados no contexto da equipe não serão analisados, mesmo sendo importantes para a aplicação do processo.

Os três programadores possuem nível superior e pós-graduação na área de TI (mestrado, no caso do programador A), com mais de 5 anos de experiência em programação (principalmente na linguagem PHP) e fazem uso de métodos ágeis (*Scrum*). Todos têm conhecimento nas tecnologias utilizadas para desenvolver o sistema. O programador A e B são Técnicos de TI, enquanto o programador C é Analista de TI.

4.2 Coleta de dados

O método de pesquisa de arquivo se refere a dados históricos como, por exemplo, o número de defeitos de um *software*, o tempo gasto nas fases do desenvolvimento ou os documentos produzidos no projeto de um sistema, arquivados por alguém ou por organizações. Podem exigir permissão para acesso ou ser publicamente acessíveis.

Este método envolve a localização dos dados, coleta sistemática dos dados, análise e interpretação desses dados. (RUNESON e HOST, 2009). Não serão feitas distinções entre dados de arquivos e documentações, como alguns autores sugerem, mas sim entre dados qualitativos e quantitativos.

Basicamente, cada programador, em cada iteração, preencheu um conjunto de arquivos (planilhas eletrônicas e formulários de texto), armazenando três medidas principais: o tempo gasto para realizar cada fase do desenvolvimento de *software*, os defeitos inseridos e removidos (incluindo o tempo gasto para essas atividades) e o tamanho total, em linhas de código, dos programas construídos. Além disso, também foram registrado os dados de qualidade do processo.

Para guiar a coleta dos dados, o modelo SCRUM-PSP-IF oferece um roteiro ordenado para o uso dos arquivos, com as fórmulas das métricas já incluídas nas planilhas. As atividades de equipe também foram armazenadas em arquivos, pois servem de insumo para a construção dos módulos. Porém, não fazem parte do escopo da análise e dos resultados.

Salienta-se que os testes de integração, necessários ao final de cada iteração, foram realizados pela equipe, porém não foram registrados nos documentos (até porque esses não apresentam tal *template* ou formulário).

O conjunto de arquivos utilizados pelos programadores foi coletado de acordo com o número de iterações do projeto. Para três iterações, com três programadores, foram necessários seis conjuntos de arquivos. Comparados aos 30 materiais (aproximadamente) do PSP de nível 2.1, mesmo subtraindo os *scripts* (guias), que também foram utilizados pelo SCRUM-PSP-IF, são apenas 25% dos materiais utilizados no novo modelo. Isso mostra, claramente, que o processo proposto não pode ser considerado pesado, seguindo a filosofia dos métodos ágeis, como o *Scrum*.

Todos os arquivos não incluídos nos apêndices, além dos incluídos, estão presentes em um repositório *online*, no *Redmine* (gerenciador de projetos) do IFFAR, no endereço eletrônico http://redmine.iffarroupilha.edu.br/redmine/projects/case_norton/files. No entanto, devido às políticas de segurança da Direção de TI, esse repositório só pode ser acessado através de uma senha.

4.2.1 Arquivos coletados pela equipe

A equipe ágil foi responsável por criar o documento de Lançamento/Relançamentos, o conjunto de artefatos *Scrum* (*Backlog* do produto e *Backlogs* das *Sprints*) e participar das constantes reuniões de projeto (reunião diária, reunião de planejamento das *Sprints* e reunião de retrospectiva das *Sprints*, ao final das iterações).

Os proprietários do produto (usuários) foram os principais responsáveis por manter o *backlog* do produto, que contém os requisitos, atualizados e condizentes com as necessidades do produto. A equipe de desenvolvimento e o líder do projeto também colaboraram para manter esse artefato. O time de desenvolvimento foi o responsável por criar os documentos de *Backlog* das *Sprints*, contendo as tarefas a serem realizadas em uma determinada janela de tempo (nas quatro semanas do projeto). Nesse trabalho foram desenvolvidas três *sprint*, duas delas de uma semana de duração e a terceira com duas semanas de duração.

4.2.2 Arquivos coletados pelos programadores

Os arquivos coletados por cada engenheiro de *software* consistem de um conjunto de artefatos do PSP (alguns adaptados), solicitados pelo SCRUM-PSP-IF. Como cada iteração exige o mesmo conjunto de *logs*, *templates*, *checklists* e resumos, nas três iterações foram gerados cinquenta e quatro arquivos (planilhas eletrônicas e documentos de texto), alguns presentes nos apêndices e os demais presentes no repositório do estudo de caso. Eles serão devidamente referenciados no decorrer do trabalho.

Iteração 1

A iteração 1 foi realizada entre 30 de janeiro de 2017 à 03 de fevereiro de 2017, conforme documento de *Backlog da Sprint* 1, produzido pelo time de desenvolvimento. O primeiro *backlog* da *sprint* reuniu alguns requisitos funcionais, retirados do documento de *Backlog* do Produto, incluindo: manter instituição, manter processo seletivo e manter usuário. Foram criadas as classes e as telas principais dessas funcionalidades.

Os arquivos preenchidos pelo Programador A, na iteração 1, contemplam:

1. *Logs* de Tempo e Defeitos e Resumo de Projeto - I1P1 - Manter Inst (APÊNDICES 01, 02 e 03);

2. *Template* de Especificação Operacional - I1P1 - Manter Inst;

3. *Template* de Especificação Funcional - I1P1 - Manter Inst;

4. *Checklist* de Revisão de Projeto - I1P1 - Manter Inst;

5. *Checklist* de Revisão de Código - I1P1 - Manter Inst;

6. *Template* de Relatório de Teste - I1P1 - Manter Inst.

Os arquivos preenchidos pelo Programador B, na iteração 1, contemplam:

1. *Logs* de Tempo e Defeitos e Resumo de Projeto - I1P2 - Manter Proc_Sel (APÊNDICES 04, 05 e 06);

2. *Template* de Especificação Operacional - I1P2 - Manter Proc_Sel;

3. *Template* de Especificação Funcional - I1P2 - Manter Proc_Sel;

4. *Checklist* de Revisão de Projeto- I1P2 - Manter Proc_Sel;

5. *Checklist* de Revisão de Código- I1P2 - Manter Proc_Sel;

6. *Template* de Relatório de Teste- I1P2 - Manter Proc_Sel;

Os arquivos preenchidos pelo Programador C, na iteração 1, contemplam:

1. *Logs* de Tempo e Defeitos e Resumo de Projeto - I1P3 - Manter User (APÊNDICES 07, 08 e 09);

2. *Template* de Especificação Operacional - I1P3 - Manter User;

3. *Template* de Especificação Funcional - I1P3 - Manter User;

4. *Checklist* de Revisão de Projeto- I1P3 - Manter User;

5. *Checklist* de Revisão de Código- I1P3 - Manter User;

6. *Template* de Relatório de Teste- I1P3 - Manter User.

Fazendo uma breve análise da primeira iteração, pode-se dizer que, na especificação de requisitos, foi a menos problemática entre as três, pois apenas as necessidades mais bem especificadas foram selecionadas para serem codificadas (cadastros básicos). Como já se tinha

experiência sobre essas funcionalidades básicas (e suas dependências) nos sistemas legados, uma porcentagem do código fonte pôde ser reutilizada para a construção das classes (lembrando o SCRUM-PSP-IF solicita apenas o registro do total de linhas de código, conforme explicado no capítulo três). O processo de trabalho seguiu seu fluxo normal, do início ao fim da iteração, com reuniões diárias e interação com os principais *stakeholders*. A maior dificuldade, segundo os programadores, foi no preenchimento dos documentos do SCRUM-PSP-IF, devido a pouca experiência dos programadores nessa atividade. Portanto, o tempo para preenchimento dos formulários e *templates* foi alto, comparado com as outras iterações. A quantidade de erros detectados foi relevante, porém a remoção dos erros não se mostrou muito eficiente. Os testes foram realizados sem maiores dificuldades.

Iteração 2

A iteração 2 foi realizada entre 06 de fevereiro de 2017 à 10 de fevereiro de 2017, conforme documento de *Backlog* da *Sprint* 2. O segundo *backlog* da *sprint* reuniu alguns requisitos funcionais, retirados do documento de *Backlog* do Produto, incluindo: manter curso, manter componente e manter *log*. Foram criadas as classes e telas principais para manter essas funcionalidades.

Os arquivos preenchidos pelo Programador A, na iteração 2, contemplam:

1. *Logs* de Tempo e Defeitos e Resumo de Projeto - I2P1 - Manter Curso (APÊNDICES 10, 11 e 12);

2. *Template* de Especificação Operacional - I2P1 - Manter Curso;

3. *Template* de Especificação Funcional - I2P1 - Manter Curso;

4. *Checklist* de Revisão de Projeto - I2P1 - Manter Curso;

5. *Checklist* de Revisão de Código - I2P1 - Manter Curso;

6. *Template* de Relatório de Teste - I2P1 - Manter Curso.

Os arquivos preenchidos pelo Programador B, na iteração 2, contemplam:

1. *Logs* de Tempo e Defeitos e Resumo de Projeto - I2P2 - Manter Comp (APÊNDICES 13, 14 e 15);

2. *Template* de Especificação Operacional - I2P2 - Manter Comp;

3. *Template* de Especificação Funcional- I2P2 - Manter Comp;

4. *Checklist* de Revisão de Projeto- I2P2 - Manter Comp;

5. *Checklist* de Revisão de Código- I2P2 - Manter Comp;

6. *Template* de Relatório de Teste- I2P2 - Manter Comp.

Os arquivos preenchidos pelo Programador C, na iteração 2, contemplam:

1. *Logs* de Tempo e Defeitos e Resumo de Projeto - I2P3 - Manter Log (APÊNDICES 16, 17 e 18);

2. *Template* de Especificação Operacional - I2P3 - Manter *Log*;

3. *Template* de Especificação Funcional - I2P3 - Manter *Log*;

4. *Checklist* de Revisão de Projeto- I2P3 - Manter *Log*;

5. *Checklist* de Revisão de Código- I2P3 - Manter *Log*;

6. *Template* de Relatório de Teste- I2P3 - Manter *Log*.

Quanto à análise da segunda iteração, pode-se dizer que esta teve praticamente o mesmo nível de complexidade da primeira iteração, porém os requisitos não estavam tão claros e necessitaram ser refinados. Assim como a primeira, essa iteração também utilizou uma porcentagem do código fonte dos sistemas legados para a construção das novas classes. O processo de trabalho seguiu seu fluxo normal, tendo apresentado algumas dificuldades nos testes de integração (SCRUM-PSP-IF registra apenas os testes unitários, realizados por cada programador, conforme capítulo três). Provavelmente, o fato do novo modelo não ter um *template* específico para esse tipo de teste pode ter contribuído para tais dificuldades. O tempo

para preenchimento dos formulários e *templates* foi mais baixo do que o da primeira iteração. A quantidade de erros detectados e removidos foi maior do que na primeira iteração.

Iteração 3

A iteração 3 foi realizada entre 13 de fevereiro de 2017 à 24 de fevereiro de 2017, conforme documento de *Backlog* da *Sprint* 3 (produzido pelo time de desenvolvimento). O terceiro *backlog* da *sprint* reuniu alguns requisitos funcionais, retirados do documento de *Backlog* do Produto, incluindo: manter processo seletivo de professores substitutos (visão administrativa), visão do candidato, visão do usuário de relatórios, manter pagamento, manter homologação e manter inscrição do candidato. Foram criadas também as funcionalidades de consultar pagamento, confirmar pagamento, gerar homologação, incluindo os relatórios da visão administrativa.

Os arquivos preenchidos pelo Programador A, na iteração 3, contemplam:

1. *Logs* de Tempo e Defeitos e Resumo de Projeto - I3P1 - Visão Admin (APÊNDICES 23, 24 e 25);

2. *Template* de Especificação Operacional - I3P1 - Visão Admin (APÊNDICE T);

3. *Template* de Especificação Funcional - I3P1 - Visão Admin (APÊNDICES 20 e 21);

4. *Checklist* de Revisão de Projeto - I3P1 - Visão Admin;

5. *Checklist* de Revisão de Código - I3P1 - Visão Admin;

6. *Template* de Relatório de Teste - I3P1 - Visão Admin (APÊNDICE X).

Os arquivos preenchidos pelo Programador B, na iteração 3, contemplam:

1. *Logs* de Tempo e Defeitos e Resumo de Projeto - I3P2 - Visão Candidato (APÊNDICES 29, 30 e 31);

2. *Template* de Especificação Operacional - I3P2 - Visão Candidato (APÊNDICE AC);

3. *Template* de Especificação Funcional - I3P2 - Visão Candidato (APÊNDICE AD);

4. *Checklist* de Revisão de Projeto- I3P2 - Visão Candidato;

5. *Checklist* de Revisão de Código- I3P2 - Visão Candidato;

6. *Template* de Relatório de Teste- I3P2 - Visão Candidato (APÊNDICE AE).

Os arquivos preenchidos pelo Programador C, na iteração 3, contemplam:

1. *Logs* de Tempo e Defeitos e Resumo de Projeto - I3P3 - Visão usuário relatórios (APÊNDICES 35, 36 e 37);

2. *Template* de Especificação Operacional - I3P3 - Visão usuário relatórios (APÊNDICE AI);

3. *Template* de Especificação Funcional- I3P3 - Visão usuário relatórios (APÊNDICE AJ);

4. *Checklist* de Revisão de Projeto - I3P3 - Visão usuário relatórios;

5. *Checklist* de Revisão de Código - I3P3 - Visão usuário relatórios;

6. *Template* de Relatório de Teste - I3P3 - Visão usuário relatórios (APÊNDICE AL).

Analisando a terceira iteração, certamente foi a mais difícil de ser realizada, pois, além de contar com maior número de requisitos, previu funcionalidades de maior complexidade, bem como aspectos de *layout* do sistema. A divisão do trabalho teve que ser revista, pois não estava devidamente balanceada entre os programadores (dois tinham muito trabalho e um nem tanto). O processo de trabalho, apesar de seguir o fluxo normal, também apresentou dificuldades, segundo os programadores, nos testes de integração. O tempo para preenchimento dos formulários e *templates* foi mais baixo, comparado com a primeira e a segunda iteração. A quantidade de erros detectados foi extremamente relevante, bem como a remoção dos erros, que se mostrou muito eficiente, se comparada com as demais iterações. Como principal lição aprendida, fica o fato de ter alocado pouco trabalho nas duas primeiras iterações, comparadas com a terceira, portanto, as entregas correram alto risco de não serem concluídas no prazo, com altas possibilidades de atrasar a entrega do produto final de *software*, pois novas iterações teriam que ser geradas.

4.3 Análise dos dados

Wohlin e Aurum (2015) argumentam que a análise de dados deve ser conduzida de forma distinta para dados qualitativos e quantitativos. Processos de pesquisa quantitativa utilizam técnicas como análise estatística descritiva ou análise de correlação para compreender os dados coletados. Já processos de pesquisa qualitativa utilizam técnicas de geração ou confirmação de hipóteses para extrair conclusões dos dados. Segundo Wohlin e Aurum (2015), análise temática e análise hermenêutica são técnicas qualitativas, enquanto análise estatística é classificada como técnica quantitativa de análise de dados. Já a teoria fundamentada pode ser considerada uma técnica tanto qualitativa quanto quantitativa.

Nesse trabalho, com a utilização do método de pesquisa estudo de caso, muitos dados foram gerados e armazenados, em arquivos. Para analisar esses dados foram necessários métodos quantitativos (estatísticos). Independente do método, a preparação dos dados é um passo importante para a análise (RUNESON e HOST, 2009).

Quanto ao método estatístico, os dados podem ser coletados de forma quantitativa, onde estatísticas são aplicadas, diretamente, para analisar os dados. Alternativamente, os dados podem ser coletados de forma qualitativa e depois convertidos para a forma quantitativa, usando a frequência de palavras, eventos, pessoas, temas, por exemplo, e, em seguida, estatísticas devem ser aplicadas para analisar esses dados. Os dados podem ser analisados usando análise estatística descritiva ou análise estatística inferencial (WOHLIN e AURUM, 2015).

A análise descritiva envolve a síntese de dados, descrevendo e agregando esses dados e apresentando associação entre eles. A média, a mediana, o modo, o desvio e a variância são exemplos de métodos utilizados na análise descritiva, bem como diferentes tipos de gráficos. A análise inferencial envolve, por exemplo, testes estatísticos de hipóteses, análise de regressão e estimativa usando técnicas de mineração de dados. O teste de hipótese é usado para fazer inferências sobre uma população. A análise de regressão refere-se a métodos que ajudam a entender como mudanças em uma variável afetam outra variável (WOHLIN e AURUM, 2015).

Nessa pesquisa será utilizada a análise estatística descritiva básica para associar alguns dados, realizar médias e comparar algumas variáveis, complementar às métricas (medidas) do próprio modelo SCRUM-PSP-IF (*PQI*, *A/FR*, *Yield*, *Defect Density* e *Review Rate*), baseadas no PSP, que realizam a maioria dos cálculos necessários para o relato dos resultados,

concentrando-se nas variáveis globais de defeitos, tempo e tamanho. Essas três medidas servirão de base para os cálculos dos índices de qualidade e, consequentemente, dos resultados da pesquisa.

4.3.1 Preparação dos dados

A preparação dos dados tem o objetivo de conferir a consistência dos dados coletados, se certificando que cada programador preencheu todas as informações que o processo integrado SCRUM-PSP-IF exigiu. Para isso, foi elaborada uma lista de verificação (quadro 11), com base nas necessidades de preenchimento (e corretude) dos documentos do modelo.

Quadro 11 – *Checklist* de consistência de dados.

Número	Descrição
1	O tempo gasto nas oito fases do SCRUM-PSP-IF, preenchido no *Log* de Registro de Tempo, está de acordo com os dados preenchidos no Resumo de Plano de Projeto.
2	Os valores para os defeitos injetados e removidos são equivalentes no Resumo de Plano de Projeto.
3	Os valores para defeitos injetados e removidos estão de acordo com o número de defeitos do *Log* de Registro de Defeitos.
4	As fases dos erros injetados e removidos no *Log* de Registro de Defeitos estão de acordo com o número de os erros inseridos e removidos em cada fase do Resumo de Plano de Projeto.
5	A coluna *Fix Defect* foi usada corretamente, para indicar qual erro se estava tentando corrigir quando o novo erro foi inserido.
6	Os valores da coluna Tempo Delta, do *Log* de Registro de Tempo, foram calculados corretamente.
7	O resumo dos dados dos *Logs* de Tempo e Defeitos foi devidamente repassado para o Resumo de Plano de Projeto.
8	Os dois *templates* de projeto estão completos e consistentes.
9	As duas *cheklists* (revisão de projeto e revisão de código) foram devidamente utilizadas.
10	O *template* de teste está completo e consistente.
11	O total de LOC foi preenchido na coluna de tamanho do programa do Resumo de Plano de Projeto.
12	Os dados de qualidade do PQI foram preenchidos no Resumo de Plano de Projeto.

Fonte: elaborado pelo autor.

A *checklist* inclui a verificação de consistência no preenchimento do tempo gasto nas oito fases, bem como os valores de defeitos injetados e removidos, para que essas informações estejam iguais no formulário de resumo e nos respectivos *logs*. O preenchimento dos *templates* de projeto, de testes e *checklists* de revisões de projeto e código também são verificados. Por fim, a verificação se deu no preenchimento do total de linhas de código produzidas e nos dados de qualidade preenchidos no resumo de plano de projeto, conferindo se os cálculos dos índices estão sendo realizados com os dados corretos.

A lista de verificação foi aplicada no conjunto de documentos das três iterações do projeto. Foram constatadas algumas inconsistências no preenchimento dos dados, exibidas nos quadros 12, 13 e 14, considerando as iterações, os programadores e os itens da lista. O "X" representa que o item está correto. Os números, de 1 a 12, representam os 12 itens da *checklist* de consistência dos dados (quadro 11). Após as constatações dos erros de preenchimento, as devidas correções foram realizadas, acompanhadas pelos programadores.

Quadro 12 – Resultados da verificação de consistência de dados – Iteração 1.

Programador	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
A	X	X	X	X	X		X	X	X	X	X	X
B	X	X	X	X	X	X	X	X	X	X	X	
C	X	X	X	X	X	X	X	X	X	X	X	X

Fonte: elaborado pelo autor.

No quadro 12, na iteração 1, o programador A não calculou corretamente o campo delta, no registro de tempo do *log* de tempo (APÊNDICE A), enquanto o programador B não informou adequadamente todos os dados de qualidade para o PQI (APÊNDICE G).

Quadro 13 – Resultados da verificação de consistência de dados – Iteração 2.

Programador	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
A		X	X	X	X	X	X	X	X	X	X	X
B	X	X	X	X	X	X	X	X	X	X	X	X
C	X	X			X	X	X	X	X	X	X	X

Fonte: elaborado pelo autor.

Na iteração 2, do quadro 13, o preenchimento de dados do programador A apresentou diferença entre o *log* de registro de tempo (APÊNDICE J) e o resumo de plano de projeto (APÊNDICE M). Enquanto que o programador C teve incoerência entre os valores e fases dos defeitos removidos, entre o log de defeitos (APÊNDICE O) e o resumo do projeto (APÊNDICE P).

Quadro 14 – Resultados da verificação de consistência de dados – Iteração 3.

Programador	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
A	X	X	X	X	X	X	X	X	X	X	X	X
B	X	X	X	X	X	X	X	X	X	X	X	X
C	X	X	X	X	X	X	X	X	X	X	X	X

Fonte: elaborado pelo autor.

No quadro 14, que representa a verificação da iteração 3, nenhuma inconsistência foi encontrada.

4.3.2 Análise dos dados de defeitos

Para analisar os dados de defeitos foram consideradas seis categorias: defeitos injetados por fase, defeitos removidos por fase, total de defeitos injetados, total de defeitos removidos, defeitos injetados acumulados e defeitos removidos acumulados (quadro 15). Os dois últimos consideram a soma dos defeitos das três iterações, de cada programador. Esses dados foram retirados dos formulários de Resumo de Plano de Projeto (APÊNDICES 03, 06, 09, 12, 15, 18, 25, 31, 37).

Quadro 15 – Defeitos – Iteração 1 – Programador A.

Fase	**Defeitos injetados**	**Defeitos removidos**
Planejamento	0	0
Projeto	6	0
Revisão do Projeto	0	2
Código	8	0
Revisão do Código	0	6
Compilação	0	0
Teste	0	6
Total da Iteração	14	14
Acumulado	14	14

Fonte: elaborado pelo autor.

O quadro 15 mostra o resumo dos defeitos da primeira iteração, do programador A. Os dados mostram que a revisão do código foi mais eficiente que a revisão do projeto, pois enquanto a primeira removeu 43% do total de defeitos da iteração, a segunda removeu 14%, restando 43% dos defeitos para os testes. Se forem consideradas apenas as remoções de defeitos injetados nas respectivas fases, a fase de revisão de projeto teve eficiência de 33% e a fase de revisão de código teve eficiência de 75%. Obviamente, na fase de revisão de código se tem a chance de corrigir erros que não foram corrigidos na fase de projeto, mas o contrário não se mostra possível.

O quadro 16 mostra o resumo dos defeitos da segunda iteração (de uma semana), do programador A. Esses dados mostram que as revisões de projeto e código foram eficientes, ambas removendo 38% dos defeitos, restando 24% dos defeitos para os testes. Se forem consideradas apenas as remoções de defeitos injetados nas respectivas fases, a fase de revisão de projeto teve eficiência de 83% e a fase de revisão de código teve eficiência de 71%.

Quadro 16 – Defeitos – Iteração 2 – Programador A.

Fase	Defeitos injetados	Defeitos removidos
Planejamento	0	0
Projeto	6	0
Revisão do Projeto	0	5
Código	7	0
Revisão do Código	0	5
Compilação	0	0
Teste	0	3
Total da Iteração	13	13
Acumulado	27	27

Fonte: elaborado pelo autor.

O quadro 17 mostra o resumo dos defeitos da terceira iteração, do programador A. Esses dados mostram que tanto a revisão do projeto quanto a revisão código foram eficientes, removendo 37% e 42% dos defeitos, respectivamente, restando apenas 21% dos defeitos para os testes. Se forem consideradas apenas as remoções de defeitos injetados nas respectivas fases, a fase de revisão de projeto teve eficiência de 87% e a fase de revisão de código teve eficiência de 73%.

Quadro 17 – Defeitos – Iteração 3 – Programador A.

Fase	Defeitos injetados	Defeitos removidos
Planejamento	0	0
Projeto	8	0
Revisão do Projeto	0	7
Código	11	0
Revisão do Código	0	8
Compilação	0	0
Teste	0	4
Total da Iteração	19	19
Acumulado	46	46

Fonte: elaborado pelo autor

Analisando e comparando a injeção e remoção de defeitos, nas três iterações, do programador A (quadros 15, 16 e 17), é possível verificar gradativas melhorias na revisão do projeto, nas iterações do projeto. Inclusive, dobrando seu desempenho da primeira para a segunda iteração (de 14% para 38%). Isso se deve, principalmente, à efetividade no uso da *checklist* de revisão de projeto, acompanhada do correto registro dos defeitos encontrados e removidos. A revisão do código se manteve em um bom nível em todas as iterações, mostrando que a *checklist* de revisão dessa fase foi utilizada com eficiência, em todas as iterações. O aumento do número de defeitos removidos nas fases de projeto e codificação diminuiu consideravelmente os esforços (e tempo) para encontrar e remover esses defeitos na fase de testes.

O quadro 18 mostra o resumo dos defeitos da primeira iteração, do programador B, onde a revisão do código foi mais eficiente que a revisão do projeto, pois a primeira removeu 42% dos defeitos e a segunda removeu apenas 8%, restando 50% dos defeitos para os testes. Considerando apenas remoções de defeitos injetados nas respectivas fases, a fase de revisão de projeto teve eficiência de 33% e a fase de revisão de código teve eficiência de 55%.

Quadro 18 – Defeitos – Iteração 1 – Programador B.

Fase	Defeitos injetados	Defeitos removidos
Planejamento	0	0
Projeto	3	0
Revisão do Projeto	0	1
Código	9	0
Revisão do Código	0	5
Compilação	0	0
Teste	0	6
Total da Iteração	12	12
Acumulado	12	12

Fonte: elaborado pelo autor.

O quadro 19 mostra o resumo dos defeitos da segunda iteração, do programador B, onde a revisão do código foi menos eficiente que a revisão do projeto, pois enquanto a primeira removeu apenas 19% dos defeitos, a segunda removeu 45%, restando 36% dos defeitos para os testes. Se forem consideradas apenas as remoções de defeitos injetados nas respectivas fases, a fase de revisão de projeto teve eficiência de 83% e a fase de revisão de código teve eficiência de 40%.

Quadro 19 – Defeitos – Iteração 2 – Programador B.

Fase	Defeitos injetados	Defeitos removidos
Planejamento	0	0
Projeto	6	0
Revisão do Projeto	0	5
Código	5	0
Revisão do Código	0	2
Compilação	0	0
Teste	0	4
Total da Iteração	11	11
Acumulado	23	23

Fonte: elaborado pelo autor.

