PONTIFÍCIA UNIVERSIDADE CATÓLICA DE GOIÁS
ESCOLA DE CIÊNCIAS EXATAS E DA COMPUTAÇÃO
GRADUAÇÃO EM ENGENHARIA DE COMPUTAÇÃO

SCRUM E PERSONAL SOFTWARE PROCESS: UM FRAMEWORK DE GESTÃO DE PROJETO PARA DESENVOLVEDORES STANDALONE

PABLO DIAS COUTO

GOIÂNIA
2020

PABLO DIAS COUTO

SCRUM E PERSONAL SOFTWARE PROCESS: UM FRAMEWORK DE GESTÃO DE PROJETO PARA DESENVOLVEDORES STANDALONE

Trabalho de Conclusão de Curso apresentado a Escola de Ciências Exatas e da Computação da Pontifícia Universidade Católica de Goiás como requisito parcial para obtenção do título de Bacharel em Engenharia da Computação.

Orientador: Rafael Leal Martins

GOIÂNIA

2020

PABLO DIAS COUTO

SCRUM E PERSONAL SOFTWARE PROCESS: UM FRAMEWORK DE GESTÃO DE PROJETO PARA DESENVOLVEDORES STANDALONE

Este Trabalho de Conclusão de Curso julgado adequado para obtenção o título de Bacharel em Engenharia de Computação, e aprovação em sua forma final pela Escola de Ciências Exatas e da Computação, da Pontifícia Universidade Católica de Goiás, em ___/____/_____.

_____

Prof. Ma. Ludmilla Reis Pinheiro dos Santos

Coord. de Trabalho de Conclusão de Curso

Banca examinadora:

_____

Orientador: Rafael Leal Martins

_____

Fabricio Schlag

_____

Adriana Siveira de Souza

GOIÂNIA

2020

AGRADECIMENTOS

Agradeço primeiramente a minha mãe, Jussaria por me proporcionar a oportunidade de investir nos meus estudos e me suprir de todo o apoio necessário para continuar firme nessa caminhada.

Aos professores, que compartilharam de seus conhecimentos comigo tornando possível a realização deste trabalho.

Finalmente, agradeço aos meus amigos e colegas de classes.

A vocês, o meu Muito Obrigado!

RESUMO

Este trabalho apresenta a proposta de um framework ágil de projetos para desenvolvedores *standalone*, através da adaptação dos papéis, eventos e artefatos do *Scrum*, junto à adoção de elementos e práticas do *Personal Software Process* (PSP). Discorre sobre os conceitos básicos de projetos e seu ciclo de vida. Descreve de forma geral o framework de gestão de projetos ágeis *Scrum* e o método de melhoria do processo pessoal PSP.Com este trabalho, espera-se que qualquer indivíduo que trabalhe em projetos de desenvolvimento de software sozinhos, em formato *standalone*, possam utilizar-se do *framework* proposto para entregar projetos de qualidade seguindo práticas ágeis presentes no *Scrum* e elementos de gestão pessoal do PSP.

Palavras-chave: Ágil, Projetos, Autogestão, *Scrum*, PSP.

ABSTRACT

This work presents a proposal for an agile project framework for autonomous developers, through the adaptation of Scrum roles, events and artifacts, along with the adoption of elements and practices of the Personal Software Process (PSP). Discusses the basic concepts of projects and their life cycle. General description of the Scrum agile project management framework and the method for improving the personal PSP process. With this work, it is expected that any individual who works on software development projects alone, in an autonomous format, can use the proposed framework to deliver quality projects following agile practices present in the Scrum framework and personal management elements of the PSP.

Keywords: Agile, Projects, Self-management, Scrum, PSP.

LISTA DE FIGURAS

LISTA DE SIGLAS

PMBOK - *Project Management Body of Knowledge*

PROBE - *Proxy-Based Estimating*

PSP - *Personal Software Process*

PSP0-RPP - PSP0 Resumo de Plano de Projeto

PWC - *PricewaterhouseCoopers*

RUP - *Rational Unified Process*

UML - *Unified Modelling Language*

XP - *Extreme Programming*

SUMÁRIO

1. INTRODUÇÃO

Com o notório aumento em demandas de projetos de software em um cenário totalmente globalizado e aliado ao crescimento no número de desenvolvedores que trabalham de forma autônoma e remota, da qual chamaremos aqui de desenvolvedores *standalone*, é natural que metodologias de gerência de projetos ágeis e processos de melhorias de qualidade de softwares, sejam cada vez mais procurados por aqueles que, mesmo trabalhando sozinhos, querem manter certa qualidade e eficiência em seus projetos.