O quadro 20 mostra o resumo dos defeitos da terceira iteração, do programador B, onde a revisão do código foi mais eficiente que a revisão do projeto, pois enquanto a primeira removeu 55% dos defeitos, a segunda removeu 20%, restando 15% dos defeitos para os testes. Salienta-se que na fase de codificação foram removidos 2 defeitos (injetados na fase de projeto), o que representa 10% do total de defeitos. Considerando as remoções dos defeitos injetados nas respectivas fases, a fase de revisão de projeto teve eficiência de 85% e a fase de revisão de código teve eficiência de 57%.

Quadro 20 – Defeitos – Iteração 3 – Programador B.

Fase	Defeitos injetados	Defeitos removidos
Planejamento	0	0
Projeto	13	0
Revisão do Projeto	0	11
Código	7	2
Revisão do Código	0	4
Compilação	0	0
Teste	0	3
Total da Iteração	20	20
Acumulado	43	43

Fonte: elaborado pelo autor.

Analisando e comparando a injeção e remoção de defeitos, nas três iterações, do programador B (quadros 18, 19 e 20), houve melhoria significativa na revisão do projeto entre as duas primeiras iterações, quintuplicando o desempenho dessa atividade (passando de 8% para 45%). A efetividade da revisão do código removeu, aproximadamente, um terço dos erros em cada iteração. Os números exitosos das revisões possibilitaram reduzir, gradativamente, o número de defeitos presentes na fase de testes das três iterações (6, 4 e 3, respectivamente). Mesmo o período da terceira iteração sendo o dobro de tempo das demais, o número de defeitos diminuiu (o que é extremamente recomendável).

O quadro 21 mostra o resumo dos defeitos da primeira iteração, do programador C. Os defeitos injetados totalizaram 10: quatro (4) injetados na fase de projeto e seis (6) na fase de codificação. Os dados mostram que a revisão do código foi mais eficiente que a revisão do projeto, pois enquanto a primeira removeu 40% dos defeitos, a segunda removeu apenas 20%, restando 40% dos defeitos para os testes. Se forem consideradas apenas as remoções de

defeitos injetados nas respectivas fases, a fase de revisão de projeto teve eficiência de 50% e a fase de revisão de código teve eficiência de 67%.

Quadro 21 – Defeitos – Iteração 1 – Programador C.

Fase	**Defeitos injetados**	**Defeitos removidos**
Planejamento	0	0
Projeto	4	0
Revisão do Projeto	0	2
Código	6	0
Revisão do Código	0	4
Compilação	0	0
Teste	0	4
Total da Iteração	10	10
Acumulado	10	10

Fonte: elaborado pelo autor.

O quadro 22 mostra o resumo dos defeitos da segunda iteração, do programador C. Os defeitos injetados totalizaram 9 (19 defeitos acumulados): cinco (5) injetados na fase de projeto e quatro (4) na fase de codificação. Na remoção dos defeitos, quatro (4) foram removidos na revisão do projeto e dois (2) defeitos foram removidos na revisão da codificação. Um total de três (3) defeitos foi removido na fase de testes. Os dados mostram que a revisão do código foi menos eficiente que a revisão do projeto, pois enquanto a primeira removeu apenas 22% dos defeitos, a segunda removeu 45%, restando 33% dos defeitos para os testes. Se forem consideradas apenas as remoções de defeitos injetados nas respectivas fases, a fase de revisão de projeto teve eficiência de 80% e a fase de revisão de código teve eficiência de 50%.

Quadro 22 – Defeitos – Iteração 2 – Programador C.

Fase	**Defeitos injetados**	**Defeitos removidos**
Planejamento	0	0
Projeto	5	0
Revisão do Projeto	0	4
Código	4	0
Revisão do Código	0	2
Compilação	0	0
Teste	0	3
Total da Iteração	9	9
Acumulado	19	19

Fonte: elaborado pelo autor.

O quadro 23 mostra o resumo dos defeitos da terceira iteração, do programador C. Os defeitos injetados totalizaram 12 (31 defeitos acumulados): seis (6) na fase de projeto e seis (6) na fase de codificação. Esses dados mostram que a revisão do código foi menos eficiente que a revisão do projeto nessa iteração, pois enquanto a primeira removeu 33% do total de defeitos, a segunda removeu 50%, restando 17% dos defeitos para os testes. Se forem consideradas apenas as remoções de defeitos injetados nas respectivas fases, a fase de revisão de projeto teve eficiência de 100% e a fase de revisão de código teve eficiência de 67%.

Quadro 23 – Defeitos – Iteração 3 – Programador C.

Fase	**Defeitos injetados**	**Defeitos removidos**
Planejamento	0	0
Projeto	6	0
Revisão do Projeto	0	6
Código	6	0
Revisão do Código	0	4
Compilação	0	0
Teste	0	2
Total da Iteração	12	12
Acumulado	31	31

Fonte: elaborado pelo autor.

Analisando e comparando os dados de injeção e remoção de defeitos, nas três iterações, do programador C (quadros 21, 22 e 23), percebe-se que houve melhoria na revisão do projeto no decorrer das iterações, chegando a remover metade do total de defeitos da terceira iteração. Isso significa que apenas a metade dos defeitos foi distribuída nas fases posteriores ao projeto. A revisão do código se manteve na média de remover um terço dos defeitos a cada iteração, o que representa uma boa média de efetividade. O número de defeitos removidos na fase de testes (4, 3 e 2) diminuiu devido ao êxito nas fases de revisões.

Globalmente, a análise dos dados de defeitos dos três programadores mostrou uma importante melhoria na detecção e remoção de defeitos nas fases de projeto e codificação, resultados da efetiva utilização dos *checklists* de revisões do SCRUM-PSP-IF. Quanto menos defeitos são encontrados após a realização dos testes de *software*, maiores são as chances de esse produto apresentar a qualidade desejada pelos seus solicitantes.

4.3.3 Análise dos dados de tempo

Para realizar a análise dos dados de tempo, será considerado o tempo gasto pelos programadores em cada fase do SCRUM-PSP-IF, nas três iterações, bem como as médias gerais (das fases e iterações) e os totais, por fase. Esses dados foram retirados dos formulários de Resumo de Plano de Projeto (APÊNDICES 03, 06, 09, 12, 15, 18, 25, 31, 37). As colunas PA, PB e PC significam, respectivamente, Programador A, Programador B e Programador C.

O quadro 24 apresenta o resumo do tempo gasto pelos programadores. A iteração 3 teve duração de duas semanas, ao contrário das iterações 1 e 2, que tiveram duração de uma semana. A fase de planejamento mostra o tempo gasto para obtenção (ou refinamento) dos requisitos do módulo, com média geral de 35 minutos.

A relação entre o tempo gasto no projeto e na revisão do projeto forneceu aos programadores o tempo necessário não só para utilizar as verificações de forma coerente, mas também para alcançar altos índices de qualidade na revisão do projeto, medido, posteriormente, pelo componente $PQI_{DesignReviewTime}$.

A relação entre o tempo de codificação e o tempo da revisão do código mostra uma tendência dos métodos ágeis: uma proporção muito maior de tempo de codificação do que de tempo de revisão (ou documentação). O tempo de compilação se mostrou quase irrisório, pois não foram encontrados defeitos nessa fase, sendo que na iteração 3 os tempos foram 25, 15 e 10,

pois foram realizadas cinco, três e duas compilações, respectivamente. O tempo dos testes se mostrou coerente tanto para testar as funcionalidades dos módulos quanto para o preenchimento do formulário *template*. O tempo médio da fase de *postmortem* da iteração 2 mostrou melhoria no tempo de preenchimento, comparado com a iteração 1.

Quadro 24 – Registros de Tempo.

Fase	Tempo Iteração 1				Tempo Iteração 2				Tempo Iteração 3				Média Geral
	PA	PB	PC	Média Fase	PA	PB	PC	Média Fase	PA	PB	PC	Média Fase	
Plan.	15	55	20	30	45	20	25	30	60	30	40	43	35
Projeto	85	100	80	88	110	90	85	95	155	110	95	120	100
Rev. Projeto	60	75	50	61	90	75	60	75	95	75	70	80	72
Código	580	930	1010	840	845	1005	920	923	1925	2055	2000	1993	1252
Rev. Código	105	70	45	73	175	85	90	116	380	215	175	256	148
Comp.	5	5	5	5	5	5	5	5	25	15	10	16	9
Teste	95	115	125	111	75	145	175	131	225	235	340	266	170
Post.	110	110	125	115	45	75	90	70	180	150	125	151	112
Média Iteração	131	182	182	165	173	187	181	180	380	360	356	365	236
Total	1055	1460	1460	1325	1390	1500	1450	1446	3045	2885	2855	2928	1830

Fonte: elaborado pelo autor.

4.3.4 Análise dos dados de tamanho

Mesmo que os dados referentes ao tamanho dos programas sejam provenientes apenas do total de linhas de código (desconsiderando linhas novas, reutilizadas ou alteradas, por exemplo), servem de base para algumas medidas de qualidade do SCRUM-PSP-IF e serão referenciados com base no quadro 25, que considera o tamanho dos programas e suas médias. O valor acumulado nas três iterações, de cada programador, também será considerado.

Quadro 25 – Tamanho dos programas desenvolvidos.

Tamanho (LOC) Iteração 1	Tamanho (LOC) Iteração 2	Tamanho (LOC) Iteração 3	Acumulado (LOC)	Média de Tamanho (LOC)
Programador A				
339	398	754	1491	497
Programador B				
508	367	1063	1938	646
Programador C				
342	312	864	1518	506

Fonte: elaborado pelo autor.

Relacionando o tamanho dos programas com o tempo de desenvolvimento, na fase de Código (quadro 24), o programador A teve uma média de 35LOC por hora na codificação da Iteração 1; 28LOC na Iteração 2 e 23LOC por hora na Iteração 3 (28 LOC de média geral).

O programador B teve uma média de 32LOC por hora na codificação da Iteração 1, 22LOC na Iteração 2 e 31LOC por hora na codificação da Iteração 3 (28 LOC de média geral).

O programador C teve uma média de 20LOC por hora na codificação da Iteração 1, 20LOC na Iteração 2 e 26LOC por hora na codificação da Iteração 3 (22 LOC de média geral).

Provavelmente, a complexidade sobre as funcionalidades a serem codificadas influencia o desempenho dos programadores. Independente disso, uma média de mais de 20 linhas de código por hora (para todos os programadores) pode ser considerado um desempenho eficiente. Os programadores A e C tiveram desempenhos próximos, porém, o programador C atingiu produções de mais de 600 linhas de código por iteração, demonstrando desempenho superior, provavelmente por dominar melhor as tecnologias envolvidas no desenvolvimento ou por ter reutilizado códigos existentes.

4.4 Relato dos resultados da pesquisa

O relato dos resultados da pesquisa descreve os dados de forma que o leitor possa entende os assuntos que foram estudados, facilitando a leitura e interpretação das descobertas de pesquisa, bem como servindo de principal fonte de informação para julgar a qualidade do estudo (RUNESON e HOST, 2009).

Os resultados dessa pesquisa são compostos por cinco medidas de qualidade de *software* do PSP, utilizadas pelo SCRUM-PSP-IF: Índice de Qualidade do Processo (PQI – *Process Quality Index*), Densidade de Defeitos (*Defect Density*), Taxa de Revisão (*Review Rate*), Rendimento (*Yield*) e A/FR (*Appraisal to Failure Ratio*) para demostrar o desempenho individual dos engenheiros de *software* no processo de desenvolvimento de *software*.

4.4.1 PQI – Process Quality Index – Índice de Qualidade do Processo

Os componentes PQI representam as principais medidas do SCRUM-PSP-IF, normalizados entre os valores 0 e 1, sendo que zero indica baixa qualidade no processo, enquanto um representa alta qualidade no processo. Os resultados dos componentes PQI que ultrapassarem

o valor máximo (o que pode ser considerado como extremamente satisfatório), como por exemplo, o componente PQI $_{DesignReviewTime}$, serão representados pelo valor máximo (1). O PQI global é representado pelo produto dos quatro índices.

Conforme Humphrey (2000a), o valor global do índice de qualidade do processo deve estar acima de 0,4 para apresentar um índice de qualidade global aceitável. Porém, não menciona nada quanto à proibição de avaliar os componentes separadamente. Os quadros 26, 27 e 28 mostram os resultados da qualidade do processo, individualmente e globalmente, considerando os programadores e as iterações.

Quadro 26 – Resumo de Qualidade – Programador A.

Medida PQI	Iteração 1	Iteração 2	Iteração 3
PQI $_{DesignCodeTime}$	0,15	0,13	0,08
PQI $_{DesignReviewTime}$	1,41 (1)	1,63 (1)	1,23 (1)
PQI $_{CodeReviewTime}$	0,36	0,41	0,39
PQI $_{UnitTestDefects}$	0,45	0,83	1
PQI global	0,02	0,04	0,03

Fonte: elaborado pelo autor.

Quadro 27 – Resumo de Qualidade – Programador B.

Medida PQI	Iteração 1	Iteração 2	Iteração 3
PQI $_{DesignCodeTime}$	0,11	0,09	0,05
PQI $_{DesignReviewTime}$	1,50 (1)	1,66 (1)	1,36 (1)
PQI $_{CodeReviewTime}$	0,15	0,17	0,21
PQI $_{UnitTestDefects}$	0,62	0,66	1,25 (1)
PQI global	0	0	0,01

Fonte: elaborado pelo autor.

Quadro 28 – Resumo de Qualidade – Programador C.

Medida PQI	Iteração 1	Iteração 2	Iteração 3
PQI $_{DesignCodeTime}$	0,08	0,09	0,05
PQI $_{DesignReviewTime}$	1,25 (1)	1,41 (1)	1,47 (1)
PQI $_{CodeReviewTime}$	0,09	0,20	0,17
PQI $_{UnitTestDefects}$	0,62	0,71	1.4 (1)
PQI global	0	0,01	0

Fonte: elaborado pelo autor.

Analisando os resultados globais dos três últimos quadros, o índice de qualidade do processo, calculado pelo produto dos quatro componentes do PQI não alcançou a meta mínima de qualidade de 0,4, em nenhuma das três iterações, principalmente pelo baixo valor alcançado pelo componente $PQI_{DesignCodeTime}$, pois na grande maioria dos métodos ágeis, como *Scrum*, o tempo de projeto (se existir), é extremamente inferior ao tempo de codificação Porém, individualmente, alguns componentes se destacaram em todas as iterações, como pode ser visto a seguir.

$PQI_{DesignReviewTime}$

O componente de qualidade de revisão de projeto do SCRUM-PSP-IF, $PQI_{DesignReviewTime}$, expresso pela divisão entre o dobro do tempo de revisão de projeto e o tempo de projeto foi a medida do PQI que apresentou melhor desempenho, alcançando o índice máximo de qualidade, pelos três programadores, nas três iterações. Isso se deu, principalmente, pela eficiência na utilização da *checklist* de projeto, que ultrapassou os valores recomendados de que o tempo gasto na fase de revisão de projeto deve ser no mínimo a metade do tempo gasto na fase de projeto (HUMPHREY, 2000a).

Conforme quadro 26, o programador A atingiu índices de 1,41, 1,63 e 1,23 na primeira, segunda e terceira iteração, respectivamente. O programador B atingiu índices de 1,50, 1,66 e 1,36 na primeira, segunda e terceira iteração, respectivamente (conforme quadro 27). Enquanto que o programador C atingiu índices de 1,25, 1,41 e 1,47 na primeira, segunda e terceira iteração, respectivamente (conforme quadro 28). Esses valores foram normalizados para 1, o valor máximo do PQI.

A relação entre o tempo de projeto e o tempo da revisão de projeto foi bem coerente, pois alcançou o topo do índice de qualidade. Porém, isso garante apenas que o tempo de revisão foi bem alocado, mas não informa se a remoção dos defeitos foi bem sucedida. Para obter os resultados quanto ao avanço (ou não) do desempenho na correção dos erros de projeto, pode-se relacionar o $PQI_{DesignReviewTime}$ com o número de defeitos injetados e removidos na fase de projetos.

Na iteração 1, o programador A injetou 6 defeitos no projeto e removeu apenas 2 desses defeitos na revisão do projeto (os outros 4 defeitos foram removidos na fase de testes). Na iteração 2, o mesmo programador injetou 6 defeitos no projeto, removendo 5 defeitos na

revisão do projeto (o outro defeito foi removido na fase de revisão de código). Já na iteração 3, o programador A injetou 8 defeitos no projeto e removeu 7 defeitos na revisão do projeto (o outro defeito foi removido na fase de testes).

As constatações do parágrafo anterior mostram que o uso da *checklist* de projeto, no decorrer das iterações, melhorou significantemente a eficiência na remoção dos defeitos injetados pelo programador A, pois enquanto ele removeu apenas um terço dos defeitos de projeto na revisão da iteração 1, removeu 5 dos 6 defeitos de projeto da iteração 2 e 7 dos 8 defeitos de projeto da iteração 3, restando apenas 1 defeito (nas duas últimas iterações) para serem removidos pelas fases posteriores. Na iteração 1, os 4 defeitos de projeto que não foram corrigidos na fase de revisão de projeto, foram corrigidos apenas na fase de testes, enquanto que o defeito restante da iteração 2 foi corrigido na fase de revisão de código e o defeito restante da iteração 3 foi corrigido na fase de testes. Além da porcentagem de defeitos corrigidos na fase de revisão de projeto ter aumentado significativamente, a porcentagem de número de defeitos para serem corrigidos nos testes diminuiu, desonerando tempo de projeto e aumentando a qualidade do produto final.

Na iteração 1, o programador B injetou 3 defeitos no projeto e removeu apenas 1 defeito na revisão do projeto (os outro 2 defeitos foram removidos na fase d testes). Na iteração 2, o mesmo programador injetou 6 defeitos no projeto e removeu 5 na revisão do projeto (o defeito restante foi removido na fase de testes). Já na iteração 3, o programador B injetou 13 defeitos no projeto, removendo 11 defeitos na revisão do projeto (os outros 2 defeitos foram removidos na fase de revisão de código).

As constatações do parágrafo anterior mostram que o uso da *checklist* de projeto, no decorrer das iterações, também melhorou significantemente a eficiência na remoção dos defeitos injetados pelo programador B, pois enquanto ele removeu apenas um defeito de projeto na revisão de projeto da iteração 1, removeu 5 dos 6 defeitos de projeto da iteração 2 e 11 dos 13 defeitos de projeto da iteração 3, restando apenas 3 defeitos para serem removidos pelas fases posteriores nas duas últimas iterações. Na iteração 1 e 2, os defeitos de projeto foram corrigidos apenas na fase de testes, enquanto que os 2 defeitos de projeto restantes da iteração 3 foram corrigidos na fase de codificação. Além da porcentagem de defeitos corrigidos na fase de revisão de projeto aumentar significativamente, incluindo as remoções de defeitos de projeto em fases posteriores, como na fase de codificação, a porcentagem de número de

defeitos restantes para correções na fase de testes diminuiu consideravelmente, aumentando a qualidade do produto final e desonerando tempo de projeto.

Na iteração 1, o programador C injetou 4 defeitos no projeto e removeu 2 defeitos na revisão do projeto (os outros 2 foram removidos nas fases de revisão do código e testes). Na iteração 2, o programador C injetou 5 defeitos no projeto e removeu 4 na revisão do projeto (o defeito restante foi removido na revisão código). Já na iteração 3, o programador B injetou 6 defeitos no projeto, removendo todos os defeitos de projeto na revisão do projeto.

As constatações do parágrafo anterior mostram que o uso da *checklist* de projeto, no decorrer das iterações, melhorou significantemente a eficiência na remoção dos defeitos injetados pelo programador C, pois enquanto ele removeu metade dos defeitos de projeto na revisão de projeto da iteração 1, removeu 4 dos 5 defeitos de projeto da iteração 2, ele removeu todos os defeitos de projeto da iteração 3. Na iteração 1 os defeitos de projeto restantes foram corrigidos na fase de revisão de código (1) e na fase de testes (1), enquanto que o defeito restante da iteração 2 foi corrigido na fase de revisão de código. Na iteração 3 não restaram defeitos da fase de projeto. Além da porcentagem de defeitos corrigidos na fase de revisão de projetos aumentar significativamente, a porcentagem de defeitos corrigidos na fase de testes diminuiu, aumentando a qualidade do produto final.

PQI $_{\text{UnitTestDefects}}$

Como quase todos os programas (módulos) produzidos não ultrapassaram 1.000 linhas de código (conforme quadro 31), para calcular corretamente o PQI $_{\text{UnitTestDefects}}$ foi criada uma regra de três simples para encontrar a proporcionalidade de defeitos nos testes unitários para cada 1.000 linhas, com base nos defeitos reais encontrados nos testes.

Nesse contexto, o programador A construiu o programa com total de 339 LOC na primeira iteração, 398 na segunda e 754 na terceira iteração. Como foram removidos 6 defeitos na fase de testes da primeira iteração, 3 defeitos na segunda e 4 defeitos na terceira, se o programa tivesse 1000 linhas de código seriam removidos, aproximadamente, 17 defeitos na iteração 1, 7 defeitos na iteração 2 e 5 defeitos na iteração 3.

O programador B criou um programa com total de 508 LOC na primeira iteração, 367 na segunda e 1063 na terceira. Como foram removidos 6 defeitos na fase de testes da primeira

iteração, 4 defeitos na segunda e 3 defeitos na terceira, se o programa tivesse 1000 linhas de código seriam removidos, aproximadamente, 11 defeitos na iteração 1, 10 defeitos na iteração 2 e 3 defeitos na iteração 3.

O programador C desenvolveu programa com total de 342 LOC na primeira iteração, 312 na segunda e 864 na terceira. Como foram removidos 4 defeitos na fase de testes da primeira iteração, 3 defeitos na segunda e 2 defeitos na terceira, se o programa tivesse 1000 linhas de código seriam removidos, aproximadamente, 11 defeitos na iteração 1, 9 defeitos na iteração 2 e 2 defeitos na iteração 3. Esses cálculos foram feitos para os 8 módulos (programas) produzidos, pois na terceira iteração do programador B, que ultrapassou 1000 linhas de código, isso não foi mais necessário.

A medida de qualidade do programa, representada pelo componente PQI $_{\text{UnitTestDefects}}$ (expressa pela divisão de 10 pela soma do número de defeitos (encontrados nos testes unitários a cada 1000 linhas de código) com 5, que representa o valor alvo para essa métrica, ou seja, a densidade de defeitos da fase de testes unitários não deve ser superior a 5 defeitos/KLOC) obteve um alto rendimento, pois os 3 programadores aumentaram o índice de qualidade desse componente no decorrer das iterações, obtendo valores máximos do índice na última iteração.

O programador A teve um índice para PQI $_{\text{UnitTestDefects}}$ de 0,45 na primeira iteração, subiu para 0,83 na segunda iteração e alcançou o valor máximo, 1, na terceira iteração. O programador B teve um índice para PQI $_{\text{UnitTestDefects}}$ de 0,62 na primeira iteração, subiu para 0,66 na segunda iteração e ultrapassou o valor máximo, 1, na terceira iteração. O programador C teve um índice para PQI $_{\text{UnitTestDefects}}$ de 0,62 na primeira iteração, subiu para 0,71 na segunda iteração e ultrapassou o valor máximo, 1, na terceira iteração.

Os dados acima mostram uma evolução satisfatória dos três programadores no decorrer das iterações, pois houve uma gradativa melhoria na redução de defeitos dos testes unitários, mostrando, também, a eficiência no uso das listas de verificação (*checklists*) do projeto.

PQI $_{\text{DesignCodeTime}}$

O componente de qualidade do projeto do SCRUM-PSP-IF, PQI $_{\text{DesignCodeTime}}$, representado pela divisão entre o tempo total da fase de projeto e o tempo total de codificação apresentou um baixo desempenho para todos os programadores, em todas as iterações, não alcançando

sequer valores aceitáveis. É recomendado que o tempo gasto na fase de projeto seja maior que o tempo gasto na fase de codificação (HUMPHREY, 2000a).

Enquanto que o tempo de projeto foi extremamente menor que o tempo de codificação (uma tendência natural em projetos ágeis de *software*), o PQI $_{DesignCodeTime}$ nunca apresentará um alto valor. No decorrer das iterações, inclusive, esse índice diminuiu, pois o programador A obteve 0,15, 0,13 e 0,08 para a iteração 1, 2 e 3, respectivamente, o programador B obteve 0,11, 0,09 e 0,05 para as iterações 1, 2 e 3 e o programador C obteve 0,08, 0,09 e 0,05 para a iteração 1, 2 e 3.

PQI $_{CodeReviewTime}$

O componente de qualidade de revisão de código, expresso pela divisão entre o dobro do tempo de revisão de código e o tempo de codificação também apresentou um baixo rendimento. Porém, se mostrou um pouco mais eficiente do que a medida anterior (qualidade do projeto), principalmente pelas iterações do programador A.

É recomendado que o tempo gasto na fase de revisão de código seja, no mínimo, a metade do tempo gasto na fase de codificação (HUMPHREY, 2000a). Considerando que o tempo de revisão do código é consideravelmente menor que o tempo de codificação (outra tendência natural em projetos ágeis), o $PQI_{CodeReviewTime}$ dificilmente será maior que 0,4 (valor alvo global). Situação que só ocorreu na iteração 2, do programador A, com 0,41. As iterações 1 e 3 do mesmo programador apresentaram valores de 0,36 e 0,39, respectivamente.

O programador B obteve índices de 0,15, 0,17 e 0,21 para as iterações 1, 2 e 3, respectivamente. O que mostrou que, apesar de não alcançar um índice aceitável, o programador teve avanços gradativos nas três iterações.

Já o programador C obteve índices de 0,09, 0,20 e 0,17 para as iterações 1, 2 e 3, respectivamente. Mesmo não alcançando os valores desejáveis, o programador melhorou consideravelmente os valores da segunda e terceira iteração, comparados com a primeira.

4.4.2 Densidade de defeitos (*defect density*)

Originalmente, a densidade dos defeitos do PSP é medida referenciando os defeitos encontrados em um programa a cada mil linhas de código, considerando as linhas de código novas e alteradas (HUMPHREY, 2000a). Como o SCRUM-PSP-IF utiliza apenas o total de linhas de código dos programas como medida de tamanho, sem considerar outras categorias, a densidade deve ser calculada utilizando esse valor.

Os quadros de defeitos injetados e removidos nas três iterações (15 até o 23), bem como o tamanho dos programas (quadro 31) fornecem os dados necessários para realiza o cálculo da densidade dos defeitos (quadro 29) do projeto de *software* SIPS, separando os resultados por iteração e programador, utilizando a fórmula: 1000 * número total de defeitos / total de LOC do programa. Segundo Humphrey (2000a), a densidade de defeitos também pode ser medida para fases específicas do processo de desenvolvimento.

Quadro 29 – Densidade de Defeitos.

Programador		**Total de Defeitos**	**Total de LOC**	**Densidade Defeitos**
A	Iteração 1	14	339	41
	Iteração 2	13	398	32
	Iteração 3	19	754	25
B	Iteração 1	12	508	23
	Iteração 2	11	367	29
	Iteração 3	20	1063	18
C	Iteração 1	10	342	29
	Iteração 2	9	312	28
	Iteração 3	12	864	13

Fonte: elaborado pelo autor.

O quadro 29 mostra índices extremamente satisfatórios para a densidade dos defeitos nos módulos desenvolvidos nas três iterações do SCRUM-PSP-IF, principalmente se comparados aos níveis alcançados por engenheiros de *software* que não utilizam PSP, cuja densidade de defeitos pode variar de 20 a 40 (ou mais) defeitos a cada mil linhas de código (HUMPHREY, 2000a).

Os resultados da análise da densidade de defeitos consideram a melhoria da qualidade do programa, devido à redução do número de erros após cada iteração:

1. O programador A partiu de uma densidade de defeitos com valor 41 na primeira iteração, baixando para 32 na segunda iteração e 25 na terceira iteração.
2. O programador B partiu de uma densidade de defeitos com valor 23 na primeira iteração e, apesar de ter aumentado a densidade para 29 na segunda iteração, diminui consideravelmente para 18 a densidade na terceira iteração.
3. O programador C partiu de uma densidade de defeitos com valor 29 na primeira iteração, baixando para 28 na segunda iteração e para 13 na terceira iteração, mostrando o melhor desempenho entre os programadores, nessa medida.

4.4.3 Taxa de revisão (*review rate*)

Dados do PSP mostram que quando a revisão de projeto ou código é mais rápida do que cerca de 150 a 200 linhas de código novas ou alteradas por hora, mais defeitos tendem a passar despercebidos (HUMPHREY, 2000a). Kemerer e Paulk (2009) também sugerem uma taxa de inspeção de código inferior a 200 LOC por hora e, ainda, indicam uma taxa inferior a quatro (4) páginas por hora quando a inspeção for realizada na documentação do projeto do *software*. A taxa de revisão é uma medida que atua como uma espécie de guia, orientando cada programador quanto ao tempo que deve gastar nas revisões e inspeções de código e projeto.

As categorias das *checklists* de revisão de projeto e código do SCRUM-PSP (adaptadas do PSP), somadas aos formulários de resumo de projeto, possibilitaram aos programadores verificar tanto o tempo de projeto detalhado quanto o número de linhas de código revisadas no programa, na busca por defeitos, como mostram os quadros 30 e 31.

Quanto à revisão de projeto (quadro 30), os resultados do programador A mostram uma taxa de revisão de projeto de 30 minutos por página nas iterações 1 e 2 e 23 minutos por página na iteração 3. Os resultados do programador B mostram uma taxa de revisão de projeto de 37 minutos por página na iteração 1, 25 minutos por página na iteração 2 e 18 minutos por página na iteração 3. Os resultados do programador C mostram uma taxa de revisão de projeto de 25 minutos por página na iteração 1, 30 minutos por página na iteração 2 e 17 minutos por página na iteração 3. Esse números mostram a evolução gradativa dos 3 programadores nas revisões de projeto, mesmo o programador C tendo aumentado o tempo de revisão por páginas na iteração 2, baixou drasticamente o tempo de revisão por página na iteração 3, superando as duas anteriores.

Quadro 30 – Taxa de Revisão de Projeto.

Programador		N.º páginas projeto	Tempo revisão projeto	Taxa de Revisão Projeto
A	Iteração 1	2	60	Uma pág. a cada 30min
	Iteração 2	3	90	Uma pág. a cada 30min
	Iteração 3	4	95	Uma pág. a cada 23min
B	Iteração 1	2	75	Uma pág. a cada 37min
	Iteração 2	3	75	Uma pág. a cada 25min
	Iteração 3	4	75	Uma pág. a cada 18min
C	Iteração 1	2	50	Uma pág. a cada 25min
	Iteração 2	2	60	Uma pág. a cada 30min
	Iteração 3	4	70	Uma pág. a cada 17min

Fonte: elaborado pelo autor.

Quanto à revisão de código (quadro 31), os resultados do programador A mostram uma taxa de revisão de código de 193 LOC por hora na iteração 1, 137 LOC na iteração 2 e 119 LOC na iteração 3. Os resultados do programador B mostram uma taxa de revisão de código de 437 LOC por hora na iteração 1, 262 LOC na iteração 2 e 303 LOC por hora na iteração 3. Os resultados do programador C mostram uma taxa de revisão de código de 456 LOC por hora na iteração 1, 208 LOC por hora na iteração 2 e 297 LOC por hora na iteração 3.

Quadro 31 – Taxa de Revisão de Código.

Programador		LOC revisadas	Tempo revisão Código	Taxa de Revisão Código
A	Iteração 1	339	105	193 LOC por hora
	Iteração 2	398	175	137 LOC por hora
	Iteração 3	754	380	119 LOC por hora
B	Iteração 1	508	70	437 LOC por hora
	Iteração 2	367	85	262 LOC por hora
	Iteração 3	1063	215	303 LOC por hora
C	Iteração 1	342	45	456 LOC por hora
	Iteração 2	312	90	208 LOC por hora
	Iteração 3	864	175	297 LOC por hora

Fonte: elaborado pelo autor.

Esses resultados mostram que o programador A teve um bom desempenho, pois além de não ultrapassar as 200 LOC (indicada como valor base), o que poderia denotar uma revisão muito acelerada, ele não baixou de 119 LOC por hora, o que poderia indicar uma revisão de código lenta. Os programadores B e C obtiveram resultados menos satisfatórios, pois ultrapassaram, em todas as iterações, as 200 LOC por hora, o que demonstrou, claramente, uma revisão extremamente rápida (o que pode prejudicar o desempenho da busca por defeitos, na fase de codificação).

4.4.4 Rendimento (*yield*)

O SCRUM-PSP-IF também utilizou a medida de rendimento do PSP, considerando os defeitos injetados e removidos em cada fase do processo de desenvolvimento, tendo como resultado a porcentagem total dos defeitos encontrados e removidos nas fases, conforme quadros 32, 33 e 34. Pode-se observar que o rendimento da fase é calculado apenas nas fases em que houveram defeitos removidos.

Os resultados do programador A (quadro 32) demonstram evolução no rendimento da fase de revisão do projeto, pois aumentou seu próprio desempenho de 33,3% na primeira iteração para 83,3% na segunda iteração e 87,5% na terceira iteração. Também demonstram evolução no rendimento na fase de revisão de código, pois aumentou o rendimento de 50% na iteração 1, para 62,5% na iteração 2 e 66,6% na iteração 3. Os testes mostraram eficiência máxima, removendo todos os defeitos encontrados.

Quadro 32 – Rendimento das Fases – Programador A.

Fase	**Defeitos injetados**			**Defeitos removidos**			**Defeitos entrada fase**			**Rendimento da fase**		
	I1	**I2**	**I3**	**I1**	**I2**	**I3**	**I1**	**I2**	**I3**	**I1**	**I2**	**I3**
Projeto	6	6	8	0	0	0	0	0	0			
Revisão do Projeto	0	0	0	2	5	7	6	6	8	33,3%	83,3%	87,5%
Código	8	7	11	0	0	0	4	1	1			
Revisão do Código	0	0	0	6	5	8	12	8	12	50%	62,5%	66,6%
Compilação	0	0	0	0	0	0	6	3	4			
Teste	0	0	0	6	3	4	6	3	4	100%	100%	100%
Total	14	13	19	14	13	19	-	-	-	-	-	-

Fonte: elaborado pelo autor.

Os resultados do programador B (quadro 33) demonstram evolução no rendimento da fase de revisão do projeto, pois aumentou seu desempenho de 25% na primeira iteração para 83,3% na segunda iteração e 84,6% na terceira iteração. Também demonstram evolução no rendimento na fase de revisão de código, mesmo o rendimento tendo diminuído de 45,5% na iteração 1 para 28,5% na iteração 2, porém aumentou para 66,6% na iteração 3. Os testes mostraram eficiência máxima, removendo 100% dos defeitos encontrados. Salienta-se que na iteração 3 a fase de código apresentou rendimento de 28,5%.

Quadro 33 – Rendimento das Fases – Programador B.

Fase	**Defeitos injetados**			**Defeitos removidos**			**Defeitos entrada fase**			**Rendimento da fase**		
	I1	**I2**	**I3**	**I1**	**I2**	**I3**	**I1**	**I2**	**I3**	**I1**	**I2**	**I3**
Projeto	3	6	13	0	0	0	0	0	0			
Revisão do Projeto	0	0	0	1	5	11	3	6	13	25%	83,3%	84,6%
Código	9	5	7	0	0	2	2	1	2			28,5%
Revisão do Código	0	0	0	5	2	4	11	6	7	45,5%	28,5%	57,1%
Compilação	0	0	0	0	0	0	6	4	3			
Teste	0	0	0	6	4	3	6	4	3	100%	100%	100%
Total	12	11	20	12	11	20	-	-	-	-	-	-

Fonte: elaborado pelo autor.

Os resultados do programador C (quadro 34) demonstram evolução no rendimento da fase de revisão do projeto, pois aumentou o desempenho de 50% na primeira iteração para 80% na segunda iteração e 100% na terceira iteração. Também demonstram manutenção do rendimento de 66,6% (da iteração 1) para a iteração 3, passando por um rendimento de 50% na iteração 2. Os testes mostraram eficiência máxima, removendo 100% dos defeitos encontrados.

Quadro 34 – Rendimento das Fases – Programador C.

Fase	Defeitos injetados			Defeitos removidos			Defeitos entrada fase			Rendimento da fase		
	I1	**I2**	**I3**	**I1**	**I2**	**I3**	**I1**	**I2**	**I3**	**I1**	**I2**	**I3**
Projeto	4	5	6	0	0	0	0	0	0			
Revisão do Projeto	0	0	0	2	4	6	4	5	6	50%	80%	100%
Código	6	4	6	0	0	0	2	1	0			
Revisão do Código	0	0	0	4	2	4	4	3	2	66,6%	50%	66,6%
Compilação	0	0	0	0	0	0	4	3	2			
Teste	0	0	0	4	3	2	4	3	2	100%	100%	100%
Total	10	9	12	10	9	12	-	-	-	-	-	-

Fonte: elaborado pelo autor.

4.4.5 A/FR – *Appraisal to Failure Ratio*

A medida de qualidade A/FR do SCRUM-PSP-IF, retirada das práticas do PSP, mede o custo de qualidade, relacionando o tempo gasto com as revisões de projeto e de código (o tempo de *appraisal*, 100 * (tempo da revisão de projeto + tempo da revisão do código) / tempo total de desenvolvimento) com o tempo gasto nos testes (o tempo de *failure*, 100 * (tempo da compilação + tempo dos testes unitários) / tempo total de desenvolvimento). A relação (valor de *ratio*) entre esses dois tipos de custos de qualidade é descrita pela divisão do resultado do custo de avaliação pelo resultado do custo de falha (A/FR = COQ *appraisal* / COQ *failure*).

Considerando o total de tempo de compilação, o total de tempo dos testes unitários e o total de tempo de desenvolvimento (quadro 24) se pode chegar ao índice do custo de qualidade de falha. Considerando o tempo total de revisão de projeto, o tempo total de revisão de código e o tempo total de desenvolvimento, pode-se chegar ao índice de qualidade de avaliação (ou reparo). O quadro 35 apresenta o custo de qualidade de avaliação, o custo de qualidade de falha e a relação entre esses dois custos.

Quadro 35 – Relação de avaliação e falha – custos da qualidade.

Programador/Iteração		**Custo de Avaliação**	**Custo de Falha**	**A/FR**
A	Iteração 1	15,6	9,4	1,6
	Iteração 2	19	5,7	3,3
	Iteração 3	15,5	8,2	1,8
B	Iteração 1	9,9	8,2	1,2
	Iteração 2	10,6	10	1
	Iteração 3	10	8,6	1,1
C	Iteração 1	6,5	8,9	0,7
	Iteração 2	10,3	12,4	0,8
	Iteração 3	8,5	12,2	0,6

Fonte: elaborado pelo autor.

Com base no valor alvo indicado pelo SEI (HUMPHREY, 2000a), cerca de 2,0, pode-se observar que o programador A teve um bom desempenho, pois ultrapassou esse valor na segunda iteração (3,3) e se aproximou desse valor nas iterações um e três (1,6 e 1,8), respectivamente. O programador B teve um desempenho médio, pois ultrapassou a metade do valor base (média de 1,1 para as três iterações). Já o programador C teve um desempenho um pouco abaixo do programador B, (média de 0,7 para as três iterações).

De modo geral, a relação A/F aplicada pelo SCRUM-PSP-IF no projeto de *software* teve um bom desempenho, a um custo global razoável de qualidade. Como resultado prático no processo de desenvolvimento de *software* se tem baixa densidade de defeito nos testes unitários e alta qualidade do produto final. Se medido em muitas organizações, provavelmente A/FR apresentaria um valor próximo à zero. Porém, em equipes com alto nível de maturidade, o valor excederia 2,0.

4.4.6 Relato dos programadores sobre o SCRUM-PSP-IF

Essa seção apresenta um breve relato, dos três programadores (A, B e C), com justificativas sobre a viabilidade (ou não) do processo proposto. Esses relatos foram enviados por *e-mail*, entre 11 e 15 de agosto de 2017.

Programador A

“O SCRUM-PSP-IF propõe desenvolver *software* através da mistura da agilidade e disciplina, dosando o quanto de cada técnica deve ser utilizado, o que pode representar um dos principais segredos para o sucesso do modelo. Considero o processo integrado proposto totalmente viável, desde que cada engenheiro de *software* tenha a disciplina e dedicação necessárias para utilizar os documentos do modelo, na ordem e de forma corretas. Uma vantagem interessante do SCRM-PSP-IF é com relação aos testes, pois o modelo trás um *template* para os testes serem registrados e esses são planejados e realizados de acordo com o projeto individual do módulo, realizado nas fases anteriores, além dos próprios requisitos, elencados no início do projeto do sistema. As reuniões *Scrum* promoveram interações diárias, que auxiliaram no uso dos materiais do PSP e no alinhamento das atividades de desenvolvimento. SCRUM-PSP-IF foi uma contribuição interessante e se mostrou eficiente no que se propôs a fazer, reduzir o número de defeitos e aumentar a qualidade dos produtos de *software*.”

Programador B

“A primeira impressão do SCRUM-PSP-IF foi que não seria viável, pois é formado por um conjunto considerável de artefatos do PSP, que despendem de tempo, dedicação e muita atenção para serem preenchidos. Positivamente, pesou o fato de a equipe estar habituada a métodos ágeis de desenvolvimento, utilizando, geralmente, os requisitos para documentar e projetar sistemas. Mensurar o tamanho dos programas é um ponto extremamente positivo, pois com isso se tem informações de quantas linhas de código foram produzidas (desempenho) em determinado período, por determinado programador. Como o modelo foi proposto para pequenos projetos, se adequou bem aos projetos executados pela Instituição.”

Programador C

“O processo se mostrou viável para gestão (o que *Scrum* já fazia), mas principalmente para o planejamento e acompanhamento do projeto de *software*. SCRUM-PSP-IF manteve a divisão cíclica das tarefas e os artefatos *Scrum* (*backlogs* do produto e das *sprints*), o que colaborou para a sua utilização. Porém, adicionou documentos do PSP que exigiram uma mudança de hábito no desenvolvimento de *software*. Com isso, foi possível melhorar a qualidade final dos

produtos, com índices de qualidades bem definidos, reduzindo o número de defeitos e cumprindo os prazos de entrega. Uma das principais vantagens diz respeito ao registro de defeitos, devidamente categorizados, que disponibilizam ao programador (e sua equipe) um histórico de erros e soluções, o que melhora, consideravelmente, o seu trabalho e a qualidade dos produtos. Sem contar que a automelhoria, o autogerenciamento melhoraram o trabalho e as responsabilidades como equipe."

4.4.7 Análise do relato dos programadores sobre o SCRUM-PSP-IF

Considerando o relato dos três programadores, pode-se dizer que o SCRUM-PSP-IF foi avaliado positivamente, pois foi considerado viável para ser utilizado em pequenos projetos de *software* que utilizam método ágil de desenvolvimento *Scrum*. A integração com o PSP trouxe a disciplina necessária para cada programador realizar seu trabalho com mais eficiência e qualidade.

Como pontos positivos, o *template* de testes, que registrou os cenários de testes necessários para validar os requisitos e o projeto do programa; as reuniões diárias do time de desenvolvimento, que trouxeram soluções aos problemas enfrentados; a gestão trazida pelo uso do SCRUM-PSP-IF, onde cada indivíduo conseguir planejar, acompanhar e melhorar o seu trabalho e o registro dos defeitos, que gerou um histórico dos defeitos encontrados e a solução para os defeitos removidos, ajudando os programadores a não cometerem mais os mesmos erros e trazendo soluções para erros similares.

Como pontos negativos, pode-se citar o número de documentos a ser preenchido no modelo proposto, que pode ser considerado cansativo e a resistência a mudanças, pois os programadores tiveram que alterar a sua forma de desenvolver sistemas.

4.5 Ameaça à validade do estudo

A principal ameaça à validade do estudo diz respeito ao protocolo de estudo de caso, pois este não foi proposto antes da realização dos procedimentos de campo, no projeto do estudo de caso. Porém, o projeto não envolve apenas a criação do protocolo, mas também a definição do próprio caso (projeto de *software*), das unidades de análise (programadores) e do ambiente de aplicação (reitoria), incluindo a devida caracterização do projeto de *software*, do

gerenciamento, da tecnologia, do sistema e dos membros da equipe. Portanto, o projeto contempla os objetivos do estudo de caso, bem como as questões de pesquisa e métodos.

4.6 Considerações finais do capítulo

Este capítulo apresentou a aplicação do modelo integrado SCRUM-PSP-IF, através de um estudo de caso exploratório, na Reitoria de uma Instituição Pública de Ensino, entre os meses de janeiro e fevereiro de 2017. Os dados históricos foram coletados utilizando arquivos e analisados através do método de análise estatística, gerando os resultados da pesquisa. O estudo de caso envolveu as etapas de projeto, coleta, análise dos dados e relatórios de resultados, com base nas diretrizes propostas por Runeson e Host (2009) e Wholin e Aurum (2014). O projeto do estudo de caso definiu o Caso (um projeto de desenvolvimento de software), tendo como Unidade de Análise três engenheiros de software, seguindo as características de aplicação do modelo SCRUM-PSP-IF, apresentadas no capítulo três.

Foram coletados um total de cinquenta e quatro arquivos (documento e planilhas) dos três programadores, em três iterações, ou seja, seis arquivos por programador (*Logs* de Tempo e Defeitos e Resumo de Projeto, *Template* de Especificação Operacional, *Template* de Especificação Funcional, *Checklist* de Revisão de Projeto, *Checklist* de Revisão de Código e *Template* de Relatório de Teste.), em cada iteração. Esse número representa, aproximadamente, 25% dos materiais do PSP. Portanto, SCRUM-PSP-IF pode ser considerado como um modelo leve ou, minimamente, não pesado, seguindo a filosofia ágil. A equipe foi responsável por gerar os documentos de lançamento, *backlogs* do produto e das *sprints*, totalizando cinco arquivos. Salienta-se que esses últimos não foram analisados, mas serviram de base para viabilizar o projeto, pois sem os requisitos e o planejamento das tarefas a serem realizadas nas *sprints* (trabalho da equipe), cada programador não conseguiria realizar o seu próprio trabalho e, ainda, deve-se considerar que o objetivo da pesquisa é avaliar o trabalho dos indivíduos e não da equipe, por isso os arquivos coletados por estes não fazem parte da análise.

Para análise dos dados foi utilizada a análise estatística descritiva básica, complementar às métricas do próprio SCRUM-PSP-IF (*PQI*, *A/FR*, *Yield*, *Defect Density* e *Review Rate*), focando nas variáveis globais de defeitos, tempo e tamanho (serviram de base para calcular os índices de qualidade e demonstrar os resultados da pesquisa). Salienta-se que antes de analisar os dados, foi utilizada uma lista de verificação de consistência desses dados. Por fim, foram analisados os dados de defeitos, tempo e tamanho, mostrando a melhoria da qualidade no desenvolvimento no decorrer das iterações, através da diminuição da densidade de defeitos, o aumento da produtividade (em linhas de código), em menor tempo, no decorrer das iterações.

No relato dos resultados da pesquisa, os dados foram descritos de forma clara para que o leitor possa entender os objetivos da pesquisa, facilitando a interpretação dos resultados, compostos por cinco medidas de qualidade (índice de qualidade do processo - PQI, densidade de defeitos, taxa de revisão, rendimento e *appraisal to failure ratio*). Como PQI é uma medida formada por cinco componentes, pode-se analisá-los separadamente (até porque o PQI global, que consiste no produto dos componentes, não apresentou um índice satisfatório). Sendo assim, os componentes PQI DesignReviewTime e PQI UnitTestDefects obtiveram resultados extremamente satisfatórios, para todos os programadores, ao final das iterações. Inclusive, melhorando no decorrer das iterações. As medidas de rendimento e densidade de defeitos também se destacaram, positivamente, alcançando resultados extremamente satisfatórios.

5 CONCLUSÕES

Este capítulo tem o objetivo de explicitar as contribuições provenientes da realização desta pesquisa, bem como as oportunidades para trabalhos futuros. Nesse contexto, nas próximas seções serão descritas as contribuições do trabalho. Logo após, serão apresentadas algumas propostas para trabalhos futuros.

5.1 Contribuições da pesquisa

Os desafios no desenvolvimento de *software* moderno exigem tanto características de adequabilidade quanto de previsibilidade. Nesse cenário, utilizar práticas consolidadas do *PSP* para complementar as capacidades ágeis do *Scrum* parece ser uma ação bem coerente na busca pela melhoria da qualidade do produto de *software*.

O SCRUM-PSP-IF, um modelo integrado de processo de desenvolvimento, instanciado e aperfeiçoado de outro modelo existente (RONG et al., 2010), especificou claramente quais os elementos do *PSP* devem servir de recursos para cada desenvolvedor na construção dos seus produtos, não deixando de lado as atividades e artefatos dos métodos ágeis, no contexto da equipe (reuniões diárias, de planejamento, de retrospectiva, além das *sprints* e *backlog* do produto).

Partindo do pressuposto de que quem insere os erros nos programas são os próprios programadores, guiá-los de forma disciplinada, mesmo em projetos ágeis de desenvolvimento, que trabalham com prazos e exigências poucos negociáveis, melhora significativamente o desempenho dos engenheiros de *software* e, consequentemente, da equipe, pois reduz o número de erros presentes nos testes e também no produto final. SCRUM-PSP-IF disciplinou os programadores, utilizando elementos do *PSP* para melhorar o processo do método *Scrum*.

Combinar dois processos com filosofias conflitantes foi a primeira grande contribuição dessa pesquisa, pois enquanto o PSP é um processo de nível individual, o *Scrum* é um processo de nível de equipe. Se aproveitando dessa relação complementar, indivíduo e equipe, a pesquisa buscou reduzir os conflitos integrando esses dois processos.

Os resultados exitosos das medidas do índice de qualidade do processo (e seus quatro componentes), da densidade dos defeitos, da taxa de revisão, do rendimento e da relação de avaliação e falha representam as demais contribuições da pesquisa, produzindo constatações que confirmam que o uso do SCRUM-PSP-IF melhorou significativamente a qualidade no

desenvolvimento ágil de *software* no decorrer das iterações, através da integração do *Personal Software Process* com *Scrum*.

O uso do PQI definiu o índice de qualidade do processo de desenvolvimento através de quatro componentes. Globalmente, considerando o produto dos quatro componentes, pode-se dizer que os resultados não foram satisfatórios, pois nenhum programador alcançou o valor alvo da medida (0,4), em nenhuma das iterações. Porém, considerando os componentes separadamente, dois deles se destacaram positivamente, alcançando o valor máximo da medida (valor 1), pelos três programadores: revisão do projeto (componente $PQI_{DesignReviewTime}$) e qualidade do programa, representada pelo componente $PQI_{UnitTestDefects}$. A qualidade de revisão de código (componente $PQI_{CodeReviewTime}$) obteve índices relevantes nas iterações do programador A, praticamente alcançando 0,40 como resultado da medida. Já a qualidade do projeto, representada pelo quarto elemento ($PQI_{DesignCodeTime}$) apresentou índices praticamente zerados, considerando que o tempo de codificação foi muito maior que o tempo de projeto.

Quanto à medida de densidade dos defeitos, todos os programadores tiveram uma média de menos de 40 defeitos a cada 1000 linhas de código ao final das iterações, o que demonstrou um ótimo desempenho na redução do número de defeitos no produto final.

Quanto à medida de taxa de revisão de projeto todos os programadores tiveram ótimo desempenho, pois melhoraram consideravelmente o tempo de revisão de projeto (valor alvo de 30 minutos por página) da primeira iteração (com média de 30 minutos) para a terceira iteração, com uma média de 20 minutos por página. Porém, nas medidas da taxa de revisão de código apenas o programador A alcançou os valores alvo desejados, cerca de 200 LOC por hora de revisão. O que contribuiu fortemente para o sucesso desse índice de qualidade.

Quanto à medida de rendimento, que considera a quantidade de defeitos inseridos e removidos nas fases, o rendimento da fase de revisão do projeto obteve altos índices de qualidade, chegando a uma média de 90% de efetividade na remoção dos erros. A revisão de código alcançou valores relevantes, passando de 60% de efetividade. Já a fase de testes unitários teve desempenho máximo, removendo todos os defeitos inseridos no programa antes dos testes.

Por fim, quanto à medida que relaciona o valor da avaliação com o valor da falha (A/FR), demonstrou um ótimo desempenho para o programador A, que alcançou o valor alvo (2,0) na

média geral das iterações e um bom desempenho para o programador B, que alcançou a metade do valor alvo.

Portanto, todas as cinco medidas de qualidades do PSP, utilizadas pelo SCRUM-PSP, obtiveram índices satisfatórios de qualidade. Algumas, como dois dos componentes do PQI ($PQI_{DesignReviewTime}$ e $PQI_{UnitTestDefects}$), densidade de defeitos, taxa de revisão de projeto e rendimento obtiveram valores máximos para todos os programadores. Outras, como a taxa de revisão de código e A/FR obtiveram valores aceitáveis apenas para um ou dois programadores.

Algo que deve ser ressaltado dentre as contribuições desse trabalho é a inserção das fases de projeto, de revisão de projeto e de revisão de código no desenvolvimento ágil, se aproveitando dos *templates* e *checklists* de qualidade que o PSP disponibiliza (e que o SCRUM-PSP-IF utilizou). Uma grande porcentagem das equipes ágeis de desenvolvimento sequer utiliza a fase de projeto detalhado em seu ciclo de construção de sistemas e muito menos fases de revisão, pois transformam diretamente os requisitos do produto em código fonte (e dificilmente revisam esse código).

Somado a todas essas contribuições, pode-se dizer, ainda, que o SCRUM-PSP-IF facilitou muito a gestão do trabalho dos engenheiros de *software*, promovendo autogerenciamento e autodisciplina. Finalizando, o uso do PQI auxiliou a medir e controlar a qualidade do processo e avaliar a qualidade dos módulos produzidos.

5.2 Trabalhos futuros

O objetivo desse trabalho foi propor um modelo integrado de processos para a melhoria da qualidade no desenvolvimento de *software*, identificando as boas práticas de automelhoria e planejamento do PSP e de agilidade e simplicidade do *Scrum*, implantando o modelo integrado em um ambiente real de projeto. Nesse contexto, a pesquisa buscou obter índices de qualidade satisfatórios no processo de desenvolvimento de *software*, com foco na diminuição de defeitos no produto final.