Dentre os métodos voltados para autogestão temos o *Personal Software Process* (PSP), que visa auxiliar desenvolvedor de software, através de processos pessoais, a controlar, gerenciar e alcançar um nível elevado de qualidade em seus projetos, no entanto o PSP não é uma metodologia para desenvolvimento de projetos de software e sim um método genérico de processos, uma proposta teórica para que um processo possa ser elaborado e colocado em prática (JÚNIOR, 2000).

Diante deste contexto e levando em consideração que grande parte das metodologias e *frameworks* de gestão de projetos são voltadas para equipes, é de grande valia analisar dentro do *framework* ágil mais utilizada no mercado, o *Scrum* (COLLABNET VERSIONONE, 2019), eventos, artefatos e práticas que podem ser adaptadas e usadas em conjunto com o PSP, a fim, de criar um *framework* de gerencia de projeto de software pessoal.

1.1 Objetivos Gerais

Propor um *framework* híbrido entre o *Scrum* e o método PSP, voltado aos desenvolvedores de software *standalone*.

1.1.1 Objetivos Específicos

- Reconhecer e apresentar os principais conceitos do *framework Scrum*.
- Estudar e descrever os principais conceitos do PSP.
- Acentuar eventos, artefatos e funções do *Framework Scrum* para desenvolvedores *standalone*.
- Utilizar elementos do PSP que podem ser adotados de forma hibrida com o *framework Scrum*.

1.2 Justificativa

É de suma importância se desenvolver este, pois, no contexto atual temos uma parcela consideráveis de desenvolvedores que trabalham em projetos software sozinhos, e ainda uma grande parte das metodologias e *frameworks* ágeis para gerenciamento de projetos de software são voltadas para trabalhos em equipe. Podendo assim, com o *framework* proposto, disseminar para essa comunidade de desenvolvedores, processos e práticas ágeis que irão auxiliá-los em seus projetos, garantindo entregas ágeis e de qualidade.

De acordo com a pesquisa anual do *Stack Overflow*, um dos maiores fóruns de desenvolvedores e programadores do mundo, 33,2% dos desenvolvedores entrevistados prefere trabalhar de casa (Stack Overflow, 2019). Com o surgimento da pandemia do novo corona vírus (COVID-19) em 2020, e as medidas emergências tomadas por empresas em todo o mundo, especialistas em gestão de negócios, como Andre Miceli, preveem crescimento de 30% no trabalho remoto mesmo após o período de estabilização dos casos de infecção e retomada normal das atividades (UOL, 2020).

Com isso é importante ressaltar o fato de que, desenvolvedores que trabalham sozinhos em projetos de softwares não estão imunes aos riscos inerentes a má gestão. Ainda existem inúmeros desafios no gerenciamento de projetos de desenvolvimento de software da qual todos aqueles que querem manter a qualidade e a satisfação de seus clientes, precisam se atentar. Dentre eles, a dificuldade que executores de projetos de software têm de entenderem os reais problemas levantados por seus clientes acarretando entregas que não demonstram valor ao negócio do cliente (SILVA & LOVATO, 2016). As constantes falhas em estimativas que levam a atrasos em cronograma e não cumprimento de prazos estipulados para o projeto é outro problema bastante presente em projetos mal geridos (PWC, 2012). E podemos citar a dificuldade em lidar com as mudanças de requisitos que por consequência afetam os escopos de diversos projetos de software em fases intermediarias e finais. Todos esses problemas e muitos outros já são enfrentados há muito tempo por metodologias consideradas ágeis, e com a formalização dos princípios e práticas ágeis consolidados no manifesto ágil (BECK et al., 2001), inúmeras metodologias e

frameworks com essas características passaram a fazer parte da rotina de quem desenvolve softwares. Também vale ressaltar que atualmente os métodos ágeis são considerados uma boa prática de projetos amplamente difundida pelo mundo.

1.3 Metodologia

Para alcançar os objetivos desse trabalho, foi realizado um trabalho de pesquisa de natureza aplicada, onde o objetivo é geração de conhecimento para uma aplicação prática dirigida a solução de um problema (GERHARDT & SILVEIRA, 2009).