Dentre os trabalhos futuros, que visam estender ou aprofundar os estudos aqui propostos, destacam-se os citados a seguir:

1. Utilizar estimativas no SCRUM-PSP-IF. A partir das primeiras iterações (no mínimo três iterações) coletando dados com o modelo integrado pode ser possível utilizar métodos de estimativas de tamanho, tempo e defeitos, como o PROBE.
2. Medir a qualidade do processo não só dos programadores, mas da equipe. Reproduzir para a equipe as medidas de qualidade dos indivíduos. Pode ser interessante integrar algumas práticas do TSP para esse estudo.
3. Aplicar o SCRUM-PSP-IF em múltiplos estudos de caso e realizar a comparação dos resultados.
4. Integrar e aplicar o PSP em outros processos ágeis conhecidos, como o XP, DSDM, Cristal, RUP, pois são poucos os estudos focados nesse contexto.
5. Ampliar as medidas do SCRUM-PSP-IF, utilizando outras medidas de qualidade do PSP, como *Development time ratios*, *Defect ratios*, *Defects per hour*, *Defect removal leverage*.

REFERÊNCIAS

ALBUQUERQUE, Jones de Oliveira, MEIRA, Silvio R. **PSP-JOA Processo de Software Pessoal - Uma abordagem orientada a JAVA**. Dissertação de Mestrado. Recife: Universidade Federal de Pernambuco, 1997, 141p.

KHAN, Abdul Kadir, "**Impact of Personal Software Process on software quality**," IOSR Journal of Computer Engineering, Vol. 1, Issue 5, May-Jun 2012.

ASSOCIAÇÃO PARA PROMOÇÃO DA EXCELÊNCIA DO SOFTWARE BRASILEIRO – SOFTEX. **MPS.BR – Guia Geral MPS de Software: 2016**. Disponível em: http://www.softex.br/wp-content/uploads/2013/07/MPS.BR_Guia_Geral_Software_2016.pdf.

ASSOCIAÇÃO PARA PROMOÇÃO DA EXCELÊNCIA DO SOFTWARE BRASILEIRO – SOFTEX. MPS.BR – **Guia Geral MPS de Software: 2012**. Disponível em: www.softex.br/mpsbr.

ASSOCIAÇÃO PARA PROMOÇÃO DA EXCELÊNCIA DO SOFTWARE BRASILEIRO – SOFTEX. MPS.BR – **Guia Geral MPS de Gestão de Pessoas: 2014** – versão Beta: novembro 2014. Disponível em: www.softex.br/mpsbr.

BOEHM, Barry, TURNER, Richard. **Balancing agility and discipline: a guide for the perplexed**. Addison-Wesley, 2004.

COLEMAN, G., VERBRUGGEN, R. **A quality software process for rapid application development**. Software Quality Journal 117, 107-117, 1998.

DZHUROV, Yani, **Personal Extreme Programming – An Agile Process for Autonomous Developers**. In Proceedings of International Conference on Software, Services & Semantic Technologies, Sofia, Bulgaria, 2009.

EASTERBROOK, S., SINGER, J., STOREY, M.A., DAMIAN, D. **Selecting Empirical Methods for Software Engineering Research. In Guide to Advanced Empirical Software Engineering**, Edited by Shull F., Singer J., Sjoberg, D.I.K., pp. 285-311, 2008.

GLAZER, Hillel, DALTON, Jeff, ANDERSON, David, KONRAD David J. M., SHRUM, Sandy. **CMMI® or Agile:Why Not Embrace Both**. TECHNICAL NOTE, CMU/SEI-2008-TN-003.

HUMPHREY, WATTS S. **A discipline for Software Engineering**, Reading, MA: Addison-Wesley Publishing Company, 1995, 789 p.

HUMPHREY, WATTS S. **Introduction to the Personal Software Process**, Reading, MA: Addison-Wesley Publishing Company, 1997, 278 p.

HUMPHREY, Watts. **The Personal Software Process (CMU/SEI-2000-TR-022)**, Pittsburghh, PA: SEI, Carnegie Mellon University, 2000a.

HUMPHREY, Watts. **The Team Software Process (CMU/SEI-2000-TR-023)**, Pittsburghh, PA: SEI, Carnegie Mellon University, 2000b.

HUMPHREY, Watts S. PSP: **A Self-Improvement Process for Software Engineers**. Boston, MA: Addison-Wesley Publishers, 2005 (ISBN: 0321305493).

ILIEVA, S., STEFANOVA, E .**Expert approach for e-business software development**. In Proceedings of International conference Basic Technologies for E-business, Albena, 2002.

ISO/IEC. International Organization for Stardarization/International Electrotechnical Comission. **ISO/IEC 12207: Systems and software engineering -- Software life cycle processes**, Geneve: 2008.

ISO/IEC. International Organization for Stardarization/International Electrotechnical Comission. **ISO/IEC 15504-6: Information technology -- Process assessment - Part 6: An exemplar system life cycle process assessment model**, Geneve: 2013.

KEMERER, Chris F., PAULK, Mark C., **The Impact of Design and Code Reviews on Software Quality: An Empirical Study Based on PSP Data**, IEEE transactions on software engineering, Vol 35 no.4, 2009.

KOSCIANSKI, André; SOARES, Michel Santos. **Qualidade de Software**. 2.ed. São Paulo: Novatec, 2007.

LI, Hui, TAO, Peiji, LI, Wenfeng. **Combining XP and RUP to develop small projects**. Computer Engineering & Design 2005.

LYNCH, Jennifer. **Standish Group 2015 Chaos Report - Q&A with Jennifer Lynch**. Disponível em: <https://www.infoq.com/articles/standish-chaos-2015>. Acesso em: maio de 2016.

MIHAYLOV, Ilian. **The expert approach – a case study**, CompSysTech '03 Proceedings of the 4th international conference on Computer systems and technologies: e-Learning.

POMEROY-HUFF, M. et al. **The Personal Software Process (PSP) body of knowledge**. Technical Report, CMU/SEI-2009-SR-018, v.2.0, ago. 2009.

PRESSMAN, R. S. **Engenharia de Software**. 6. ed. São Paulo: Mc Graw Hill, 2006.

RONG, Guoping, SHAO, Dong e ZHABG, He. **SCRUM-PSP: Embracing Process Agility and Discipline**, Software Engineering Conference (ASPEC), 2010 17th Asia Pacific.

RUNESON, P. e HOST, M.. **Guidelines for conducting and reporting case study research in software engineering**. Empirical Software Engineering 14(2). The most cited EMSE paper ever., 131–164, 2009.

SANDE, Deysiane. **A Strategy to Support Software Planning Based on Piece of Work and Agile Paradigm**, 2011.

SEI. Software Engineering Institute. **CMMI for Development (CMMI-DEV), Version 1.3, Technical Report CMU/SEI-2010-TR-033**. Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2010.

SEI. Software Engineering Institute. **Self-Study PSP Material**, 2006. Disponível em: <http://www.sei.cmu.edu/tsp/tools/student/>. Acesso em: janeiro de 2016.

SHEN, Mengjiao, RONG, Guoping e SHAO, Dong. **Integrating PSP with agile process: a systematic review**, 2th Internatinal Conference On Systems Engineering and Modeling. Atlantis Press, Paris, France, 2013.

SOMMERVILLE, I. **Engenharia de Software**. 9. ed. São Paulo: Pearson Prentice Hall, 2011.

SVENSSON, Harald, **Developing Support for Agile and Plan-Driven Methods**, KTH, Sweden. 2001.

SUPHAK, Suwanya, WERASAK, Kurutach. **Applying Agility Frameworkin Small and Medium Enterprises**, 2011.

TURK, D., FRANCE, R., RUMPE, B. **Agile Software Processes: Principles, Assumptions and Limitations**. Technical Report. Colorado State University, 2002.

VERSIONONE. 2015. **10th Annual State of Agile Report**. Disponível em: <https://explore.versionone.com/state-of-agile/versionone-10th-annual-state-of-agile-report-2>. Acesso em: maio de 2016.

VERSIONONE. 2016. **11th Annual State of Agile Report**. Disponível em: < https://explore.versionone.com/state-of-agile/versionone-11th-annual-state-of-agile-report-2>. Acesso em: abril de 2017.

WOHLIN, Claes; AURUM, A. **Towards a Decision-making Structure for Selecting a Research Design in Empirical Software Engineering**, Empirical Software Engineering: An International Journal, Vol. 20. No. 6, pp. 1427-1455, 2015.

WILLIAMS, L., **Integrating Pair Programming into a Software Development Process**, Software Engineering Education and Training. Proceedings.14th Conference on Software Engineering Education and Training , USA, 2001.

WENGRONG, Yang e SHEN, Mengjiao. **Investigating the Benefits of Combining PSP with Agile Software Development**, EAST2011.

YIN, R. K. **Case Study Research, Design and Methods**, 3rd ed. Newbury Park, Sage Publications, 2002.

APÊNDICE A – Log de registro de tempo – Iteração 1 – Programador A

Programador	Programador A	**Data**	30/01/2017 - 03/02/2017
Sistema	SIPS	**Programa/Módulo**	Manter Instituição
Instrutor	*Scrum Master*	**Linguagem**	PHP

Projeto	Fase	Data e Tempo de Inicio	Tempo de Interrupção	Data e Tempo de Fim	Tempo Delta	Comentários
SIPS	Planejamento	30/01/2017 - 08:30	15	30/01/2017 - 09:00	15	Reunião diária de 15min. Elicitação do RQF-20.
SIPS	Projeto	30/01/2017 - 09:00	0	30/01/2017 - 10:25	85	Prrenchimento dos templates de especificação operacional e funcional.
SIPS	Revisão do Projeto	30/01/2017 - 10:25	0	30/01/2017 - 11:25	60	Utilização do checklist de revisão de projeto.
SIPS	Codificação	30/01/2017 - 11:25	35	30/01/2017 - 13:55	115	Outras atividades de 35min. Criação da Classe instituicao.class.php
SIPS	Codificação	31/02/2017 - 08:20	70	31/02/2017 - 13:40	140	Reunião diária de 15min. Check mail e outras atividades. Classe Instituicao.class.php (133 LOC)
SIPS	Codificação	01/02/2017 - 08:05	30	01/02/2017 - 14:00	205	Reunião diária de 30min. index_instituicao.php; cadastrar_instituicao.php; exibir_insituicao.php; editar_instituicao.php; excluir_instituicao.php
SIPS	Codificação	02/02/2017 - 08:10	30	02/02/2017 - 11:00	120	Reunião diária de 15min. index_instituicao.php; cadastrar_instituicao.php; exibir_insituicao.php; editar_instituicao.php; excluir_instituicao.php (206 LOC)
SIPS	Revisão da Codificação	02/02/2017 - 11:00	40	02/02/2017 - 13:25	105	Outras atividades de 40min. Verificação de e-mails. Utilização do checklist de revisão de código.
SIPS	Compilação	02/02/2017 - 13:25	0	02/02/2017 - 13:30	5	
SIPS	Teste	03/02/2017 - 08:10	15	03/02/2017 - 10:00	95	Reunião diária de 15min
SIPS	Postmortem	03/02/2017 - 10:00	10	03/02/2017 - 12:00	110	Preencimento do resumo de projeto com os totais da iteração 1.

APÊNDICE B – Log de registro de defeitos – Iteração 1 – Programador A

Programador	Programador A	**Data**	30/01/2017 - 03/02/2017
Sistema	SIPS	**Programa/Módulo**	Manter Instituição
Instrutor	*Scrum Master*	**Linguagem**	PHP

Projeto	Data	Número	Tipo	Injetado	Removido	Tempo correção	Ref.
SIPS	30/01/2017	1	10	Projeto	Revisão do Projeto	5min	-

Descrição: Fluxo alternativo mal definido na etapa 14. Informar mensagem de erro (aviso).

Projeto	Data	Número	Tipo	Injetado	Removido	Tempo correção	Ref.
SIPS	02/02/2017	2	70	Codificação	Revisão do Código	10min	-

Descrição: Sistema não exibiu mensagem adequada de êxito no novo cadastro. Exibir "Instituição cadastrada com sucesso."

Projeto	Data	Número	Tipo	Injetado	Removido	Tempo correção	Ref.
SIPS	03/02/2017	3	100	Projeto	Testes	10min	-

Descrição: O campo observações da Instituição não deve ser obrigatório.

Projeto	Data	Número	Tipo	Injetado	Removido	Tempo correção	Ref.
SIPS	03/02/2017	4	100	Projeto	Testes	10min	-

Descrição: Utilizar máscara para CEP, separando os 5 primeiros números dos três últimos utilizando hífen.

Projeto	Data	Número	Tipo	Injetado	Removido	Tempo correção	Ref.
SIPS	30/01/2017	5	10	Projeto	Revisão do Projeto	5min	-

Descrição: Ausência da classe pai.

Projeto	Data	Número	Tipo	Injetado	Removido	Tempo correção	Ref.
SIPS	03/02/2017	6	10	Projeto	Testes	10min	-

Descrição: CEP com tipo de dados int ao invés de string.

Projeto	Data	Número	Tipo	Injetado	Removido	Tempo correção	Ref.
SIPS	03/02/2017	7	10	Projeto	Testes	30min	-

Descrição: Método excluirInstituição recebe parâmetro de ID da instituição, retornando bool.

Projeto	Data	Número	Tipo	Injetado	Removido	Tempo correção	Ref.

Projeto	Data	Número	Tipo	Injetado	Removido	Tempo correção	Ref.
SIPS	02/02/2017	8	80	Codificação	Revisão do código	20min	-

Descrição: Função para atualizar (editar) instituição deve receber variável id_instituicao como parâmetro.

Projeto	Data	Número	Tipo	Injetado	Removido	Tempo correção	Ref.
SIPS	02/02/2017	9	10	Codificação	Revisão do código	15min	-

Descrição: SQL com mais de 200 caracteres na linha 73, arquivo instituicao.class.php.

Projeto	Data	Número	Tipo	Injetado	Removido	Tempo correção	Ref.
SIPS	03/02/2017	10	10	Codificação	Testes	15min	-

Descrição: Faltou incluir a classe de gerenciamento de Instituicao (Instituicao.class.php). Arquivo cadastrar.php.

Projeto	Data	Número	Tipo	Injetado	Removido	Tempo correção	Ref.
SIPS	03/02/2017	11	40	Codificação	Testes	5min	-

Descrição: Método excluirInstituição recebe parâmetro de ID da instituição e retorna bool.

Projeto	Data	Número	Tipo	Injetado	Removido	Tempo correção	Ref.
SIPS	02/02/2017	12	20	Codificação	Revisão do código	10min	-

Descrição: Arquivo exibir.php: caminho do include não encontrado.

Projeto	Data	Número	Tipo	Injetado	Removido	Tempo correção	Ref.
SIPS	02/02/2017	13	40	Codificação	Revisão do código	5min	-

Descrição: Arquivo excluir.php: faltou incluir a classe pai generic.class.php.

Projeto	Data	Número	Tipo	Injetado	Removido	Tempo correção	Ref.
SIPS	02/02/2017	14	60	Codificação	Revisão do código	5min	-

Descrição: Tag html center obsoleta. Substituir por CSS.

APÊNDICE C – Resumo de plano de projeto – Iteração 1 – Programador A

Programador	Programador A	**Data**	30/01/2017 - 03/02/2017
Sistema	SIPS	**Programa/Módulo**	Manter Instituição
Instrutor	*Scrum Master*	**Linguagem**	PHP

Tamanho do Programa	**Atual**	**Até a Data**
Tamanho Total (T)	339 LOC	339 LOC

Tempo na Fase (min.)	**Real**	**Até a Data**
Planejamento	15	15
Projeto	85	85
Revisão de Projeto	60	60
Código	580	580
Revisão do Código	105	105
Compilação	5	5
Teste	95	95
Postmortem	110	110
Total	1055	1055

Defeitos Injetados	**Real**	**Até a Data**
Planejamento	0	0
Projeto	6	6
Revisão de Projeto	0	0
Código	8	8
Revisão do Código	0	0
Compilação	0	0
Teste	0	0
Total	14	14

Defeitos Removidos		Real	Até a Data
Planejamento		0	0
Projeto		0	0
Revisão de Projeto		2	2
Código		0	0
Revisão do Código		6	6
Compilação		0	0
Teste		6	6
Total	14	14	

Resumo de Qualidade	PQI - Process Quality Process			
	Tempo projeto	**Tempo de Codificação**	**TOTAL**	
$PQI_{DesignCodeTime}$	85	580	**0,1466**	Qualidade do projeto, sendo expressa pela divisão entre o tempo de projeto e o tempo de codificação. É recomendado que o tempo gasto na fase de projeto seja maior que o tempo gasto na fase de codificação.
	Tempo revisão projet	**Tempo de projeto**	**TOTAL**	
$PQI_{DesignReviewTime}$	60	85	**1,4118**	Qualidade de revisão de projeto, sendo expressa pela divisão entre o dobro do tempo de revisão de projeto e o tempo de projeto. É recomendado que o tempo gasto na fase de revisão de projeto seja, no mínimo, metade do tempo gasto na fase de projeto.
	Tempo revisão código	**Tempo de códificação**	**TOTAL**	
$PQI_{CodeReviewTime}$	105	580	**0,3621**	Qualidade de revisão de código, sendo expressa pela divisão entre o dobro do tempo de revisão de código e o tempo de codificação. É recomendado que o tempo gasto na fase de revisão de código seja, no mínimo, a metade do tempo gasto na fase de codificação.
	Defeitos per KLOC testes unitários		**TOTAL**	
$PQI_{UnitTestDefects}$	17		0,45455	Qualidade do programa, sendo expressa pela divisão de 10 pela soma do número de defeitos (encontrados nos testes unitários a cada 1000 linhas de código) com 5, que representa o valor alvo para essa métrica, ou seja, a densidade de defeitos da fase de testes unitários não deve ser superior a 5 defeitos/KLOC.

APÊNDICE D – Log de registro de tempo – Iteração 1 – Programador B

Programador	Programador B	**Data**	30/01/2017 - 03/02/2017
Sistema	SIPS	**Programa/Módulo**	Manter Processo Seletivo
Instrutor	*Scrum Master*	**Linguagem**	PHP

Projeto	Fase	Data e Tempo de Inicio	Tempo de Interrupção	Data e Tempo de Fim	Tempo Delta	Comentários
SIPS	Planejamento	30/01/2017 - 08:10	25	30/01/2017 - 09:30	55	Reunião Scrum. 15min. Responder alguns e-mails. 10min.
SIPS	Projeto	30/01/2017 - 09:30	10	30/01/2017 - 11:20	100	Outras tarefas. 10min. Criação do projeto operacional e funcional.
SIPS	Revisão do Projeto	30/01/2017 - 11:20	0	30/01/2017 - 12:35	75	Lista de verificação de revisão de projeto.
SIPS	Codificação	30/01/2017 - 12:35	5	30/01/2017 - 14:00	80	Codificação da classe ProcessoSeletivo.class.php.
SIPS	Codificação	31/02/2017 - 08:00	60	31/02/2017 - 13:30	270	Reunião Scrum. 15min. Responder e-mails. 15min. Suporte a sistemas. 30min. Continuação da codificação Classe ProcessoSeletivo.class.php. (198 LOC)
SIPS	Codificação	01/02/2017 - 08:10	90	01/02/2017 - 14:00	260	Reunião Scrum. 30min. Responder e-mails. 20min. Suporte a outros sistemas. 40min. Codificação dos arquivos index_ps.php; cadastrar_ps.php; exibir_ps.php; editar_ps.php; excluir_ps.php.
SIPS	Codificação	02/02/2017 - 08:10	30	02/02/2017 - 14:00	320	Reunião diária de 15min. Responder e-mails. 15min. Continuação da codificação dos arquivos index_ps.php; cadastrar_ps.php; exibir_ps.php; editar_ps.php; excluir_ps.php (310 LOC).
SIPS	Revisão da Codificação	03/02/2017 - 08:00	20	03/02/2017 - 09:30	70	Reunião Scrum. 15min. Outras atividades. 5min. Checklist de revisão de código.
SIPS	Compilação	03/02/2017 - 09:30	0	03/02/2017 - 09:35	5	
SIPS	Teste	03/02/2017 - 09:35	10	03/02/2017 - 11:40	115	Verificação de e-mail. 10min. Utilização do template de teste.
SIPS	Postmortem	03/02/2017 - 11:40	0	03/02/2017 - 14:00	110	Preencimento do formulário (planilha) de resumo de projeto.

APÊNDICE E – Log de registro de defeitos – Iteração 1 – Programador B

Programador	Programador B	**Data**	30/01/2017 - 03/02/2017
Sistema	SIPS	**Programa/Módulo**	Manter Processo Seletivo
Instrutor	*Scrum Master*	**Linguagem**	PHP

Projeto	Data	Número	Tipo	Injetado	Removido	Tempo correção	Ref.
SIPS	03/02/2017	1	10	Projeto	Testes	25min	-

Descrição: Ausência de pesquisa se o processo seletivo está ou não ativo, considerando a instituição.

Projeto	Data	Número	Tipo	Injetado	Removido	Tempo correção	Ref.
SIPS	30/01/2017	2	10	Projeto	Revisão do Projeto	5min	-

Descrição: Na etapa 10 seleciona a opção editar e não na etapa 08.

Projeto	Data	Número	Tipo	Injetado	Removido	Tempo correção	Ref.
SIPS	03/02/2017	3	60	Projeto	Testes	10min	-

Descrição: A mensagem de erro da etapa 18 deve estar condicionada ao processo seletivo ativo.

Projeto	Data	Número	Tipo	Injetado	Removido	Tempo correção	Ref.
SIPS	03/02/2017	4	10	Codificação	Testes	10min	-

Descrição: Faltou listar as heranças de Generic.class.php e Db.class.php.últimos utilizando hífen.

Projeto	Data	Número	Tipo	Injetado	Removido	Tempo correção	Ref.
SIPS	03/02/2017	5	10	Codificação	Revisão do Código	15min	-

Descrição: O método Listar todos os processos seletivos não deve receber parâmetro.

Projeto	Data	Número	Tipo	Injetado	Removido	Tempo correção	Ref.
SIPS	03/02/2017	6	80	Codificação	Revisão do Código	5min	-

Descrição: pesquisarPSporNome recebe o ID do processo (não o nome). Porém deve retornar o nome do processo (ou mensagem de processo não encontrado).

Projeto	Data	Número	Tipo	Injetado	Removido	Tempo correção	Ref.
SIPS	03/02/2017	7	40	Codificação	Testes	20min	-

Descrição: Iniciar a sessão de usuário para incluir a classe de usuário e possibilitar a criação do objeto (index_ps.php).

Projeto	Data	Número	Tipo	Injetado	Removido	Tempo correção	Ref.
SIPS	03/02/2017	8	80	Codificação	Revisão do código	5min	-

Descrição: Incluir a classe de processo seletivo no index_ps.php.

Projeto	Data	Número	Tipo	Injetado	Removido	Tempo correção	Ref.
SIPS	03/02/2017	9	50	Codificação	Revisão do código	15min	-

Descrição: Deve ser usado unset para esvaziar o vetor $_post. (cadastrar_ps.php).

Projeto	Data	Número	Tipo	Injetado	Removido	Tempo correção	Ref.
SIPS	03/02/2017	10	10	Codificação	Testes	5min	-

Descrição: $_POST[ano_p_seletivo] deve ser utilizado para realizar update do processo. (editar_ps.php)

Projeto	Data	Número	Tipo	Injetado	Removido	Tempo correção	Ref.
SIPS	03/02/2017	11	40	Codificação	Testes	10min	-

Descrição: Se a coleta os dados do BD (ID passado como parâmetro pela URL) não encontrar dados, envia isso para a mensagem de erro getMessage().Arquivo Excluir.php.

Projeto	Data	Número	Tipo	Injetado	Removido	Tempo correção	Ref.
SIPS	03/02/2017	12	20	Codificação	Revisão do código	5min	-

Descrição: Incluir db.in.php e db.class.php (generic foi incluída).

APÊNDICE F – Resumo de plano de projeto – Iteração 1 – Programador B

Programador	Programador B	**Data**	30/01/2017 - 03/02/2017
Sistema	SIPS	**Programa/Módulo**	Manter Processo Seletivo
Instrutor	*Scrum Master*	**Linguagem**	PHP

Tamanho do Programa	**Atual**	**Até a Data**
Tamanho Total (T)	508 LOC	508 LOC

Tempo na Fase (min.)	**Real**	**Até a Data**
Planejamento	55	55
Projeto	100	100
Revisão de Projeto	75	75
Codificação	930	930
Revisão do Código	70	70
Compilação	5	5
Teste	115	115
Postmortem	110	110
Total	1460	1460

Defeitos Injetados	**Real**	**Até a Data**
Planejamento	0	0
Projeto	3	3
Revisão de Projeto	0	0
Codificação	9	9
Revisão do Código	0	0
Compilação	0	0
Teste	0	0

Total	12	12

Defeitos Removidos	**Real**	**Até a Data**
Planejamento	0	0
Projeto	0	0
Revisão de Projeto	1	1
Codificação	0	0
Revisão do Código	5	5
Compilação	0	0
Teste	6	6
Total	12	12

Resumo de Qualidade	**PQI - Process Quality Process**			
	Tempo Projeto	**Tempo de Codificação**	**TOTAL**	
$PQI_{DesignCodeTime}$	100	930	**0,107526882**	Qualidade do projeto, sendo expressa pela divisão entre o tempo de projeto e o tempo de codificação. É recomendado que o tempo gasto na fase de projeto seja maior que o tempo gasto na fase de codificação.
	Tempo Revisão Projeto	**Tempo de Projeto**	**TOTAL**	
$PQI_{DesignReviewTime}$	75	100	**1,5**	Qualidade de revisão de projeto, sendo expressa pela divisão entre o dobro do tempo de revisão de projeto e o tempo de projeto. É recomendado que o tempo gasto na fase de revisão de projeto seja, no mínimo, metade do tempo gasto na fase de projeto.
	Tempo Revisão Código	**Tempo de Codificação**	**TOTAL**	
$PQI_{CodeReviewTime}$	70	930	**0,150537634**	Qualidade de revisão de código, sendo expressa pela divisão entre o dobro do tempo de revisão de código e o tempo de codificação. É recomendado que o tempo gasto na fase de revisão de código seja, no mínimo, a metade do tempo gasto na fase de codificação.
	Defeitos per KLOC testes unitários		**TOTAL**	
$PQI_{UnitTestDefects}$	11		0,625	Qualidade do programa, sendo expressa pela divisão de 10 pela soma do número de defeitos (encontrados nos testes unitários a cada 1000 linhas de código) com 5, que representa o valor alvo para essa métrica, ou seja, a densidade de defeitos da fase de testes unitários não deve ser superior a 5 defeitos/KLOC.

APÊNDICE G – Log de registro de tempo – Iteração 1 – Programador C

Programador	Programador C	**Data**	30/01/2017 - 03/02/2017
Sistema	SIPS	**Programa/Módulo**	Manter Usuário
Instrutor	*Scrum Master*	**Linguagem**	PHP

Projeto	Fase	Data e Tempo de Inicio	Tempo de Interrupção	Data e Tempo de Fim	Tempo Delta	Comentários
SIPS	Planejamento	30/01/2017 - 08:00	20	30/01/2017 - 08:40	20	Reunião (15min). Checar e-mails (5min).
SIPS	Projeto	30/01/2017 - 08:40	0	30/01/2017 - 10:00	80	Especificação operacional e funcional.
SIPS	Revisão do Projeto	30/01/2017 - 10:00	5	30/01/2017 - 10:55	50	Lista de verificação de revisão de projeto.
SIPS	Codificação	30/01/2017 - 10:55	30	30/01/2017 - 14:00	145	Início da criação da Classe Usuario.class.php.
SIPS	Codificação	31/02/2017 - 08:15	40	31/02/2017 - 13:45	290	Reunião (15min). Checar-mails (10min). Suporte a usuários (15min). Continuação da codificação Classe Usuario.class.php (150 LOC)
SIPS	Codificação	01/02/2017 - 08:00	90	01/02/2017 - 13:55	265	Reunião (30min). Checar e-mails (15min). Suporte a usuários (45min). Início da criação dos arquivos index_usuario.php; cadastrar_usuario.php; editar_usuario.php e excluir_usuario.php.
SIPS	Codificação	02/02/2017 - 08:20	30	02/02/2017 - 14:00	310	Reunião (15min). Checar e-mails (10min). Telefonema (5min). Continuação da criação dos arquivos index_usuario.php; cadastrar_usuario.php; editar_usuario.php e excluir_usuario.php (292 LOC)
SIPS	Revisão da Codificação	03/02/2017 - 08:00	20	03/02/2017 - 09:05	45	Reunião (15min). Checar e-mail (5min). Checklist de revisão de código.
SIPS	Compilação	03/02/2017 - 09:05	0	03/02/2017 - 09:10	5	
SIPS	Teste	03/02/2017 - 09:10	25	03/02/2017 - 11:40	125	Suporte a usuários (25min). Utilização do template de teste.
SIPS	Postmortem	03/02/2017 - 11:40	5	03/02/2017 - 13:50	125	Telefonema (5min). Preencimento da planilha de resumo de projeto.

APÊNDICE H – Log de registro de defeitos – Iteração 1 – Programador C

Programador	Programador C	**Data**	30/01/2017 - 03/02/2017
Sistema	SIPS	**Programa/Módulo**	Manter Usuário
Instrutor	*Scrum Master*	**Linguagem**	PHP

Projeto	Data	Número	Tipo	Injetado	Removido	Tempo correção	Ref.
SIPS	30/01/2017	1	10	Projeto	Revisão do Projeto	5min	-

Descrição: Adicionar consluta por situação à etapa 7 do template de especificação operacional.

Projeto	Data	Número	Tipo	Injetado	Removido	Tempo correção	Ref.
SIPS	30/01/2017	2	10	Projeto	Revisão do Projeto	10min	-

Descrição: Não projetar o log ajuntamento com a classe usuário.

Projeto	Data	Número	Tipo	Injetado	Removido	Tempo correção	Ref.
SIPS	03/02/2017	3	80	Projeto	Revisão do Código	5min	-

Descrição: consultarSituaçãoUsuario deve receber o ID do usuário e não a situação do usuário.

Projeto	Data	Número	Tipo	Injetado	Removido	Tempo correção	Ref.
SIPS	03/02/2017	4	10	Projeto	Testes	15min	-

Descrição: criarUsuario e atualizarUsuario devem receber os mesmos parâmetros (os 8 atributos de usuário).

Projeto	Data	Número	Tipo	Injetado	Removido	Tempo correção	Ref.
SIPS	03/02/2017	5	80	Codificação	Revisão do Código	10min	-

Descrição: Retorno do tipo booleano para os métodos criar e atualizar usuário.

Projeto	Data	Número	Tipo	Injetado	Removido	Tempo correção	Ref.

SIPS	03/02/2017	6	80	Codificação	Revisão do Código	5min	-

Descrição: Método Atualizar (editar) usuário não recebe o ID como parâmetro.

Projeto	Data	Número	Tipo	Injetado	Removido	Tempo correção	Ref.
SIPS	03/02/2017	7	80	Codificação	Revisão do Código	5min	-

Descrição: Método inserir usuário não recebe o ID como parâmetro.

Projeto	Data	Número	Tipo	Injetado	Removido	Tempo correção	Ref.
SIPS	03/02/2017	8	80	Codificação	Testes	5min	-

Descrição: Na função pública de criarUsuario devem ser feitos dois testes antes do insert: um testando se o login já existe (através da função consultarUsuarioPorSiape) e outro testando se o e-mail já existe (através da função consultarUsuarioPorEmail). Usuario.class.php

Projeto	Data	Número	Tipo	Injetado	Removido	Tempo correção	Ref.
SIPS	03/02/2017	9	70	Codificação	Testes	5min	-

Descrição: Incluir os caminhos do DEFINE: include("../includes/define.inc.php"); cadastrar_usuario.php

Projeto	Data	Número	Tipo	Injetado	Removido	Tempo correção	Ref.
SIPS	03/02/2017	10	10	Codificação	Testes	5min	-

Descrição: Ausência da classe genérica para gerenciamento de erros.

APÊNDICE I – Resumo de plano de projeto – Iteração 1 – Programador C

Programador	Programador C	**Data**	30/01/2017 - 03/02/2017
Sistema	SIPS	**Programa/Módulo**	Manter Usuário
Instrutor	*Scrum Master*	**Linguagem**	PHP

Tamanho do Programa	**Atual**	**Até a Data**
Tamanho Total (T)	342 LOC	342 LOC

Tempo na Fase (min.)	**Real**	**Até a Data**
Planejamento	20	20
Projeto	80	80
Revisão de Projeto	50	50
Codificação	1010	1010
Revisão do Código	45	45
Compilação	5	5
Teste	125	125
Postmortem	125	125
Total	1460	1460

Defeitos Injetados	**Real**	**Até a Data**
Planejamento	0	0
Projeto	4	4
Revisão de Projeto	0	0
Codificação	6	6
Revisão do Código	0	0
Compilação	0	0

Teste	0	0
Total	10	10

Defeitos Removidos	**Real**	**Até a Data**
Planejamento	0	0
Projeto	0	0
Revisão de Projeto	2	2
Codificação	0	0
Revisão do Código	4	4
Compilação	0	0
Teste	4	4
Total	10	10

Resumo de Qualidade	**PQI - Process Quality Process**			
	Tempo Projeto	**Tempo de Codificação**	**TOTAL**	Qualidade do projeto, sendo expressa pela divisão entre o tempo de projeto e o tempo de codificação. É recomendado que o tempo gasto na fase de projeto seja maior que o tempo gasto na fase de codificação.
$PQI_{DesignCodeTime}$	80	1010	**0,079207921**	
	Tempo Revisão Projeto	**Tempo de Projeto**	**TOTAL**	Qualidade de revisão de projeto, sendo expressa pela divisão entre o dobro do tempo de revisão de projeto e o tempo de projeto. É recomendado que o tempo gasto na fase de revisão de projeto seja, no mínimo, metade do tempo gasto na fase de projeto.
$PQI_{DesignReviewTime}$	50	80	**1,25**	
	Tempo Revisão Código	**Tempo de Codificação**	**TOTAL**	Qualidade de revisão de código, sendo expressa pela divisão entre o dobro do tempo de revisão de código e o tempo de codificação. É recomendado que o tempo gasto na fase de revisão de código seja, no mínimo, a metade do tempo gasto na fase de codificação.
$PQI_{CodeReviewTime}$	45	1010	**0,089108911**	
	Defeitos per KLOC testes unitários		**TOTAL**	Qualidade do programa, sendo expressa pela divisão de 10 pela soma do número de defeitos (encontrados nos testes unitários a cada 1000 linhas de código) com 5, que representa o valor alvo para essa métrica, ou seja, a densidade de defeitos da fase de testes unitários não deve ser superior a 5 defeitos/KLOC.
$PQI_{UnitTestDefects}$	11		0,625	

APÊNDICE J – Log de registro de tempo – Iteração 2 – Programador A

Programador	Programador A	**Data**	10/02/2017
Sistema	SIPS	**Programa/Módulo**	Manter Curso
Instrutor	*Scrum Master*	**Linguagem**	PHP

Projeto	Fase	Data e Tempo de Inicio	Tempo de Interrupção	Data e Tempo de Fim	Tempo Delta	Comentários
SIPS	Planejamento	06/02/2017 - 08:00	15	06/02/2017 - 09:00	45	Reunião diária de 15min. Elicitação do RQF-22.
SIPS	Projeto	06/02/2017 - 09:00	20	06/01/2017 - 11:10	110	Outras atividades de 20min. Prrenchimento dos templates de especificação operacional e funcional.
SIPS	Revisão do Projeto	06/02/2017 - 11:00	0	06/02/2017 - 12:30	90	Utilização do checklist de revisão de projeto.
SIPS	Codificação	06/02/2017 - 12:30	0	06/02/2017 - 13:50	80	Criação da classe Curso.class.php
SIPS	Codificação	07/02/2017 - 08:05	70	07/02/2017 - 13:45	140	Reunião diária de 15min. Classe Curso.class.php.
SIPS	Codificação	08/02/2017 - 08:00	50	08/02/2017 - 14:00	310	Reunião diária de 15min. Outras atividades de 35min. Conclusão classe Curso.class.php (158 LOC). Início da criação do index_curso.php; cadastrar_curso.php; exibir_curso.php; editar_curso.php; excluir_curso.php
SIPS	Codificação	09/02/2017 - 08:10	30	09/02/2017 - 13:55	315	Reunião diária de 15min. Outras atividades de 15min. index_curso.php; cadastrar_curso.php; exibir_curso.php; editar_curso.php; excluir_curso.php (240 LOC)
SIPS	Revisão da Codificação	10/02/2017 - 08:20	30	10/02/2017 - 11:45	175	Reunião diária de 15min. Outras atividades de 15min. Utilização da checklist de revisão de código.
SIPS	Compilação	10/02/2017 - 11:45	0	10/02/2017 - 11:50	5	
SIPS	Teste	10/02/2017 - 11:50	10	10/02/2017 - 13:15	75	Outras atividades 10min. Utilização do template de testes.
SIPS	Postmortem	10/02/2017 - 13:15	0	10/02/2017 - 14:00	45	Preencimento do resumo de projeto com os totais da iteração 1.

APÊNDICE L – Log de registro de defeitos – Iteração 2 – Programador A

Programador	Programador A	**Data**	10/02/2017
Sistema	SIPS	**Programa/Módulo**	Manter Curso
Instrutor	*Scrum Master*	**Linguagem**	PHP

Projeto	Data	Número	Tipo	Injetado	Removido	Tempo correção	Ref.
SIPS	06/02/2017	1	10	Projeto	Revisão do Projeto	5min	-

Descrição: Faltou considerar instituição no cadastro de cursos.

Projeto	Data	Número	Tipo	Injetado	Removido	Tempo correção	Ref.
SIPS	06/02/2017	2	10	Projeto	Revisão do Projeto	5min	-

Descrição: Fluxo alternativo da etapa 8 da especificação operacional fora do contexto.

Projeto	Data	Número	Tipo	Injetado	Removido	Tempo correção	Ref.
SIPS	06/02/2017	3	10	Projeto	Revisão do Projeto	5min	-

Descrição: Falta das vagas para portadores de deficiências.

Projeto	Data	Número	Tipo	Injetado	Removido	Tempo correção	Ref.
SIPS	06/02/2017	4	10	Projeto	Revisão do Projeto	5min	-

Descrição: Faltou associar a instituição a classe curso.

Projeto	Data	Número	Tipo	Injetado	Removido	Tempo correção	Ref.
SIPS	06/02/2017	5	10	Projeto	Revisão do Projeto	5min	-

Descrição: Etapas 9 e 10 com ações apenas do administrador, sem iteração com o SIPS.

Projeto	Data	Número	Tipo	Injetado	Removido	Tempo correção	Ref.
SIPS	10/02/2017	6	70	Codificação	Revisão do código	10min	-

Descrição: Ausência de Db.class. Curso.class.php.

Projeto	Data	Número	Tipo	Injetado	Removido	Tempo correção	Ref.
SIPS	10/02/2017	7	10	Projeto	Revisão do código	5min	-

Descrição: Método listarTodosUsuarios não recebe o ID do usuário.

Projeto	Data	Número	Tipo	Injetado	Removido	Tempo correção	Ref.

Projeto	Data	Número	Tipo	Injetado	Removido	Tempo correção	Ref.
SIPS	10/02/2017	8	40	Codificação	Testes	10min	-

Descrição: Incluir a classe Instituicao.class.php no arquivo cadastrar_curso.php.

Projeto	Data	Número	Tipo	Injetado	Removido	Tempo correção	Ref.
SIPS	10/02/2017	9	80	Codificação	Revisão do código	15min	-

Descrição: Utilizar função de consultar ID do curso para localizar o curso a ser editado ou excluído. Caso encontre, realiza o SQL do update (Curso.class.php).

Projeto	Data	Número	Tipo	Injetado	Removido	Tempo correção	Ref.
SIPS	10/02/2017	10	60	Codificação	Testes	5min	-

Descrição: Melhorar mensagem de exclusão do Curso. "Curso excluído com sucesso."

Projeto	Data	Número	Tipo	Injetado	Removido	Tempo correção	Ref.
SIPS	10/02/2017	11	40	Codificação	Revisão do código	5min	-

Descrição: Incluir a classe Usuário para criar novo objeto. Arquivo index_curso.php.

Projeto	Data	Número	Tipo	Injetado	Removido	Tempo correção	Ref.
SIPS	10/02/2017	12	80	Codificação	Testes	15min	-

Descrição: Professor substituto não deve ser inserido no vetor de $nivel_ensino. Isso é representado por Superior.

Projeto	Data	Número	Tipo	Injetado	Removido	Tempo correção	Ref.
SIPS	10/02/2017	13	40	Codificação	Revisão do código	10min	-

Descrição: A sessão deve ser iniciada em todas as páginas PHP de Curso.

APÊNDICE M – Resumo de plano de projeto – Iteração 2 – Programador A

Programador	Programador A	**Data**	10/02/2017
Sistema	SIPS	**Programa/Módulo**	Manter Curso
Instrutor	*Scrum Master*	**Linguagem**	PHP

Tamanho do Programa	**Atual**	**Até a Data**
Tamanho Total (T)	398 LOC	737 LOC

Tempo na Fase (min.)	**Real**	**Até a Data**
Planejamento	45	60
Projeto	110	195
Revisão de Projeto	90	150
Código	845	1425
Revisão do Código	175	280
Compilação	5	10
Teste	75	170
Postmortem	45	155
Total	1390	2445

Defeitos Injetados	**Real**	**Até a Data**
Planejamento	0	0
Projeto	6	12
Revisão de Projeto	0	0
Código	7	15
Revisão do Código	0	0
Compilação	0	0
Teste	0	0

Total	13	27

Defeitos Removidos	**Real**	**Até a Data**
Planejamento	0	0
Projeto	0	0
Revisão de Projeto	5	7
Código	0	0
Revisão do Código	5	11
Compilação	0	0
Teste	3	9
Total	13	27

Resumo de Qualidade	**PQI - Process Quality Process**			
	Tempo Projeto	**Tempo de Codificação**	**TOTAL**	
$PQI_{DesignCodeTime}$	110	845	**0,130177515**	Qualidade do projeto, sendo expressa pela divisão entre o tempo de projeto e o tempo de codificação. É recomendado que o tempo gasto na fase de projeto seja maior que o tempo gasto na fase de codificação.
	Tempo revisão projeto	**Tempo de Projeto**	**TOTAL**	
$PQI_{DesignReviewTime}$	90	110	**1,636363636**	Qualidade de revisão de projeto, sendo expressa pela divisão entre o dobro do tempo de revisão de projeto e o tempo de projeto. É recomendado que o tempo gasto na fase de revisão de projeto seja, no mínimo, metade do tempo gasto na fase de projeto.
	Tempo revisão Código	**Tempo de Codificação**	**TOTAL**	
$PQI_{CodeReviewTime}$	175	845	**0,414201183**	Qualidade de revisão de código, sendo expressa pela divisão entre o dobro do tempo de revisão de código e o tempo de codificação. É recomendado que o tempo gasto na fase de revisão de código seja, no mínimo, a metade do tempo gasto na fase de codificação.
	Defeitos per KLOC testes unitários		**TOTAL**	
$PQI_{UnitTestDefects}$	7		0,833333333	Qualidade do programa, sendo expressa pela divisão de 10 pela soma do número de defeitos (encontrados nos testes unitários a cada 1000 linhas de código) com 5, que representa o valor alvo para essa métrica, ou seja, a densidade de defeitos da fase de testes unitários não deve ser superior a 5 defeitos/KLOC.

APÊNDICE N – Log de registro de tempo – Iteração 2 – Programador B

Programador	Programador B	**Data**	06/02/2017 - 10/02/2017
Sistema	SIPS	**Programa/Módulo**	Manter Componente
Instrutor	*Scrum Master*	**Linguagem**	PHP

Projeto	Fase	Data e Tempo de Inicio	Tempo de Interrupção	Data e Tempo de Fim	Tempo Delta	Comentários
SIPS	Planejamento	06/02/2017 - 08:15	30	06/02/2017 - 09:05	20	Reunião Scrum. 15min. Responder alguns e-mails. 15min.
SIPS	Projeto	06/02/2017 - 09:05	0	06/02/2017 - 10:35	90	Criação do projeto operacional e funcional.
SIPS	Revisão do Projeto	06/02/2017 - 10:35	10	06/02/2017 - 12:10	75	Outras tarefas. 10min. Lista de verificação de revisão de projeto.
SIPS	Codificação	06/02/2017 - 12:10	0	06/02/2017 - 13:50	100	Codificação da classe Componente.class.php
SIPS	Codificação	07/02/2017 - 08:00	30	07/02/2017 - 13:55	325	Reunião Scrum. 15min. Responder e-mails. 15min. Suporte a sistemas. 30min. Continuação da codificação Classe Componente.class.php (125 LOC)
SIPS	Codificação	08/02/2017 - 08:10	80	08/02/2017 - 14:00	270	Reunião Scrum. 30min. Responder e-mails. 20min. Suporte a outros sistemas. 30min. Codificação: index_componente.php; cadastrar_componente.php; exibir_componente.php; editar_componente.php; excluir_componente.php
SIPS	Codificação	09/02/2017 - 08:00	30	09/02/2017 - 13:40	310	Reunião diária de 15min. Responder e-mails. 15min. Continuação codificação: index_componente.php; cadastrar_componente.php; exibir_componente.php; editar_componente.php; excluir_componente.php (242 LOC)
SIPS	Revisão da Codificação	10/02/2017 - 08:10	25	10/02/2017 - 10:00	85	Reunião Scrum. 15min. Outras atividades. 10min. Checklist de revisão de código.
SIPS	Compilação	10/02/2017 - 10:00	0	10/02/2017 - 10:05	5	
SIPS	Teste	10/02/2017 - 10:05	0	10/02/2017 - 12:30	145	Verificação de e-mail. 10min. Utilização do template de teste.
SIPS	Postmortem	10/02/2017 - 12:30	15	10/02/2017 - 14:00	75	Outras atividades. 10min. Preencimento do formulário de resumo de projeto.

APÊNDICE O – Log de registro de defeitos – Iteração 2 – Programador B

Programador	Programador B	**Data**	06/02/2017 - 10/02/2017
Sistema	SIPS	**Programa/Módulo**	Manter Componente
Instrutor	*Scrum Master*	**Linguagem**	PHP

Projeto	Data	Número	Tipo	Injetado	Removido	Tempo correção	Ref.
SIPS	06/02/2017	1	10	Projeto	Revisão do Projeto	5min	-

Descrição: Fluxo alternativo da etapa 03: Todos os campos são obrigatórios. Template operacional.

Projeto	Data	Número	Tipo	Injetado	Removido	Tempo correção	Ref.
SIPS	06/02/2017	2	10	Projeto	Revisão do Projeto	5min	-

Descrição: Adicionar fluxo alternativo ao passo 05 caso o administrador não preencha todos os dados corretamente. Template operacional.

Projeto	Data	Número	Tipo	Injetado	Removido	Tempo correção	Ref.
SIPS	06/02/2017	3	10	Projeto	Revisão do Projeto	5min	-

Descrição: Adicionar fluxo alternativo ao passo 12 do Template operacional.

Projeto	Data	Número	Tipo	Injetado	Removido	Tempo correção	Ref.
SIPS	10/02/2017	4	10	Projeto	Testes	5min	-

Descrição: Adicionar fluxo alternativo ao passo 16 do Template operacional.

Projeto	Data	Número	Tipo	Injetado	Removido	Tempo correção	Ref.
SIPS	06/02/2017	5	10	Projeto	Revisão do Projeto	5min	-

Descrição: Método excluir edital recebe como parâmetro o ID e retorna booleano (se excluiu ou não).

Projeto	Data	Número	Tipo	Injetado	Removido	Tempo correção	Ref.
SIPS	10/02/2017	6	40	Codificação	Revisão do Código	5min	-

Descrição: Incluir a classe Componente nos arquivos PHP.

Projeto	Data	Número	Tipo	Injetado	Removido	Tempo correção	Ref.
SIPS	10/02/2017	7	40	Codificação	Testes	5min	-

Descrição: Incluir a classe Curso.class.php no arquivo cadastrar_componente.php.

Projeto	Data	Número	Tipo	Injetado	Removido	Tempo correção	Ref.

Projeto	Data	Número	Tipo	Injetado	Removido	Tempo correção	Ref.
SIPS	10/02/2017	8	10	Projeto	Revisão do Projeto	5min	-

Descrição: Associar componentes aos processo seletivos.

Projeto	Data	Número	Tipo	Injetado	Removido	Tempo correção	Ref.
SIPS	10/02/2017	9		Codificação	Revisão do Código	15min	-

Descrição: Utilizar função de consulta por ID para buscar o componente a excluído. Se encontrar, executa o SQL para excluir.

Projeto	Data	Número	Tipo	Injetado	Removido	Tempo correção	Ref.
SIPS	10/02/2017	10		Codificação	Testes	5min	-

Descrição: Mensagem inadequada pra alteração de componentes. "Componente atualizado com sucesso."

Projeto	Data	Número	Tipo	Injetado	Removido	Tempo correção	Ref.
SIPS	10/02/2017	11		Codificação	Testes	10min	-

Descrição: Incluir a classe ProcessoSeletivo no index_componente.php.

APÊNDICE P – Resumo de plano de projeto – Iteração 2 – Programador B

Programador	Programador B	**Data**	06/02/2017 - 10/02/2017
Sistema	SIPS	**Programa/Módulo**	Manter Componente
Instrutor	*Scrum Master*	**Linguagem**	PHP

Tamanho do Programa	**Atual**	**Até a Data**
Tamanho Total (T)	367 LOC	875 LOC

Tempo na Fase (min.)	**Real**	**Até a Data**
Planejamento	20	75
Projeto	90	190
Revisão de Projeto	75	150
Codificação	1005	1935
Revisão do Código	85	155
Compilação	5	10
Teste	145	260
Postmortem	75	185
Total	1500	2960

Defeitos Injetados	**Real**	**Até a Data**
Planejamento	0	0
Projeto	6	9
Revisão de Projeto	0	0
Codificação	5	14
Revisão do Código	0	0
Compilação	0	0
Teste	0	0

Total	11	23

Defeitos Removidos	**Real**	**Até a Data**
Planejamento	0	0
Projeto	0	0
Revisão de Projeto	5	6
Codificação	0	0
Revisão do Código	2	7
Compilação	0	0
Teste	4	10
Total	11	23

Resumo de Qualidade	**PQI - Process Quality Process**			
	Tempo Projeto	**Tempo de Codificação**	**TOTAL**	Qualidade do projeto, sendo expressa pela divisão entre o tempo de projeto e o tempo de codificação. É recomendado que o tempo gasto na fase de projeto seja maior que o tempo gasto na fase de codificação.
$PQI_{DesignCodeTime}$	90	1005	**0,089552239**	
	Tempo Revisão Proje	**Tempo de Projeto**	**TOTAL**	Qualidade de revisão de projeto, sendo expressa pela divisão entre o dobro do tempo de revisão de projeto e o tempo de projeto. É recomendado que o tempo gasto na fase de revisão de projeto seja, no mínimo, metade do tempo gasto na fase de projeto.
$PQI_{DesignReviewTime}$	75	90	**1,666666667**	
	Tempo Revisão Códig	**Tempo de Codificação**	**TOTAL**	Qualidade de revisão de código, sendo expressa pela divisão entre o dobro do tempo de revisão de código e o tempo de codificação. É recomendado que o tempo gasto na fase de revisão de código seja, no mínimo, a metade do tempo gasto na fase de codificação.
$PQI_{CodeReviewTime}$	85	1005	**0,169154229**	
	Defeitos per KLOC testes unitários		**TOTAL**	Qualidade do programa, sendo expressa pela divisão de 10 pela soma do número de defeitos (encontrados nos testes unitários a cada 1000 linhas de código) com 5, que representa o valor alvo para essa métrica, ou seja, a densidade de defeitos da fase de testes unitários não deve ser superior a 5 defeitos/KLOC.
$PQI_{UnitTestDefects}$	10		0,666666667	

APÊNDICE Q – Log de registro de tempo– Iteração 2 – Programador C

Programador	Programador C	**Data**	06/02/2017 - 10/02/2017
Sistema	SIPS	**Programa/Módulo**	Manter Log
Instrutor	*Scrum Master*	**Linguagem**	PHP

Projeto	Fase	Data e Tempo de Inicio	Tempo de Interrupção	Data e Tempo de Fim	Tempo Delta	Comentários
SIPS	Planejamento	06/02/2017 - 08:05	25	06/02/2017 - 08:55	25	Reunião (15min). Checar e-mails (10min).
SIPS	Projeto	06/02/2017 - 08:55	0	06/02/2017 - 10:30	85	Especificação operacional e funcional.
SIPS	Revisão do Projeto	06/02/2017 - 10:30	0	06/02/2017 - 11:30	60	Lista de verificação de revisão de projeto.
SIPS	Codificação	06/02/2017 - 11:30	30	06/02/2017 - 14:00	120	Início da criação da Classe Logo.class.php.
SIPS	Codificação	07/02/2017 - 08:10	40	07/02/2017 - 13:55	305	Reunião (15min). Checar-mails (10min). Suporte a usuarios (15min). Conclusão da codificação Classe Log.class.php (123 LOC) Início da criação dos arquivos index_log.php; exibir_log.php e consultar_log.php.
SIPS	Codificação	08/02/2017 - 08:00	80	08/02/2017 - 13:55	275	Reunião (15min). Checar e-mails (30min). Suporte a usuários (35min). Continuação codificação dos arquivos index_log.php; exibir_log.php e consultar_log.php.
SIPS	Codificação	09/02/2017 - 08:20	30	09/02/2017 - 12:30	220	Reunião (15min). Checar e-mails (10min). Telefonema (5min). Conclusão dos arquivos index_log.php; exibir_log.php e consultar_log.php (189 LOC)
SIPS	Revisão da Codificação	09/02/2017 - 12:30	0	09/02/2017 - 14:00	90	Checklist de revisão de código.
SIPS	Compilação	09/02/2017 - 14:00	0	09/02/2017 - 14:05	5	
SIPS	Teste	10/02/2017 - 08:20	25	10/02/2017 - 11:40	175	Reunião (15min). Checar e-mails (10min). Utilização do template de teste.
SIPS	Postmortem	10/02/2017 - 11:40	40	10/02/2017 - 13:50	90	Suporte a usuários (40min). Preencimento da planilha de resumo de projeto.

APÊNDICE R – Log de registro de defeitos – Iteração 2 – Programador C

Programador	Programador C	**Data**	06/02/2017 - 10/02/2017
Sistema	SIPS	**Programa/Módulo**	Manter Log
Instrutor	*Scrum Master*	**Linguagem**	PHP

Projeto	Data	Número	Tipo	Injetado	Removido	Tempo correção	Ref.
SIPS	06/02/2017	1	10	Projeto	Revisão do Projeto	5min	-

Descrição: Na etapa 5 do template de especificação operacional, os parâmetros de pesquisa devem ser nome de usuário, IP, código, data e hora.

Projeto	Data	Número	Tipo	Injetado	Removido	Tempo correção	Ref.
SIPS	06/02/2017	2	10	Projeto	Revisão do Projeto	10min	-

Descrição: O registro do log deve conter o nome do usuário.

Projeto	Data	Número	Tipo	Injetado	Removido	Tempo correção	Ref.
SIPS	06/02/2017	3	10	Projeto	Revisão do Projeto	5min	-

Descrição: Usuário deve ser informada como classe pai de log.

Projeto	Data	Número	Tipo	Injetado	Removido	Tempo correção	Ref.
SIPS	06/02/2017	4	10	Projeto	Revisão do Projeto	15min	-

Descrição: Relacionamento entre as classes deve ser 1...N (um usuário possui muitos logs).

Projeto	Data	Número	Tipo	Injetado	Removido	Tempo correção	Ref.
SIPS	10/02/2017	5	80	Projeto	Revisão do Código	10min	-

Descrição: A consulta de log por usuário deve receber o ID do usuário e retornar um array com o nome (se o encontrar).

Projeto	Data	Número	Tipo	Injetado	Removido	Tempo correção	Ref.
SIPS	10/02/2017	6	80	Codificação	Revisão do Código	5min	-

Descrição: Método para gravar log não recebe o ID como parâmetro.

Projeto	Data	Número	Tipo	Injetado	Removido	Tempo correção	Ref.

	Projeto	Data	Número	Tipo	Injetado	Removido	Tempo correção	Ref.
	SIPS	10/02/2017	7	80	Codificação	Testes	15min	-
Descrição:	A variável $ip deve receber função getenv na linha 41. Log.class.php.							
	Projeto	Data	Número	Tipo	Injetado	Removido	Tempo correção	Ref.
	SIPS	10/02/2017	8	50	Codificação	Testes	10min	-
Descrição:	Apresentar, na página inicial dos logs dos usuários, os 50 primeiros registros e opções para avançar ou realizar busca avançada.							
	Projeto	Data	Número	Tipo	Injetado	Removido	Tempo correção	Ref.
	SIPS	10/02/2017	9	60	Codificação	Testes	5min	-
Descrição:	Na consulta por data e hora, o parâmetro que deve ser retornado é o nome do usuário. Recebe data e hora a ser pesquisado.							

APÊNDICE S – Resumo de plano de projeto – Iteração 2 – Programador C

Programador	Programador C	**Data**	06/02/2017 - 10/02/2017
Sistema	SIPS	**Programa/Módulo**	Manter Log
Instrutor	*Scrum Master*	**Linguagem**	PHP

Tamanho do Programa	**Atual**	**Até a Data**
Tamanho Total (T)	312 LOC	654 LOC

Tempo na Fase (min.)	**Real**	**Até a Data**
Planejamento	25	45
Projeto	85	165
Revisão de Projeto	60	110
Codificação	920	1930
Revisão do Código	90	135
Compilação	5	10
Teste	175	300
Postmortem	90	215
Total	1450	2910

Defeitos Injetados	**Real**	**Até a Data**
Planejamento	0	0
Projeto	5	9
Revisão de Projeto	0	0
Codificação	4	10
Revisão do Código	0	0
Compilação	0	0
Teste	0	0
Total	9	19

Defeitos Removidos	**Real**	**Até a Data**
Planejamento	0	0
Projeto	0	0
Revisão de Projeto	4	6
Codificação	0	0
Revisão do Código	2	6
Compilação	0	0
Teste	3	7
Total	9	19

Resumo de Qualidade	**PQI - Process Quality Process**			
	Tempo Projeto	**Tempo de Codificação**	**TOTAL**	
$PQI_{DesignCodeTime}$	85	920	**0,092391304**	Qualidade do projeto, sendo expressa pela divisão entre o tempo de projeto e o tempo de codificação. É recomendado que o tempo gasto na fase de projeto seja maior que o tempo gasto na fase de codificação.
	Tempo Revisão Projeto	**Tempo de Projeto**	**TOTAL**	
$PQI_{DesignReviewTime}$	60	85	**1,411764706**	Qualidade de revisão de projeto, sendo expressa pela divisão entre o dobro do tempo de revisão de projeto e o tempo de projeto. É recomendado que o tempo gasto na fase de revisão de projeto seja, no mínimo, metade do tempo gasto na fase de projeto.
	Tempo Revisão Código	**Tempo de Codificação**	**TOTAL**	
$PQI_{CodeReviewTime}$	90	920	**0,195652174**	Qualidade de revisão de código, sendo expressa pela divisão entre o dobro do tempo de revisão de código e o tempo de codificação. É recomendado que o tempo gasto na fase de revisão de código seja, no mínimo, a metade do tempo gasto na fase de codificação.
	Defeitos per KLOC testes unitários		**TOTAL**	
$PQI_{UnitTestDefects}$	9		0,714285714	Qualidade do programa, sendo expressa pela divisão de 10 pela soma do número de defeitos (encontrados nos testes unitários a cada 1000 linhas de código) com 5, que representa o valor alvo para essa métrica, ou seja, a densidade de defeitos da fase de testes unitários não deve ser superior a 5 defeitos/KLOC.

APÊNDICE T – Template de esp. operacional - Iteração 3 – Programador A

Programador	Programador A	Data	13/02/2017
Sistema	SIPS	Programa/Módulo	Visão Administrativa (pagamento, homologação)
Instrutor	*Scrum Master*	Linguagem	PHP

Número do cenário	Admin01	**Objetivo do usuário**	Administrar o SIPS.
Objetivo do cenário		Criar/montar as telas de login, página inicial, página de administração e página de relatórios, além das funcionalidades de consultar inscrição, confirmar pagamento e gerar homologação.	

Fonte (Source)	**Etapa**	**Ação**	**Comentários**
Usuário	01	Acessa a tela de login do SIPS.	- Adicionar Contato: proseletivo@iffarroupilha.edu.br
Sistema	02	Exibe a tela de login.	
Usuário	03	Digita o número do SIAPE, a senha e clica no botão entrar.	O sistema não limita o número de tentativas.
Sistema	04	Exibe a página inicial de processos (com os processos ativos) e o conjunto de menus (Página Inicial, Administrar, Relatórios).	Incluir menu Sair (do sistema e voltar para página inicial de processos seletivos).
Usuário	05	Acessa menu Administrar.	.
Sistema	06	Exibe a página Administrar possibilitando manter: instituições, processos seletivos, cursos, editais, pagamentos, isenções, ensalamento, correção, classificação, usuários e relatórios.	- As 10 opções devem ser representadas por imagens e botões, com link para as respectivas páginas. -Correção e Classificação ainda não têm seus requisitos explicitados.
Usuário	07	Acessa Instituições.	
Sistema	08	Exibe a página para administrar instituições com as opções de cadastrar, consultar, editar e excluir.	- O sistema deve exibir mensagens de êxito ou fracasso nas operações das opções.
Usuário	09	Acessa Processos Seletivos.	
Sistema	10	Exibe a página para administrar processos seletivos, com as opções de cadastrar, consultar, editar e excluir.	- O sistema deve exibir mensagens de êxito ou fracasso nas operações das opções.
Usuário	11	Acessa Cursos.	
Sistema	12	Exibe a página para administrar cursos, com as opções de cadastrar, consultar, editar e excluir.	- O sistema deve exibir mensagens de êxito ou fracasso nas operações das opções.
Usuário	13	Acessa Editais.	
Sistema	14	Exibe a página para administrar editais, com as opções de cadastrar, consultar, editar e excluir.	- O sistema deve exibir mensagens de êxito ou fracasso nas operações das opções.
Usuário	15	Acessa Pagamentos.	
Sistema	16	Exibe a página para administrar pagamentos, com as funcionalidades: Consultar Pagamentos, Confirmar Pagamento e Gerar Homologação.	- As 4 funcionalidades podem até ser previstas aqui, porém Processar Pagamentos não será implementada nessa iteração.
Usuário	17	Acessa Pagamentos >Consultar Pagamentos.	
Sistema	18	Exibe a página para consultar pagamentos de candidato por nome, CPF ou número de inscrição (também podendo consultar todos), parametrizando por instituição, processo seletivo e curso.	Deve apresentar um botão para voltar à página da etapa 16.
Usuário	19	Informa parâmetros da pesquisa e clica em consultar.	
Sistema	20	Exibe a página do resultado da pesquisa em uma tabela com colunas: número da inscrição, nome, CPF, Cargo (curso), Situação (homologado ou não homologado) e com duas opções (botões): Ver dados e Confirmar Pagamento.	Deve apresentar um botão para voltar à página da etapa 18.

Usuário	21	Clica em Ver Dados.	
Sistema	22	Exibe a página com uma seção de resumo dos dados da inscrição e uma seção com a situação da inscrição: pagamento da inscrição (pago, não pago ou pago parcial), data do pagamento (caso tenha sido pago) e homologação da inscrição (não homologado ou homologado).	- Essas informações devem ser enviadas para o e-mail do candidato (caso tenha informado). - Deve apresentar um botão para voltar à página da etapa 20.
Usuário	23	Acessa Pagamentos >Confirmar Pagamento.	
Sistema	24	Exibe a página da etapa 18.	
Usuário	25	Passa pelas etapas 19, 20 e clica em Confirmar Pagamento.	
Sistema	26	Exibe a página com uma seção de resumo dos dados da inscrição e uma seção dos dados da GRU: data do pagamento, valor pago, valor da GRU e situação (pago ou pago parcial).	A opção não pago não é necessária na seção da GRU.
Usuário	27	Informa data do pagamento, valor pago, valor da GRU, seleciona a situação e clica no botão confirmar pagamento.	Deve apresentar um botão para voltar à página da etapa 20.
Sistema	28	Exibe a página para gerar homologação.	
Usuário	29	Acessa Pagamentos >Gerar Homologação.	
Sistema	30	Exibe a página para gerar homologação, com parâmetros de pesquisa: instituição, processo seletivo e curso.	Deve apresentar um botão para voltar à página da etapa 16.
Usuário	31	Informa os parâmetros e clica em Gerar Homologação.	
Sistema	32	Exibe a página do resultado da homologação, apresentando o total de inscrições homologadas (pagas), juntamente com o número da inscrição.	- Exibe uma mensagem de homologação realizada com sucesso. - Deve apresentar um botão para voltar à página da etapa 31. - A geração da homologação pode ser realizada mais de uma vez.
Usuário	33	Acessa Isenções.	
Sistema	34	Exibe a página para administrar isenções, com as funcionalidades de Exportar Solicitações, Processar Isenções e Consultar Isentos.	- As funcionalidades devem ser previstas aqui, porém serão desenvolvidas em iterações futuras.
Usuário	35	Acessa Ensalamento.	
Sistema	36	Exibe a página para administrar ensalamentos, com opções de manter (cadastrar, consultar, editar e excluir) Locais de Provas, Prédios e Salas, bem como a funcionalidade Ensalar.	- As funcionalidades devem ser previstas aqui, porém serão desenvolvidas em iterações futuras.
Usuário	37	Acessa Usuários.	
Sistema	38	Exibe a página para administrar usuários, as opções de cadastrar, consultar, editar e excluir.	- O sistema deve exibir mensagens de êxito ou fracasso nas operações das opções.
Usuário	39	Acessa Relatórios (ou menu relatórios).	
Sistema	40	Exibe a página de relatórios, com as opções de relatórios Inscritos, Homologados, Isentos, Cotistas, Portadores de Necessidades Especiais (ou adaptações especiais para prova), Pagamentos e Marketing	- As 7 opções de relatórios devem ser representadas por imagens e botões, com link para as respectivas páginas (que serão criadas pelo programador C, na iteração 3).

APÊNDICE U – Template de especificação funcional – Iteração 3 – Prog. A

Programador	Programador A	Data	13/02/2017
Sistema	SIPS	Programa/Módulo	Visão Administrativa (pagamento)
Instrutor	*Scrum Master*	Linguagem	PHP

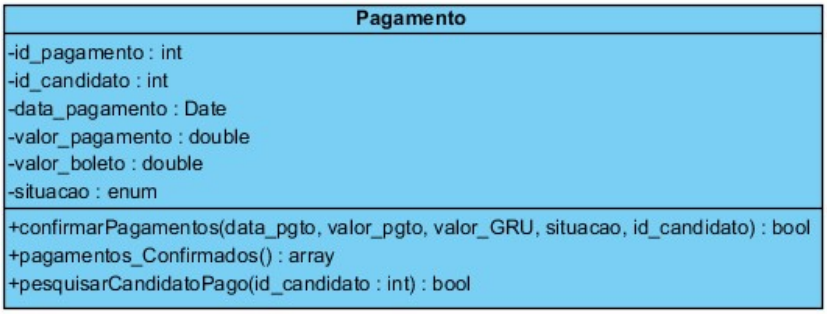
Nome da Classe	Pagamento
Classe Pai	Generic Db

Atributos

Declaração	**Descrição**
ID pagamento	Identificador de 10 dígitos do pagamento.
ID candidato	Identificador de 10 dígitos do candidato.
Data pagamento	Consiste na data do pagamento.
Valor pagamento	É o valor pago pelo candidato.
Valor boleto	Consiste no valor do boleto ou da GRU. Atualmente, a GRU é de 40,00 para cada inscrição no processo seletivo para professores substitutos.
Situação	Pago, não pago ou pago parcial.

Itens

Declaração	**Descrição**
Confirmar pagamentos	Deve receber como parâmetro a data, o valor pago, o valor da GRU e o identificador do candidato e verificar se o pagamento já foi confirmado. Se não foi constrói e executa a *query* SQL para confirmar o pagamento.
Pagamentos confirmados	Deve retornar todos os pagamentos confirmados de determinado curso.
Pesquisa candidato pago	Deve receber como parâmetro o identificador do candidato, retornando os candidatos que realizaram o pagamento integral da inscrição.



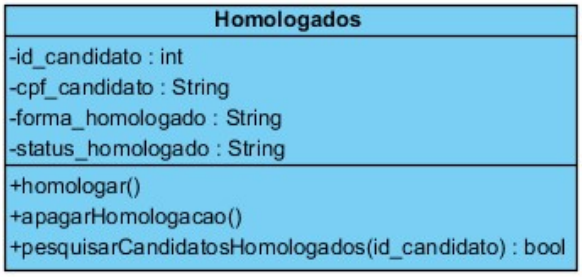
APÊNDICE V – Template de especificação funcional – Iteração 3 – Prog. A

Programador	Programador A	Data	13/02/2017
Sistema	SIPS	Programa/Módulo	Visão Administrativa (homologação)
Instrutor	*Scrum Master*	Linguagem	PHP

Nome da Classe	Homologação
Classe Pai	Generic Db

Atributos	
Declaração	**Descrição**
ID candidato	Identificador de 10 dígitos do candidato.
CPF do candidato	CPF do candidato homologado.
Forma homologado	Representa a forma de homologação.
Status homologado	Representa o status da homologação (homologado ou não homologado).

Itens	
Declaração	**Descrição**
Homologar	Homologa todos os candidatos que pagaram integralmente a inscrição.
Apagar homologação	Deve apagar a última homologação realizada, antes de realizar nova homologação.
Pesquisar candidatos homologados	Recebe o identificador do candidato como parâmetro e retorna todos os candidatos homologados.



APÊNDICE X – Template de relatório de teste – Iteração 3 – Programador A

Programador	Programador A	Data	15, 17, 21 e 23/02/2017
Sistema	SIPS	Programa/Módulo	Visão Administrativa
Instrutor	*Scrum Master*	Linguagem	PHP

Nome/Número do Teste	Admin 01.
Objetivo do Teste	Testar a navegabilidade das telas e funcionalidades.
Descrição do Teste	Acessar a área administrativa do SIPS, realizar login, acessar página inicial, página administrativa e de relatórios. Acessar todas as opções da área administrativa, especialmente as funções de pagamento, instanciando as classes. Criar os cadastros básicos.
Condições do Teste	- Os testes devem ser realizados no servidor de testes, conforme PDS. - Os testes devem ser realizados com usuário Administrador. - Verificar se todos os componentes e opções estão presentes em tela. - Verificar se os fluxos estão corretos. - Para a completa realização dos testes, contar com os artefatos produzidos pelos outros programadores.
Resultados esperados	- Telas completas e com facilidade de navegação, com conteúdos e opções devidamente organizados e presentes no devido lugar. - Funcionalidades da opção pagamentos corretas. - Cadastros básicos criados com sucesso (instituição, processo seletivo e cursos).
Resultados reais	Os testes com área administrativa foram realizados com sucesso, exceto: - Para candidatos com mais de uma inscrição paga, apenas a última inscrição deve ser homologada. - Método que Pesquisa candidato pago, os candidatos que não pagaram integralmente a inscrição não devem ser listados. - No campo de status da homologação do candidato deve constar apenas as condições de homologado ou não homologado. - Requer as classes de inscrição e candidato. Adicionar em Pagamento.class.php. - A função confirmarPagamento deve receber também o ID do candidato (Pagamento.class.php). - O arquivo Homologacao.php requer a classe de processo seletivo. - A função homologar deve iniciar com ApagarHomologacao(), no arquivo Homologacao.class.php - Ordenar os dados de todos os candidatos por data de pagamento na classe Homologacao.class.php. - Requer as classes de processo seletivo e homologação, no arquivo gerar_homologacao.class.php. - $retorno = $objHomologacao->getMessage() na linha 59 do arquivo gerar_homologacao.class.php. - Solicitar a conexão do gerenciador de acordo com as variáveis contidas em db.inc.php.

APÊNDICE Z – Log de registro de tempo – Iteração 3 – Programador A

Programador	Programador A	**Data**	13/02/2017 - 24/02/2017
Sistema	SIPS	**Programa/Módulo**	Visão Administrativa
Instrutor	*Scrum Master*	**Linguagem**	PHP

Projeto	Fase	Data e Tempo de Inicio	Tempo de Interrupção	Data e Tempo de Fim	Tempo Delta	Comentários
SIPS	Planejamento	13/02/2017 - 08:00	45	13/02/2017 - 09:45	60	Reunião diária de 30min. Outras atividades de 15min. Elicitação do RQF-19 (visão do admin), RQF-23 e RQF-25.
SIPS	Projeto	13/02/2017 - 09:45	5	13/02/2017 - 12:25	155	Outras atividades de 5min. Preenchimento dos templates de especificação operacional e funcional.
SIPS	Revisão do Projeto	13/02/2017 - 12:25	0	13/02/2017 - 14:00	95	Utilização do checklist de revisão de projeto.
SIPS	Codificação	14/02/2017 - 08:10	25	14/02/2017 - 13:10	205	Reunião diária de 15min. Outras atividades de 10min. Criação da classe Pagamento.class.php (110LOC)
SIPS	Revisão da Codificação	14/02/2017 - 13:10	0	14/02/2017 - 13:55	45	Utilização da checklist de revisão de código.
SIPS	Compilação	14/02/2017 - 13:55	0	14/02/2017 - 14:00	5	
SIPS	Teste	15/02/2017 - 08:15	30	15/02/2017 - 09:35	50	Reunião diária de 15min. Outras atividades 5min. Utilização do template de testes.
SIPS	Codificação	15/02/2017 - 09:35	60	15/02/2017 - 14:00	205	Criação da classe Homologacao.class.php (89LOC)
SIPS	Revisão da Codificação	16/02/2017 - 08:10	25	16/02/2017 - 10:45	130	Reunião diária de 15min. Outras atividades de 10min. Utilização da checklist de revisão de código.
SIPS	Compilação	16/02/2017 - 10:45	0	16/02/2017 - 10:50	5	
SIPS	Codificação	16/02/2017 - 10:45	0	16/02/2017 - 13:55	170	Páginas: index.php; logado_admin.php; pag_inicial.php.
SIPS	Codificação	17/02/2017 - 08:00	25	17/02/2017 - 10:40	175	Reunião diária de 15min. Outras atividades de 10mn. index.php; logado_admin.php; pag_inicial.php. (165LOC)
SIPS	Revisão da Codificação	17/02/2017 - 10:40	0	17/02/2017 - 12:00	80	Utilização da checklist de revisão de código.

SIPS	Compilação	17/02/2017 - 12:00	5	17/02/2017 - 12:05	5	
SIPS	Teste	17/02/2017 - 12:05	10	17/02/2017 - 13:30	75	Outras atividades 10min. Utilização do template de testes.
SIPS	Codificação	20/02/2017 - 08:00	30	20/02/2017 - 13:40	310	Reunião diária de 15min. Outras atividades de 15min. administrar.php, relatorios.php (127LOC)
SIPS	Revisão da Codificação	21/02/2017 - 08:00	0	21/02/2017 - 09:15	75	Reunião diária de 15min. Utilização da checklist de revisão de código.
SIPS	Compilação	21/02/2017 - 09:15	0	21/02/2017 - 09:20	5	
SIPS	Teste	21/02/2017 - 09:20		21/02/2017 - 09:45	25	Utilização do template de testes.
SIPS	Codificação	21/02/2017 - 09:45	10	21/02/2017 - 14:00	245	Outras atividades de 15min. Criação das funcionalidades Consultar Pagamentos, Confirmar Pagamento e Gerar Homologação.
SIPS	Codificação	22/02/2017 - 08:05	45	22/02/2017 - 13:50	300	Reunião diária de 15min. Outras atividades de 30min. gerrar_homologacao.php, consultar_pagamento.php, confirmar_pagamento.php
SIPS	Codificação	23/02/2017 - 08:05	30	22/02/2017 - 13:50	315	Reunião diária de 15min. Outras atividades de 15min. gerrar_homologacao.php, consultar_pagamento.php, confirmar_pagamento.php (263LOC)
SIPS	Revisão da Codificação	23/02/2017 - 08:00	15	23/02/2017 - 09:05	50	Reunião diária de 15min. Utilização da checklist de revisão de código.
SIPS	Compilação	23/02/2017 - 09:05	0	23/02/2017 - 09:10	5	
SIPS	Teste	23/02/2017 - 09:10	5	23/02/2017 - 10:40	85	Outras atividades de 05min. Utilização do template de testes.
SIPS	Postmortem	23/02/2017 - 10:40	10	23/02/2017 - 13:30	180	Preencimento do resumo de projeto com os totais da iteração 3.

APÊNDICE AA – Log de registro de defeitos – Iteração 3 – Programador A

Programador	Programador A	**Data**	13/02/2017 - 24/02/2017
Sistema	SIPS	**Programa/Módulo**	Visão Administrativa
Instrutor	*Scrum Master*	**Linguagem**	PHP

Projeto	Data	Número	Tipo	Injetado	Removido	Tempo correção	Ref.
SIPS	13/02/2017	1	10	Projeto	Revisão do Projeto	5min	-

Descrição: Incluir menu Sair na página inicial.

Projeto	Data	Número	Tipo	Injetado	Removido	Tempo correção	Ref.
SIPS	13/02/2017	2	10	Projeto	Revisão do Projeto	5min	-

Descrição: Correção e Classificação ainda não têm seus requisitos explicitados, mas devem constar nas opções do administrador.

Projeto	Data	Número	Tipo	Injetado	Removido	Tempo correção	Ref.
SIPS	13/02/2017	3	10	Projeto	Revisão do Projeto	5min	-

Descrição: A funcionalidade Processar Pagamentos deve ser prevista, mas não será implementada nessa iteração.

Projeto	Data	Número	Tipo	Injetado	Removido	Tempo correção	Ref.
SIPS	13/02/2017	4	10	Projeto	Testes	10min	-

Descrição: Na página de resultado da pesquisa da consulta de pagamentos, na tabela, constar o Cargo do candidato.

Projeto	Data	Número	Tipo	Injetado	Removido	Tempo correção	Ref.
SIPS	13/02/2017	5	10	Projeto	Revisão do Projeto	5min	-

Descrição: Na etapa 26 da especificação operacional da visão administrativa, a situação deve ser selecionada entre: pago ou pago parcial.

Projeto	Data	Número	Tipo	Injetado	Removido	Tempo correção	Ref.
SIPS	14/02/2017	6	70	Codificação	Testes	10min	-

Descrição: Um candidato com mais de uma inscrição paga deve ser homologada apenas a última inscrição.

Projeto	Data	Número	Tipo	Injetado	Removido	Tempo correção	Ref.
SIPS	13/02/2017	7	10	Projeto	Revisão do Projeto	5min	-

Descrição: Acrescentar Prédios à etapa 36 da especificação operacional (visão administrativa).

Projeto	Data	Número	Tipo	Injetado	Removido	Tempo correção	Ref.
SIPS	13/02/2017	8	40	Projeto	Revisão do Projeto	10min	-

Descrição: ID candidato deve constar na especificação funcional da classe pagamento.

Projeto	Data	Número	Tipo	Injetado	Removido	Tempo correção	Ref.
SIPS	14/02/2017	9	80	Codificação	Revisão do código	15min	-

Descrição: No método que Pesquisa candidato pago, candidatos que não pagaram integralmente não devem ser retornados (devem constar como não pago).

Projeto	Data	Número	Tipo	Injetado	Removido	Tempo correção	Ref.
SIPS	15/02/2017	10	70	Codificação	Revisão do código	5min	-

Descrição: Status homologado deve constar apenas as condições de homologado ou não homologado.

Projeto	Data	Número	Tipo	Injetado	Removido	Tempo correção	Ref.
SIPS	15/02/2017	11	20	Codificação	Revisão do código	5min	-

Descrição: Requer as classes de inscrição e candidato (Pagamento.php).

Projeto	Data	Número	Tipo	Injetado	Removido	Tempo correção	Ref.
SIPS	15/02/2017	12	80	Codificação	Testes	10min	-

Descrição: A função confirmarPagamento deve receber também o ID do candidato (Pagamento.php).

Projeto	Data	Número	Tipo	Injetado	Removido	Tempo correção	Ref.
SIPS	13/02/2017	13	10	Projeto	Revisão do projeto	5min	-

Descrição: Mensagem incoerente no retorno da função de busca por candidato pago: "Não há registros de pagamento de um candidato com o código informado!

Projeto	Data	Número	Tipo	Injetado	Removido	Tempo correção	Ref.
SIPS	15/02/2017	14	40	Codificação	Revisão do código	5min	-

Descrição: Requer a classe de processo seletivo (Homologacao.php).

Projeto	Data	Número	Tipo	Injetado	Removido	Tempo correção	Ref.

SIPS	16/02/2017	15	80	Codificação	Revisão do código	10min	-

Descrição: A função homologar deve iniciar com ApagarHomologacao(), para sobrescrever a última homologação realizada (se existir). (Homologacao.php)

Projeto	Data	Número	Tipo	Injetado	Removido	Tempo correção	Ref.
SIPS	17/02/2017	16	40	Codificação	Testes	20min	-

Descrição: Ordenar os dados de todos os candidatos por data de pagamento. (Homologacao.php)

Projeto	Data	Número	Tipo	Injetado	Removido	Tempo correção	Ref.
SIPS	21/02/2017	17	40	Codificação	Revisão do código	5min	-

Descrição: Requer as classes de processo seletivo e homologacao (gerar_homologacao.php).

Projeto	Data	Número	Tipo	Injetado	Removido	Tempo correção	Ref.
SIPS	22/02/2017	18	80	Codificação	Revisão do código	5min	-

Descrição: $retorno = $objHomologacao->getMessage() na linha 59. (gerar_homologacao.php).

Projeto	Data	Número	Tipo	Injetado	Removido	Tempo correção	Ref.
SIPS	23/02/2017	19	30	Codificação	Revisão do código	15min	-

Descrição: Solicitar a conexao do gerenciador de acordo com as variáveis contidas em db.inc.php.

APÊNDICE AB – Resumo de plano de projeto – Iteração 3 – Programador A

Programador	Programador A	**Data**	13/02/2017 - 24/02/2017
Sistema	SIPS	**Programa/Módulo**	Visão Administrativa
Instrutor	*Scrum Master*	**Linguagem**	PHP

Tamanho do Programa	**Atual**	**Até a Data**
Tamanho Total (T)	754 LOC	1491 LOC

Tempo na Fase (min.)	**Real**	**Até a Data**
Planejamento	60	120
Projeto	155	350
Revisão de Projeto	95	245
Código	1925	3350
Revisão do Código	380	660
Compilação	25	35
Teste	225	395
Postmortem	180	335
Total	3045	5490

Defeitos Injetados	**Real**	**Até a Data**
Planejamento	0	0
Projeto	8	20
Revisão de Projeto	0	0
Código	11	26
Revisão do Código	0	0
Compilação	0	0
Teste	0	0
Total	19	46

Defeitos Removidos	Real	Até a Data
Planejamento	0	0
Projeto	0	0
Revisão de Projeto	7	14
Código	0	0
Revisão do Código	8	19
Compilação	0	0
Teste	4	13
Total	19	46

Resumo de Qualidade - PQI - Process Quality Process

	Tempo Projeto	**Tempo de Codificação**	**TOTAL**	Qualidade do projeto, sendo expressa pela divisão entre o tempo de projeto e o tempo de codificação. É recomendado que o tempo gasto na fase de projeto seja maior que o tempo gasto na fase de codificação.
$PQI_{DesignCodeTime}$	155	1925	**0,080519481**	
	Tempo revisão projeto	**Tempo de Projeto**	**TOTAL**	Qualidade de revisão de projeto, sendo expressa pela divisão entre o dobro do tempo de revisão de projeto e o tempo de projeto. É recomendado que o tempo gasto na fase de revisão de projeto seja, no mínimo, metade do tempo gasto na fase de projeto.
$PQI_{DesignReviewTime}$	95	155	**1,225806452**	
	Tempo revisão Código	**Tempo de Codificação**	**TOTAL**	Qualidade de revisão de código, sendo expressa pela divisão entre o dobro do tempo de revisão de código e o tempo de codificação. É recomendado que o tempo gasto na fase de revisão de código seja, no mínimo, a metade do tempo gasto na fase de codificação.
$PQI_{CodeReviewTime}$	380	1925	**0,394805195**	
	Defeitos per KLOC testes unitários		**TOTAL**	Qualidade do programa, sendo expressa pela divisão de 10 pela soma do número de defeitos (encontrados nos testes unitários a cada 1000 linhas de código) com 5, que representa o valor alvo para essa métrica, ou seja, a densidade de defeitos da fase de testes unitários não deve ser superior a 5 defeitos/KLOC.
$PQI_{UnitTestDefects}$	5		1	

APÊNDICE AC – Template de esp. operacional - Iteração 3 – Programador B

Programador	Programador B	Data	13/02/2017
Sistema	SIPS	Programa/Módulo	Visão do Candidato (Inscrição)
Instrutor	*Scrum Master*	Linguagem	PHP

Número do cenário	Cand01	**Objetivo do usuário**	Se inscrever no SIPS para o processo seletivo de professores substitutos e consultar a inscrição.
Objetivo do cenário		Construir as telas e funcionalidades para realizar uma nova Inscrição de candidato no Processo Seletivo de Professores Substitutos (declaração de ciência do edital, escolha do curso, dados pessoais, confirmação dos dados e geração da GRU) e consultar uma inscrição existente.	
Fonte (Source)	**Etapa**	**Ação**	**Comentários**
Usuário	1	Acessa a tela inicial dos Processos Seletivos.	
Sistema	2	Exibe os processos seletivos ativos.	Deve possuir dois *links* de menu: Página inicial e Mais informações (*link* para o *site* do Instituto).
Usuário	3	Clica na opção Inscreva-se, de determinado processo seletivo de professor substituto.	
Sistema	4	Exibe a tela inicial de Processo Seletivo específico, com duas opções: Nova Inscrição e Consultar Inscrição.	- Deve possuir 4*links* de menu: Página inicial, Inscreva-se, Consultar Inscrição e Mais informações. - Se o período de inscrições se encerrou o *link* de menu Inscreva-se não deve ser exibido.
Usuário	5	Clica na opção Nova Inscrição.	
Sistema	6	Exibe a tela Inscreva-se, com resumo do edital e do pagamento.	Pode conter informações resumidas das áreas, da jornada de trabalho e da remuneração.
Usuário	7	Marca*checkbox* de que está ciente das regras do edital e clica em Continuar Inscrição.	Caso não marque o *checkbox* exibir mensagem de erro e não continuar.
Sistema	8	Exibe a tela Inscreva-se – Escolha de Curso (cargo).	
Usuário	9	Seleciona o Cargo e clica em Continuar Inscrição.	Pode também voltar à página inicial ou Cancelar a Inscrição.
Sistema	10	Exibe a tela Inscreva-se – Dados Pessoais.	
Usuário	11	1. Seleciona: Como ficou sabendo do processo seletivo (rádio, televisão, internet, cartaz/folder, amigos/familiares, facebook/twiter ou outros). 2. Informa: nome completo, data de nascimento, sexo, nacionalidade, RG e CPF, nome da mãe e nome do pai. 3. Seleciona: zona rural ou urbana. 4. Informa: endereço residencial, bairro, cidade, CEP, Estado (selecionar), Telefone e*E-mail*. 5. Seleciona sobre deficiência. 6. Seleciona sobre adaptação especial para prova. 7. Clica em Continuar Inscrição	- Validar CPF e *e-mail*. - Opções do item 5: não, visual, auditiva, físico-motora ou intelectual (mental). - Opções do item 6: não, intérprete de libras, prova em braile, acessibilidade arquitetônica, ampliação da prova, ledor/transcritor ou outros. - Se o candidato selecionar "outros" deve digitar a necessidade. - Utilizar máscara para CEP e telefone. - Também pode clicar em Voltar - alterar curso ou Cancelar Inscrição.
Sistema	12	Exibe a tela Inscreva-se – Resumo da Inscrição.	
Usuário	13	Clica em Confirmar Inscrição	- Também pode clicar em Voltar - alterar curso, Voltar - alterar dados pessoais ou Cancelar inscrição
Sistema	14	Exibe a tela Inscreva-se – Gerar Boleto, com informações da inscrição confirmada (incluindo o número de inscrição).	Exibir também opção de Imprimir comprovante de inscrição (com resumo da inscrição e situação (homologada ou não) e Sair.

Usuário	15	Clica em Gerar GRU, com as informações bancárias de pagamento do edital de seleção (código da unidade gestora, código da gestão, o código de recolhimentoe o valor a ser pago).	- GRU gerada no link (http://consulta.tesouro.fazenda.gov.br/gru_novosite/gru_simples.asp). - Sair volta para página inicial dos processos seletivos.
Sistema	16	Abre outra aba de navegação e exibe o site do Tesouro Nacional, para preenchimento e impressão da Guia de Recolhimento da União.	
Usuário	17	Preenche todas as informações e imprime a GRU.	Para as próximas iterações pode-se projetar uma classe Geradora da GRU e outra para do Boleto (no caso de seleção dos cursos técnicos).
Usuário	18	Passa pelas etapas 1, 2, 3, 4 e clica na opção Consultar Inscrição.	
Sistema	19	Exibe a página de área do candidato.	
Usuário	20	Informa CPF data de nascimento e clica no botão Consultar.	Ou o CPF está incorreto/inválido ou a data de nascimento não corresponde ao CPF informado (mensagens).
Sistema	20	Exibe a tela principal de consulta, com número da inscrição, nome completo, cargo escolhido, situação e dois botões (Consultar Dados e Sair).	
Usuário	21	Clica em Consultar dados.	
Sistema	22	Exibe resumo da inscrição: (número da inscrição, nome, RG, CPF, *e-mail*, campus (instituição), curso (cargo), data e hora da inscrição), a situação da inscrição (aguardando homologação ou homologado) e quatro botões: voltar, imprimir comprovante (etapa 14), gerar GRU (etapas 15, 16 e 17) e sair.	- Essas informações são enviadas (após a inscrição) para o *e-mail* informado no ato da inscrição. - O botão deve fazer *log off* no SIPS e retornar à página da etapa 4.

APÊNDICE AD – Template de especificação funcional – Iteração 3 – Prog. B

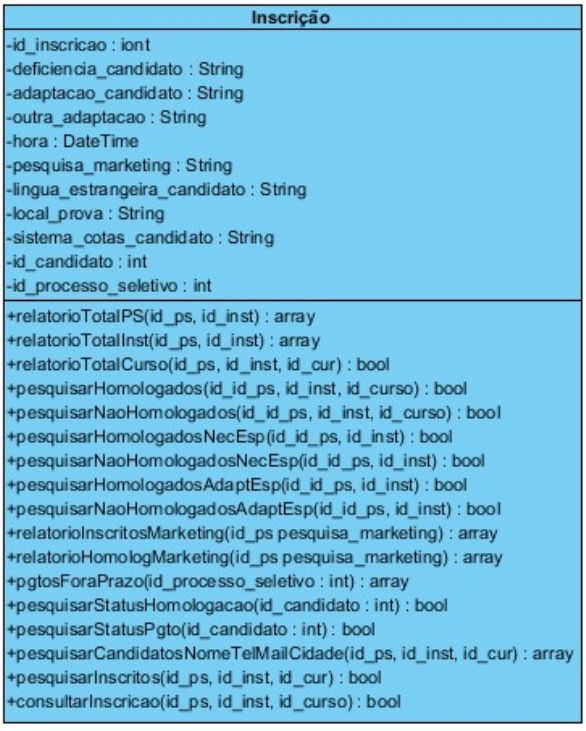
Programador	Programador B	Data	13/02/2017
Sistema	SIPS	Programa/Módulo	Visão do Candidato (Inscrição)
Instrutor	*Scrum Master*	Linguagem	PHP

Nome da Classe	Inscrição
Classe Pai	Generic Db

Atributos	
Declaração	**Descrição**
ID da inscrição	Identificador da inscrição de 10 números.
Deficiência candidato	Deficiência ou não do candidato de até 255 caracteres.
Adaptação candidato	Tipos de adaptações especiais com até 255 caracteres.
Outra adaptação	Descrição de outro tipo de adaptação de até 255 caracteres.
Hora	Data e hora da inscrição.
Pesquisa marketing	Pesquisa de como ficou sabendo do processo seletivo de até 255 caracteres.
Língua estrangeira candidato	Língua estrangeira do candidato de até 8 caracteres (apenas para alunos).
Local prova	Local da prova de até 255 caracteres (apenas para alunos).
Sistema cotas candidato	Tipos de cotas para o candidato (apenas para alunos).
ID candidato	Identificador do candidato de 10 números.
ID processo seletivo	Identificador o processo seletivo de 10 números.

Itens	
Declaração	**Descrição**
RelatorioTotalPS	Operação que retorna os dados e o número total de candidatos do processo seletivo, caso contrário retornar uma mensagem dizendo que o processo não possui candidatos.
RelatorioTotalInst	Operação que retorna os dados e o número total de candidatos do processo seletivo por instituição (campus), caso contrário retornar uma mensagem dizendo que o processo de determinado *campus* não possui candidatos.
RelatorioTotalCurso	Operação que retorna os dados e o número total de candidatos do processo seletivo por curso, caso contrário retornar uma mensagem dizendo que o curso não possui candidatos.
PesquisarHomologados	Operação que retorna os dados dos candidatos homologados, considerando parâmetros de instituição, processo seletivo e curso (cargo), caso contrário retornar uma mensagem dizendo que o curso não possui homologados.
PesquisarNaoHomologados	Operação que retorna os dados dos candidatos não homologados, considerando parâmetros de instituição, processo seletivo e curso (cargo), caso contrário retornar uma mensagem dizendo que o curso não possui candidatos não homologados.
PesquisarHomologadosNecEsp	Operação que retorna os dados dos candidatos homologados e com necessidades especiais, considerando parâmetros de tipo de necessidade, instituição, processo seletivo e curso (cargo), caso contrário retornar uma mensagem dizendo que o curso não possui candidatos homologados com necessidades especiais.
PesquisarNaoHomologadosNecEsp	Operação que retorna os dados dos candidatos não homologados e com necessidades especiais, considerando parâmetros de tipo de necessidade, instituição, processo seletivo e curso (cargo), caso contrário retornar uma mensagem dizendo que o curso não possui candidatos não homologados com necessidades especiais.

PesquisarHomologadosAdaptEsp	Operação que retorna os dados dos candidatos homologados e com adaptações especiais na prova, considerando parâmetros de tipo de adaptação, instituição, processo seletivo e curso (cargo), caso contrário retornar uma mensagem dizendo que o curso não possui candidatos homologados com adaptações especiais.
PesquisarNaoHomologadosAdaptEsp	Operação que retorna os dados dos candidatos não homologados e com adaptações especiais na prova, considerando parâmetros de tipo de adaptação, instituição, processo seletivo e curso (cargo), caso contrário retornar uma mensagem dizendo que o curso não possui candidatos não homologados com adaptações especiais.
RelatorioInscritosMarketing	Operação que retorna o total de inscritos por categoria da pesquisa de marketing, considerando parâmetros de instituição, processo seletivo e curso (cargo).
RelatorioHomologMarketing	Operação que retorna o total de homologados por categoria da pesquisa de marketing, considerando parâmetros de instituição, processo seletivo e curso (cargo).
PgtosForaPrazo	Operação que retorna o número total e os dados dos candidatos com pagamento fora do prazo, considerando parâmetros de parâmetros de instituição, processo seletivo e curso (cargo).
PesquisarStatusHomologacao	Operação que retorna o status do candidato referente à homologação (ou não homologação) em determinado curso, de determinado processo seletivo, de determinada instituição.
PesquisarStatusPgto	Operação que retorna o status do pagamento de determinada inscrição, em determinado curso, de determinado processo seletivo, de determinada instituição.
PesquisarCandidatosNomeTelMailCidade	Operação que retorna os dados (nome, telefone, *e-mail* e cidade) dos candidatos homologados, considerando parâmetros de instituição, processo seletivo e curso (cargo).
PesquisarInscritos	Operação que retorna os dados (ID, nome, telefone e cidade) dos candidatos inscritos, considerando parâmetros de instituição, processo seletivo e curso (cargo).
ConsultarInscricao	Operação consulta inscrição do candidato, com base no CPF e data de nascimento.



APÊNDICE AE – Template de relatório de teste – Iteração 3 – Programador B

Programador	Programador B	Data	16, 22, e 24/02/2017
Sistema	SIPS	Programa/Módulo	Visão do Candidato (Inscrição)
Instrutor	*Scrum Master*	Linguagem	PHP

Nome/Número do Teste	Cand 01.
Objetivo do Teste	Realizar de uma nova inscrição e consultar uma inscrição existente no SIPS.
Descrição do Teste	Na tela principal de inscrições, o candidato deve realizar uma nova inscrição, informando ciência nos dados do edital, seleciona o curso (cargo) para o qual concorre, informa seus dados pessoais e confirma a inscrição. Após, o candidato deve consultar a inscrição realizada, na tela de consulta de inscrições.
Condições do Teste	- Os testes devem ser realizados com usuário Candidato. - Todos os passos, incluindo a geração da GRU, devem ser realizados. - A consulta da inscrição deve ser realizada informando o CPF e data de nascimento.
Resultados esperados	- Inscrição realizada com sucesso. - Dados consistentes tratados corretamente. - Dados inconsistentes tratados corretamente. - GRU gerada com sucesso. - Consulta da inscrição realizada com sucesso.
Resultados reais	- A ação do botão de "Gerar GRU" deve ser aberta em uma nova aba do navegador. - A mensagem de login não realizado com sucesso deve ser melhorada, informando se é o CPF ou a data de nascimento que apresenta problema. - O layout da área da tela principal de consulta deve ser melhorado, alterado paraformato de tabela, com as informações de número da inscrição e nome nas colunas.

APÊNDICE AF – Log de registro de tempo – Iteração 3 – Programador B

Programador	Programador B	**Data**	13/02/2017 - 24/02/2017
Sistema	SIPS	**Programa/Módulo**	Visão do Candidato (Inscrição)
Instrutor	*Scrum Master*	**Linguagem**	PHP

Projeto	Fase	Data e Tempo de Inicio	Tempo de Interrupção	Data e Tempo de Fim	Tempo Delta	Comentários
SIPS	Planejamento	13/02/2017 - 08:00	30	13/02/2017 - 09:00	30	Reunião Scrum. 15min. Outras tarefas. 15min.
SIPS	Projeto	13/02/2017 - 09:00	10	13/02/2017 - 11:00	110	Criação do projeto operacional e fucnional. Outras tarefas. 10min.
SIPS	Revisão do Projeto	13/02/2017 - 11:00	0	13/02/2017 - 12:15	75	Lista de verificação de revisão de projeto.
SIPS	Codificação	13/02/2017 - 12:15	0	13/02/2017 - 13:45	100	Codificação da classe Inscricao.php
SIPS	Codificação	14/02/2017 - 08:10	50	14/02/2017 - 14:00	260	Reunião Scrum. 15min. Responder e-mails. 5min. Suporte a sistemas. 30min. Continuação da codificação Classe Inscricao.class.php
SIPS	Codificação	15/02/2017 - 08:00	60	15/02/2017 - 13:45	285	Reunião Scrum. 15min. Responder e-mails. 15min. Suporte a sistemas. 30min. Conclusão da codificação Classe Inscricao.class.php (239 LOC)
SIPS	Revisão da Codificação	16/02/2017 - 08:05	25	16/02/2017 - 09:40	70	Reunião diária de 15min. Responder e-mails. 10min.
SIPS	Compilação	16/02/2017 - 09:40	0	16/02/2017 - 09:45	5	
SIPS	Teste	16/02/2017 - 09:45	10	16/02/2017 - 11:00	65	Verificação de e-mail. 10min. Utilização do template de teste.
SIPS	Codificação	16/02/2017 - 11:00	0	16/02/2017 - 13:50	75	Codificação dos arquivos: inscricao.php, inscricao_seq1.php, inscricao_seq2.php, inscricao_seq3.php, inscricao_confirmada.php.
SIPS	Codificação	17/02/2017 - 08:00	60	17/02/2017 - 14:00	300	Reunião Scrum. 15min. Responder e-mails. 15min. Suporte a sistemas. 30min. Codificação dos arquivos: inscricao.php, inscricao_seq1.php, inscricao_seq2.php, inscricao_seq3.php, inscricao_confirmada.php.
SIPS	Codificação	20/02/2017 - 08:10	30	20/02/2017 - 14:00	320	Reunião Scrum. 15min. Responder e-mails. 15min. Codificação dos arquivos: inscricao.php, inscricao_seq1.php, inscricao_seq2.php, inscricao_seq3.php, inscricao_confirmada.php.

SIPS	Codificação	21/02/2017 - 08:30	50	21/02/2017 - 13:40	220	Reunião Scrum. 15min. Responder e-mails. 10min. Suporte a sistemas. 25min. Codificação dos arquivos: inscricao.php, inscricao_seq1.php, inscricao_seq2.php, inscricao_seq3.php, inscricao_confirmada.php. (635LOC)
SIPS	Revisão da Codificação	22/02/2017 - 08:05	25	22/02/2017 - 09:40	70	Reunião diária de 15min. Responder e-mails. 10min. Lista de verificação de revisão de codificação.
SIPS	Compilação	22/02/2017 - 09:40	0	22/02/2017 - 09:45	5	
SIPS	Teste	22/02/2017 - 09:45	0	22/02/2017 - 11:10	85	Utilização do template de teste.
SIPS	Codificação	22/02/2017 - 11:10	10	22/02/2017 - 14:00	160	Suporte a sistemas. 10min. Codificação dos arquivos: consultar_inscricao.php e consultando.php.
SIPS	Codificação	23/02/2017 - 08:00	25	23/02/2017 - 14:00	335	Reunião diária de 15min. Responder e-mails. 10min. Codificação dos arquivos: consultar_inscricao.php e consultando.php. (189LOC)
SIPS	Revisão da Codificação	24/02/2017 - 08:00	20	24/02/2017 - 09:35	75	Reunião diária de 15min. Responder e-mails. 10min. Lista de verificação de revisão de codificação.
SIPS	Compilação	24/02/2017 - 09:35	0	24/02/2017 - 09:40	5	
SIPS	Teste	24/02/2017 - 09:40	10	24/02/2017 - 11:15	85	Suporte a sistemas. 10min. Utilização do template de teste.
SIPS	Postmortem	24/02/2017 - 11:15	10	24/02/2017 - 13:45	150	Outras atividades. 10min. Preencimento do formulário de resumo de projeto.

APÊNDICE AG – Log de registro de defeitos – Iteração 3 – Programador B

Programador	Programador B	**Data**	13/02/2017 - 24/02/2017
Sistema	SIPS	**Programa/Módulo**	Visão do Candidato (Inscrição)
Instrutor	*Scrum Master*	**Linguagem**	PHP

Projeto	Data	Número	Tipo	Injetado	Removido	Tempo correção	Ref.
SIPS	13/02/2017	1	10	Projeto	Revisão do Projeto	5min	-

Descrição: O fluxo alternativo da etapa 4, do template de especificação operacional, deve possuir 4 links de menu: Página inicial, Inscreva-se, Consultar Inscrição e Mais informações.

Projeto	Data	Número	Tipo	Injetado	Removido	Tempo correção	Ref.
SIPS	13/02/2017	2	10	Projeto	Revisão do Projeto	5min	-

Descrição: Se o período de inscrições se encerrou o link de menu Inscreva-se não deve ser exibido (template de especificação operacional).

Projeto	Data	Número	Tipo	Injetado	Removido	Tempo correção	Ref.
SIPS	13/02/2017	3	10	Projeto	Revisão do Projeto	5min	-

Descrição: Na escolha do curso, etapa 8, deve exibir também Cargo, pois para professores substitutos cargo equivale à curso(template de especificação operacional).

Projeto	Data	Número	Tipo	Injetado	Removido	Tempo correção	Ref.
SIPS	13/02/2017	4	10	Projeto	Revisão do Projeto	5min	-

Descrição: Na escolha do tipo de deficiência, além de não deve constar: visual, auditiva, físico-motora ou intelectual (mental).

Projeto	Data	Número	Tipo	Injetado	Removido	Tempo correção	Ref.
SIPS	13/02/2017	5	10	Projeto	Revisão do Projeto	5min	-

Descrição: Na etapa 14 o número da inscrição já deve ser exibido.

Projeto	Data	Número	Tipo	Injetado	Removido	Tempo correção	Ref.
SIPS	13/02/2017	6	10	Projeto	Revisão do Projeto	5min	-

Descrição: Para consultar inscrição o candidato deve passar pelas etapas 1, 2, 3, 4 (etapa 18).

Projeto	Data	Número	Tipo	Injetado	Removido	Tempo correção	Ref.

Projeto	Data	Número	Tipo	Injetado	Removido	Tempo correção	Ref.
SIPS	13/02/2017	7	10	Projeto	Revisão do Projeto	5min	-

Descrição: Língua estrangeira, mesmo não sendo para professores substitutos (apenas para alunos), deve nos atributos da classe, porém deve ser opcional (e não visível para candidatos professores).

Projeto	Data	Número	Tipo	Injetado	Removido	Tempo correção	Ref.
SIPS	13/02/2017	8	10	Projeto	Revisão do Projeto	5min	-

Descrição: ID candidato e ID processo seletivo devem ser considerados na classe de Inscrição. (Especificação Funcional).

Projeto	Data	Número	Tipo	Injetado	Removido	Tempo correção	Ref.
SIPS	13/02/2017	9	10	Projeto	Revisão do Projeto	5min	-

Descrição: Deve haver opção para pesquisa de não homolgados também.

Projeto	Data	Número	Tipo	Injetado	Removido	Tempo correção	Ref.
SIPS	13/02/2017	10	10	Projeto	Revisão do Projeto	5min	-

Descrição: No método RelatorioTotalCurso, deve retornar os dados e o número total de candidatos do processo seletivo por curso, caso contrário retornar uma mensagem dizendo que o curso não possui candidatos.

Projeto	Data	Número	Tipo	Injetado	Removido	Tempo correção	Ref.
SIPS	13/02/2017	11	10	Projeto	Revisão do Projeto	5min	-

Descrição: Deve existir método de gerar relatório de marketing tanto para os inscritos quanto para os homologados.

Projeto	Data	Número	Tipo	Injetado	Removido	Tempo correção	Ref.
SIPS	14/02/2017	12	70	Projeto	Codificação	10min	-

Descrição: No resumo do edital, na página inicial da inscrição, deve conter, no mínimo, informações das áreas, da jornada de trabalho e da remuneração.

Projeto	Data	Número	Tipo	Injetado	Removido	Tempo correção	Ref.
SIPS	15/02/2017	13	70	Projeto	Codificação	10min	-

Descrição: Na segunda tela da inscrição (escolha do cargo), deve conter opções de voltar à página inicial ou Cancelar a Inscrição.

Projeto	Data	Número	Tipo	Injetado	Removido	Tempo correção	Ref.
SIPS	16/02/2017	14	80	Codificação	Revisão do Código	10min	-

Descrição: Habilitar para digitação da necessidade especial, na tela de inscrição (dados pessoais), apenas se o candidato selecionar “outros”.

Projeto	Data	Número	Tipo	Injetado	Removido	Tempo correção	Ref.

Projeto	Data	Número	Tipo	Injetado	Removido	Tempo correção	Ref.
SIPS	16/02/2017	15	40	Codificação	Revisão do Código	10min	-

Descrição: A máscara de telefone deve comportar o nono dígito de celular.

Projeto	Data	Número	Tipo	Injetado	Removido	Tempo correção	Ref.
SIPS	22/02/2017	16	40	Codificação	Revisão do Código	10min	-

Descrição: Antes de confirmar inscrição o candidato deve ter a possibilidade de clicar em Voltar - alterar curso, Voltar - alterar dados pessoais ou Cancelar inscrição.

Projeto	Data	Número	Tipo	Injetado	Removido	Tempo correção	Ref.
SIPS	22/02/2017	17	40	Codificação	Testes	5min	-

Descrição: O clique no botão "Gerar GRU" deve abrir outra aba do navegador para o site do tesouro nacional.

Projeto	Data	Número	Tipo	Injetado	Removido	Tempo correção	Ref.
SIPS	22/02/2017	18	60	Codificação	Testes	5min	-

Descrição: Melhorar mensagem de login não realizado com sucesso. Informar se é o CPF ou a data de nascimento que apresentou problema.

Projeto	Data	Número	Tipo	Injetado	Removido	Tempo correção	Ref.
SIPS	24/02/2017	19	60	Codificação	Testes	20min	-

Descrição: Melhorar layout da área da tela principal de consulta, para um formato de tabela, com as informações das colunas: número da inscrição, nome completo, cargo escolhido, situação e dois botões (Consultar Dados e Sair).

Projeto	Data	Número	Tipo	Injetado	Removido	Tempo correção	Ref.
SIPS	24/02/2017	20	70	Codificação	Revisão do Código	10min	-

Descrição: As opções para o status de homologação são aguardando homologação ou homologado.

APÊNDICE AH – Resumo de plano de projeto – Iteração 3 – Programador B

Programador	Programador B	**Data**	13/02/2017 - 24/02/2017
Sistema	SIPS	**Programa/Módulo**	Visão do Candidato (Inscrição)
Instrutor	*Scrum Master*	**Linguagem**	PHP

Tamanho do Programa	**Atual**	**Até a Data**
Tamanho Total (T)	1063 LOC	1938 LOC

Tempo na Fase (min.)	**Real**	**Até a Data**
Planejamento	30	105
Projeto	110	300
Revisão de Projeto	75	225
Codificação	2055	3990
Revisão do Código	215	370
Compilação	15	25
Teste	235	495
Postmortem	150	335
Total	2885	5845

Defeitos Injetados	**Real**	**Até a Data**
Planejamento	0	0
Projeto	13	22
Revisão de Projeto	0	0
Codificação	7	21
Revisão do Código	0	0
Compilação	0	0

Teste	0	0
Total	20	43

Defeitos Removidos	**Real**	**Até a Data**
Planejamento	0	0
Projeto	0	0
Revisão de Projeto	11	17
Codificação	2	2
Revisão do Código	4	11
Compilação	0	0
Teste	3	13
Total	20	43

Resumo de Qualidade	**PQI - Process Quality Process**			
	Tempo Projeto	**Tempo de Codificação**	**TOTAL**	
$PQI_{DesignCodeTime}$	110	2055	**0,053528**	Qualidade do projeto, sendo expressa pela divisão entre o tempo de projeto e o tempo de codificação. É recomendado que o tempo gasto na fase de projeto seja maior que o tempo gasto na fase de codificação.
	Tempo Revisão Projeto	**Tempo de Projeto**	**TOTAL**	
$PQI_{DesignReviewTime}$	75	110	**1,3636364**	Qualidade de revisão de projeto, sendo expressa pela divisão entre o dobro do tempo de revisão de projeto e o tempo de projeto. É recomendado que o tempo gasto na fase de revisão de projeto seja, no mínimo, metade do tempo gasto na fase de projeto.
	Tempo Revisão Código	**Tempo de Codificação**	**TOTAL**	
$PQI_{CodeReviewTime}$	215	2055	**0,2092457**	Qualidade de revisão de código, sendo expressa pela divisão entre o dobro do tempo de revisão de código e o tempo de codificação. É recomendado que o tempo gasto na fase de revisão de código seja, no mínimo, a metade do tempo gasto na fase de codificação.
	Defeitos per KLOC testes unitários		**TOTAL**	
$PQI_{UnitTestDefects}$	3		1,25	Qualidade do programa, sendo expressa pela divisão de 10 pela soma do número de defeitos (encontrados nos testes unitários a cada 1000 linhas de código) com 5, que representa o valor alvo para essa métrica, ou seja, a densidade de defeitos da fase de testes unitários não deve ser superior a 5 defeitos/KLOC.

APÊNDICE AI – Template de esp. operacional - Iteração 3 – Programador C

Programador	Programador C	Data	13/02/2017
Sistema	SIPS	Programa/Módulo	Visão Usuário de Relatórios (Candidato)
Instrutor	*Scrum Master*	Linguagem	PHP

Número do cenário	Rel01	**Objetivo do usuário**	Utilizar as opções de Relatórios e Administração de Pagamentos de Inscrições para Professores Substitutos no SIPS.
Objetivo do cenário		Utilizar as telas de *login* e página inicial e projetar página de administração para usuário de relatório. Esses usuários são os servidores responsáveis por esses processos nas unidades (*campi*). Geralmente, em exercício no setor de Recursos Humanos das Unidades da Instituição.	
Fonte (Source)	**Etapa**	**Ação**	**Comentários**
Usuário	01	Realiza as etapas 01, 02 e 03, 04 e 05 do Template de Especificação Operacional - I3P1 - Visão Admin.	Utiliza os *templates* (páginas PHP) criados pelo programador A, na iteração 3 para login, página inicial e menus.
Sistema	02	Exibe a página Administrar, com a opção de Pagamento.	
Usuário	03	Realiza as etapas 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31 e 32 do Template de Especificação Operacional - I3P1 - Visão Admin para administrar os pagamentos dos inscritos para seleção de professores substitutos.	Utiliza os *templates* (páginas PHP)criados pelo programador A, na iteração 3 para pagamentos, consultar pagamentos, confirmar pagamentos e gerar homologação.
Usuário	04	Acessa menu Relatórios.	I3P1.
Sistema	05	Exibe a página de Relatórios: Inscritos, Homologados, Isentos, Cotistas, Portadores de Necessidades Especiais, Pagamentos e Marketing.	- I3P1. - Cada opção de relatório deve possuir um ícone e levar para uma página específica. - Isentos e Cotistas serão implementados em outra iteração.
Usuário	06	Clica em Relatórios de Inscritos.	
Sistema	07	Exibe a página de relatórios dos Inscritos, com parâmetros de pesquisa por: Relatório (Total de inscritos no Processo Seletivo, Relação do nome dos inscritos, Relação do nome e telefone dos inscritos, Relação do nome e e-*mail*dos inscritos e Relação do nome, telefone e cidade de origem dos inscritos), Instituição e Curso.	
Usuário	08	Seleciona os parâmetros necessários e clica no botão Gerar Relatório.	Ou clica no botão Voltar (retorna para a página da etapa 05).
Sistema	09	Exibe o resultado, em forma de tabela, de acordo com os parâmetros selecionados na etapa 08 (exibindo o número total de inscritos na parte inferior), com a opção de imprimir o relatório.	No cabeçalho do resultado deve ser exibido o nome do Campus, o tipo de relatório, o curso (cargo) e a carga horária (20h ou 40h).
Usuário	06	Clica em Relatórios de Homologados.	
Sistema	07	Exibe a página de relatórios de Homologados, com parâmetros de pesquisa por: Relatório (Homologados, não homologados), Instituição e Curso.	
Usuário	08	Seleciona os parâmetros necessários e clica no botão Gerar Relatório.	Ou clica no botão Voltar (retorna para a página da etapa 05).
Sistema	09	Exibe o resultado, em forma de tabela, de acordo com os parâmetros selecionados na etapa 08 (exibindo o número total de inscritos na parte inferior), com a opção de imprimir o relatório.	No cabeçalho do resultado deve ser exibido o nome do Campus, o tipo de relatório, o curso (cargo) e a carga horária (20h ou 40h).
Usuário	06	Clica em Relatórios de PNE.	

Sistema	07	Exibe a página de relatórios de Portadores de Necessidades Especiais, com parâmetros de pesquisa por: Relatório (Números PNEs inscritos, Número de adaptações especiais inscritos, Número de PNE´s homologados, Número de adaptações especiais homologados), Instituição e Curso.	
Usuário	08	Seleciona os parâmetros necessários e clica no botão Gerar Relatório.	Ou clica no botão Voltar (retorna para a página da etapa 05).
Sistema	09	Exibe o resultado, em forma de tabela, de acordo com os parâmetros selecionados na etapa 08 (exibindo o número total de inscritos na parte inferior), com a opção de imprimir o relatório.	No cabeçalho do resultado deve ser exibido o nome do Campus, o tipo de relatório, o curso (cargo) e a carga horária (20h ou 40h).
Usuário	06	Clica em Relatórios de Marketing.	
Sistema	07	Exibe a página de relatórios de Pesquisa de Marketing, com parâmetros de pesquisa por: Relatório (Candidatos inscritos e candidatos homologados), Instituição e Curso.	
Usuário	08	Seleciona os parâmetros necessários e clica no botão Gerar Relatório.	Ou clica no botão Voltar (retorna para a página da etapa 05).
Sistema	09	Exibe o resultado, em forma de tabela, de acordo com os parâmetros selecionados na etapa 08 (exibindo o número total de inscritos na parte inferior), com a opção de imprimir o relatório.	No cabeçalho do resultado deve ser exibido o nome do Campus, o tipo de relatório, o curso (cargo) e a carga horária (20h ou 40h).
Usuário	06	Clica em Relatórios de Pagamentos.	
Sistema	07	Exibe a página de relatórios de Pagamentos, com parâmetros de pesquisa por: Relatório (Pagamentos processados e Pagamentos não processados), Instituição e Curso.	
Usuário	08	Seleciona os parâmetros necessários e clica no botão Gerar Relatório.	Ou clica no botão Voltar (retorna para a página da etapa 05).
Sistema	09	Exibe o resultado, em forma de tabela, de acordo com os parâmetros selecionados na etapa 08 (exibindo o número total de inscritos na parte inferior), com a opção de imprimir o relatório.	No cabeçalho do resultado deve ser exibido o nome do Campus, o tipo de relatório, o curso (cargo) e a carga horária (20h ou 40h).

APÊNDICE AJ – Template de especificação funcional – Iteração 3 – Prog. C

Programador	Programador C	Data	13/02/2017
Sistema	SIPS	Programa/Módulo	Visão Usuário de Relatórios (Candidato)
Instrutor	*Scrum Master*	Linguagem	PHP

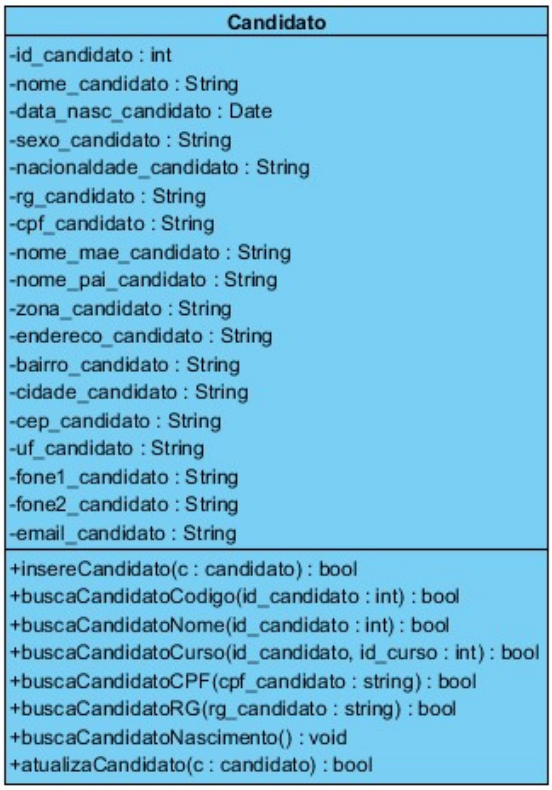
Nome da Classe	Candidato
Classe Pai	Generic Db Inscrição

Atributos

Declaração	**Descrição**
ID	ID do candidato com 10 números.
Nome	Nome do candidato com até 80 caracteres.
Data de nascimento	Data de nascimento do candidato, no formato dd/mm/aaaa.
Sexo	Sexo do candidato com até 9 caracteres.
Nacionalidade	Nome do candidato com até 20 caracteres.
RG	RG do candidato com 11 caracteres.
CPF	CPF do candidato com 14 caracteres.
Nome da mãe	Nome da mãe do candidato com até 80 caracteres.
Nome do pai	Nome do pai candidato com até 80 caracteres.
Zona	Zona do candidato com até 6 caracteres.
Endereço	Endereço do candidato com até 100 caracteres.
Bairro	Bairro do candidato com até 50 caracteres.
Cidade	Cidade do candidato com até 50 caracteres.
CEP	CEP da cidade do candidato com 9 caracteres.
UF	UF do candidato com 2 caracteres.
Telefone1	Telefone principal do candidato com 14 caracteres.
Telefone2	Telefone secundário do candidato com 14 caracteres.
E-mail	E-mail do candidato com até 70 caracteres.

Itens

Declaração	**Descrição**
Insere Candidato	Insere todos os dados de um novo candidato em uma inscrição de um determinado curso. A tabela de união deve ser considerada (inst_proc-sel_cur).
Busca Candidato por Código	Recebe o ID do candidato (nulo) e retorna o curso do candidato ou uma mensagem de aviso de código não encontrado (não há dados na tabela).
Busca Candidato por Nome	Recebe o ID do candidato e retorna o nome do candidato ou uma mensagem de aviso de código não encontrado. Utilizar variáveis de sessão para iniciar a consulta SQL.
Busca Candidato por Curso	Recebe o CPF do candidato e retorna o curso que o candidato se inscreveu ou uma mensagem de código não encontrado.
Busca Candidato por CPF	Recebe o CPF do candidato e retorna a inscrição do candidato ou uma mensagem de código não encontrado.
Busca Candidato por RG	Recebe o RG do candidato e retorna a inscrição do candidato ou uma mensagem de código não encontrado.
Busca Candidato por Data de Nascimento	Retorna a data de nascimento do candidato ou uma mensagem que não há dados nessa tabela.
Atualiza Candidato	Recebe todos os parâmetros da inserção, verifica se o código existe, altera e exibe mensagem de alteração realizada com sucesso.


Candidato
-id_candidato : int
-nome_candidato : String
-data_nasc_candidato : Date
-sexo_candidato : String
-nacionaldade_candidato : String
-rg_candidato : String
-cpf_candidato : String
-nome_mae_candidato : String
-nome_pai_candidato : String
-zona_candidato : String
-endereco_candidato : String
-bairro_candidato : String
-cidade_candidato : String
-cep_candidato : String
-uf_candidato : String
-fone1_candidato : String
-fone2_candidato : String
-email_candidato : String
+insereCandidato(c : candidato) : bool
+buscaCandidatoCodigo(id_candidato : int) : bool
+buscaCandidatoNome(id_candidato : int) : bool
+buscaCandidatoCurso(id_candidato, id_curso : int) : bool
+buscaCandidatoCPF(cpf_candidato : string) : bool
+buscaCandidatoRG(rg_candidato : string) : bool
+buscaCandidatoNascimento() : void
+atualizaCandidato(c : candidato) : bool

APÊNDICE AL – Template de relatório de teste – Iteração 3 – Programador C

Programador	Programador C	Data	23/02/2017
Sistema	SIPS	Programa/Módulo	Visão Usuário de Relatórios (Candidato)
Instrutor	*Scrum Master*	Linguagem	PHP

Nome/Número do Teste	Rel01.
Objetivo do Teste	Consultar e confirmar pagamentos, bem como testar os relatórios de Inscritos, Homologados, Portadores de Necessidades Especiais, Pagamentos e Marketing do SIPS.
Descrição do Teste	Acessar a tela administrativa para utilizar as funcionalidades de pagamento e os tipos de relatórios.
Condições do Teste	- Testar todos os parâmetros para os relatórios. - Testar a exibição desses relatórios. - Consultar pagamentos de inscritos. - Confirmar pagamento de inscritos.
Resultados esperados	- Pagamentos consultados e confirmados. - Relatórios de Inscritos, Homologados, Portadores de Necessidades Especiais, Pagamentos e Marketing devidamente gerados e exibidos.
Resultados reais	- Na função de update, buscar o candidato por CPF antes da query (Retorna false caso não exista um candidato com o código informado). - Melhorar a mensagem da linha 73, exibir: "O código Informado não foi encontrado!"

APÊNDICE AM – Log de registro de tempo – Iteração 3 – Programador C

Programador	Programador C	**Data**	13/02/2017 - 24/02/2017
Sistema	SIPS	**Programa/Módulo**	Visão Usuário de Relatórios (Candidato)
Instrutor	*Scrum Master*	**Linguagem**	PHP

Projeto	Fase	Data e Tempo de Inicio	Tempo de Interrupção	Data e Tempo de Fim	Tempo Delta	Comentários
SIPS	Planejamento	13/02/2017 - 08:05	30	13/02/2017 - 09:15	40	Reunião (15min). Checar e-mails (15min).
SIPS	Projeto	13/02/2017 - 09:15	0	13/02/2017 - 10:50	95	Especificação operacional e fucnional.
SIPS	Revisão do Projeto	13/02/2017 - 10:50	0	13/02/2017 - 12:00	70	Lista de verificação de revisão de projeto.
SIPS	Codificação	13/02/2017 - 12:00	20	13/02/2017 - 13:45	85	Início da criação da Classe Candidato.class.php.
SIPS	Codificação	14/02/2017 - 08:00	60	14/02/2017 - 14:00	300	Reunião (15min). Checar-mails (10min). Suporte a usuários (35min). Continuação da codificação Classe Candidato.class.php.
SIPS	Codificação	15/02/2017 - 08:00	40	14/02/2017 - 13:50	310	Reunião (15min). Checar-mails (10min). Suporte a usuários (15min). Conclusão da codificação Classe Candidato.class.php. (260LOC)
SIPS	Revisão da Codificação	16/02/2017 - 08:15	30	16/02/2017 - 10:00	75	Reunião (15min). Checar-mails (5min). Suporte a usuários (10min). Checklist de revisão de código.
SIPS	Compilação	16/02/2017 - 10:00	0	16/02/2017 - 10:05	5	
SIPS	Teste	16/02/2017 - 10:05	25	16/02/2017 - 12:30	120	Reunião (15min). Checar e-mails (10min). Utilização do template de teste.
SIPS	Codificação	16/02/2017 - 12:30	0	16/02/2017 - 14:00	90	Início da codificação dos arquivos: relatorio_inscritos.php, relatorio_homologados.php, relatorio_pnes.php, relatorio_pagamentos.php e relatorio_marketing.php.
SIPS	Codificação	17/02/2017 - 08:15	35	17/02/2017 - 13:50	305	Reunião (15min). Checar e-mails (10min). Continuação da codificação dos arquivos: rrelatorio_inscritos.php, relatorio_homologados.php, relatorio_pnes.php, relatorio_pagamentos.php e relatorio_marketing.php.
SIPS	Codificação	20/02/2017 - 08:00	50	20/02/2017 - 13:50	300	Reunião (15min). Checar e-mails (10min). Suporte a usuários (25min). Continuação da codificação dos arquivos: relatorio_inscritos.php, relatorio_homologados.php, relatorio_pnes.php, relatorio_pagamentos.php e relatorio_marketing.php.

SIPS	Codificação	21/02/2017 - 08:10	40	21/02/2017 - 14:00	310	Reunião (15min). Checar e-mails (10min). Suporte a usuários (15min). Continuação da codificação dos arquivos: relatorio_inscritos.php, relatorio_homologados.php, relatorio_pnes.php, relatorio_pagamentos.php e relatorio_marketing.php.
SIPS	Codificação	22/02/2017 - 08:00	45	22/02/2017 - 13:45	300	Reunião (15min). Checar e-mails (10min). Suporte a usuários (25min). Continuação da codificação dos arquivos:relatorio_inscritos.php, relatorio_homologados.php, relatorio_pnes.php, relatorio_pagamentos.php e relatorio_marketing.php. (604LOC)
SIPS	Revisão da Codificação	23/02/2017 - 08:00	25	23/02/2017 - 10:05	100	Reunião (15min). Checar e-mails (10min). Suporte a usuarios (10min). Checklist de revisão de código.
SIPS	Compilação	23/02/2017 - 10:05	0	23/02/2017 - 10:10	5	
SIPS	Teste	23/02/2017 - 10:10	0	23/02/2017 - 13:50	220	Utilização do template de teste.
SIPS	Postmortem	24/02/2017 - 08:00	55	24/02/2017 - 11:00	125	Reunião (15min). Checar e-mails (10min). Suporte a usuários (30min). Preencimento da planilha de resumo de projeto.

APÊNDICE AN – Log de registro de defeitos – Iteração 3 – Programador C

Programador	Programador C	**Data**	13/02/2017 - 24/02/2017
Sistema	SIPS	**Programa/Módulo**	Visão Usuário de Relatórios (Candidato)
Instrutor	*Scrum Master*	**Linguagem**	PHP

Projeto	Data	Número	Tipo	Injetado	Removido	Tempo correção	Ref.
SIPS		1	10	Projeto	Revisão do Projeto	5min	-

Descrição: Explicitar de qual documento as etapas 1, 2, 3, 4 e 5 se referem (Especificação Operacional - I3P1 - Visão Admin).

Projeto	Data	Número	Tipo	Injetado	Removido	Tempo correção	Ref.
SIPS		2	10	Projeto	Revisão do Projeto	5min	-

Descrição: Adicionar as etapas 31 e 32 do Template de Especificação Operacional - I3P1 - Visão Admin à etapa 03 Especificação Operacional Visão Usuário de Relatórios.

Projeto	Data	Número	Tipo	Injetado	Removido	Tempo correção	Ref.
SIPS		3	10	Projeto	Revisão do Projeto	5min	-

Descrição: Adicionar Isentos e Cotistas à página de relatórios (mesmo não sendo implementadas nessa iteração).

Projeto	Data	Número	Tipo	Injetado	Removido	Tempo correção	Ref.
SIPS		4	10	Projeto	Revisão do Projeto	5min	-

Descrição: Deve existir também o relatório com a relação do nome, telefone e cidade de origem dos inscritos.

Projeto	Data	Número	Tipo	Injetado	Removido	Tempo correção	Ref.
SIPS		5	80	Projeto	Revisão do Projeto	5min	-

Descrição: Adicionar fluxo alternativo de retorno para a página da etapa 05, clicando no botão Voltar (Especificação Operacional Visão Usuário de Relatórios).

Projeto	Data	Número	Tipo	Injetado	Removido	Tempo correção	Ref.
SIPS		6	80	Projeto	Revisão do Projeto	5min	-

Descrição: Adicionar Telefone2 à Especificação Funcional - Visão Usuário de Relatórios.

Projeto	Data	Número	Tipo	Injetado	Removido	Tempo correção	Ref.
SIPS		7	80	Codificação	Revisão do Código	15min	-

Descrição: Deve existir uma Busca Candidato por Data de Nascimento, que retorna a data ou uma mensagem de erro.

Projeto	Data	Número	Tipo	Injetado	Removido	Tempo correção	Ref.
SIPS		8	50	Codificação	Revisão do Código	5min	-

Descrição: Inscrição deve ser considerada como classe pai de candidato.

Projeto	Data	Número	Tipo	Injetado	Removido	Tempo correção	Ref.
SIPS		9	60	Codificação	Revisão do Código	5min	-

Descrição: Requer a classe Inscricao.class.php. (Candidato.class.php)

Projeto	Data	Número	Tipo	Injetado	Removido	Tempo correção	Ref.
SIPS		10	60	Codificação	Testes	5min	-

Descrição: Melhorar a mensagem da linha 73 ("Código Informado não foi encontrado!")

Projeto	Data	Número	Tipo	Injetado	Removido	Tempo correção	Ref.
SIPS		11	60	Codificação	Revisão do Código	10min	-

Descrição: Erro na nacionalidade do candidato (add na função de inserção).

Projeto	Data	Número	Tipo	Injetado	Removido	Tempo correção	Ref.
SIPS		12	60	Codificação	Testes	10min	-

Descrição: Na função de update, buscar o candidato por CPF antes da query (Retorna false caso não exista um candidato com o código informado).

APÊNDICE AO – Resumo de plano de projeto– Iteração 3 – Programador C

Programador	Programador C	**Data**	13/02/2017 - 24/02/2017
Sistema	SIPS	**Programa/Módulo**	Visão Usuário de Relatórios (Candidato)
Instrutor	*Scrum Master*	**Linguagem**	PHP

Tamanho do Programa	**Atual**	**Até a Data**
Tamanho Total (T)	864 LOC	1518 LOC

Tempo na Fase (min.)	**Real**	**Até a Data**
Planejamento	40	85
Projeto	95	260
Revisão de Projeto	70	180
Código	2000	3930
Revisão do Código	175	310
Compilação	10	20
Teste	340	640
Postmortem	125	340
Total	2855	5765

Defeitos Injetados	**Real**	**Até a Data**
Planejamento	0	0
Projeto	6	15
Revisão de Projeto	0	0
Código	6	16
Revisão do Código	0	0
Compilação	0	0
Teste	0	0

Total	12	31

Defeitos Removidos	**Real**	**Até a Data**
Planejamento	0	0
Projeto	0	0
Revisão de Projeto	6	12
Código	0	0
Revisão do Código	4	10
Compilação	0	0
Teste	2	9
Total	12	31

Resumo de Qualidade - PQI - Process Quality Process

$PQI_{DesignCodeTime}$	**Tempo Projeto** 95	**Tempo de Codificação** 2000	**TOTAL** **0,0475**	Qualidade do projeto, sendo expressa pela divisão entre o tempo de projeto e o tempo de codificação. É recomendado que o tempo gasto na fase de projeto seja maior que o tempo gasto na fase de codificação.
$PQI_{DesignReviewTime}$	**Tempo revisão projeto** 70	**Tempo de Projeto** 95	**TOTAL** **1,47368421**	Qualidade de revisão de projeto, sendo expressa pela divisão entre o dobro do tempo de revisão de projeto e o tempo de projeto. É recomendado que o tempo gasto na fase de revisão de projeto seja, no mínimo, metade do tempo gasto na fase de projeto.
$PQI_{CodeReviewTime}$	**Tempo revisão Código** 175	**Tempo de Codificação** 2000	**TOTAL** **0,175**	Qualidade de revisão de código, sendo expressa pela divisão entre o dobro do tempo de revisão de código e o tempo de codificação. É recomendado que o tempo gasto na fase de revisão de código seja, no mínimo, a metade do tempo gasto na fase de codificação.
$PQI_{UnitTestDefects}$	**Defeitos per KLOC testes unitários** 2		**TOTAL** 1,428571429	Qualidade do programa, sendo expressa pela divisão de 10 pela soma do número de defeitos (encontrados nos testes unitários a cada 1000 linhas de código) com 5, que representa o valor alvo para essa métrica, ou seja, a densidade de defeitos da fase de testes unitários não deve ser superior a 5 defeitos/KLOC.

ANEXO A – PSP defect type standard

Type Number	Type Name	Description
10	Documentation	Comments, messages
20	Syntax	Spelling, punctuation, typos, instruction formats
30	Build, Package	Change management, library, version control
40	Assignment	Declaration, duplicate names, scope, limits
50	Interface	Procedure calls and references, I/O, user formats
60	Checking	Error messages, inadequate checks
70	Data	Structure, content
80	Function	Logic, pointers, loops, recursion, computation, function defects
90	System	Configuration, timing, memory
100	Environment	Design, compile, test, or other support system problems