Desta forma, foi realizada uma revisão da literatura a respeito projetos de software e seus ciclos de vida, do *framework* ágil *Scrum* e do PSP, com intuito de levantar informações cientificas relevantes que irão servir de base para compor a solução proposta, o framework ágil de projetos para desenvolvedores *standalone*.

2. REFERENCIAL TEÓRICO

O trabalho desempenhado para um projeto de software pode por si só não ser uma tarefa fácil, por isso, um dos requisitos primordiais para se ter sucesso é um bom gerenciamento do projeto. Um projeto é um esforço temporário empreendido para criar um produto, serviço ou resultado único. Além disso, um projeto poderá envolver um único individuo ou um grupo. A aplicação de

conhecimento, habilidades, ferramentas e técnicas em prol do cumprimento dos requisitos de um projeto de software são elementos que compõe o gerenciamento de um projeto (PMBOK, 2017).

De acordo com o guia PMBOK (2017), um bom gerenciamento de projetos de software é capaz de trazer os seguintes benefícios: cumprimentos dos objetivos de negócio, satisfação das expectativas dos *stakeholder*, previsibilidade, aumento das chances de sucesso, entregas certas em momento certo, resoluções de problemas, resposta a riscos em tempo hábil, otimização de uso de recursos organizacionais, identificação, recuperação ou descarte de projetos com problemas, gerenciamento de restrições e gerenciamento de mudanças. Em contrapartida, também é citado os riscos de um mau gerenciamento ou a ausência completa do gerenciamento de projeto, o que acarreta em: perca de prazos, estouros de orçamentos, má qualidade, retrabalho, expansão descontrolada, perca de reputação, insatisfação dos *stakeholders* e a incapacidade de alcançar os objetivos para os quais o projeto foi idealizado.

Ainda de acordo com o PMBOK (2017) um projeto é algo que impulsiona mudança dentro de uma organização, que permite uma alteração de um estado atual para um estado futuro passando por um estado de transição. Espera-se que o estado futuro alcançado ao fim de um projeto seja algo que gere valor para o negócio, o valor do negócio deve ser um benefício líquido mensurável e derivado de um empreendimento de negócio. Sendo este benefício tangível, intangível ou ambos.

2.1 Ciclo de Vida de Projeto

Todo projeto de software tem uma estrutura básica chamada de ciclo de vida. O ciclo de vida é a série de fases pelas quais um projeto precisa passar para alcançar os seus objetivos (PMBOK, 2017).

2.1.1 Fases do Ciclo de Vida

As fases do ciclo de vida de um projeto são as etapas pelo qual do projeto deve passar até que atinja o seu objetivo final. As fases do ciclo de vida podem ser sequenciais, iterativas ou sobrepostas. Dentre as fases, existem no mínimo uma ou mais ligadas ao desenvolvimento do produto, chamada ciclo de vida do desenvolvimento. Os ciclos de vida do desenvolvimento podem ser preditivos, iterativos, incrementais, adaptativos ou híbridos.

Em um ciclo preditivo, todos os elementos do projeto são definidos nas fases iniciais, e quaisquer alterações no escopo do projeto são cuidadosamente gerenciadas.

No ciclo iterativo o escopo do projeto é determinado no início, porém as estimativas de prazos, esforços e custos são modificadas constantemente durante as iterações dos projetos. Dentro do ciclo iterativo o produto é desenvolvido por meio de uma série de ciclos repetidos, diferente do incremental, onde se é adicionado elementos sucessivamente à funcionalidade do produto até que ele esteja pronto.

Continuando no já citado, o ciclo de vida incremental utiliza-se de uma série de iterações que

sucessivamente adicionam funcionalidades ao produto dentro de um determinado prazo, a entrega só poderá ser considerada completa após a sua iteração final, ou seja, quando o produto estiver de acordo com todos os requisitos do projeto.

Já os ciclos de vida adaptativos, cognominado também de ciclo de vida ágil ou orientado a mudanças, são ágeis e iterativos ou incrementais, onde um escopo detalhado é definido e aprovado sempre antes de uma iteração.

E ainda, temos o ciclo de vida híbrido, ele é uma combinação do ciclo de vida adaptativo e preditivo.

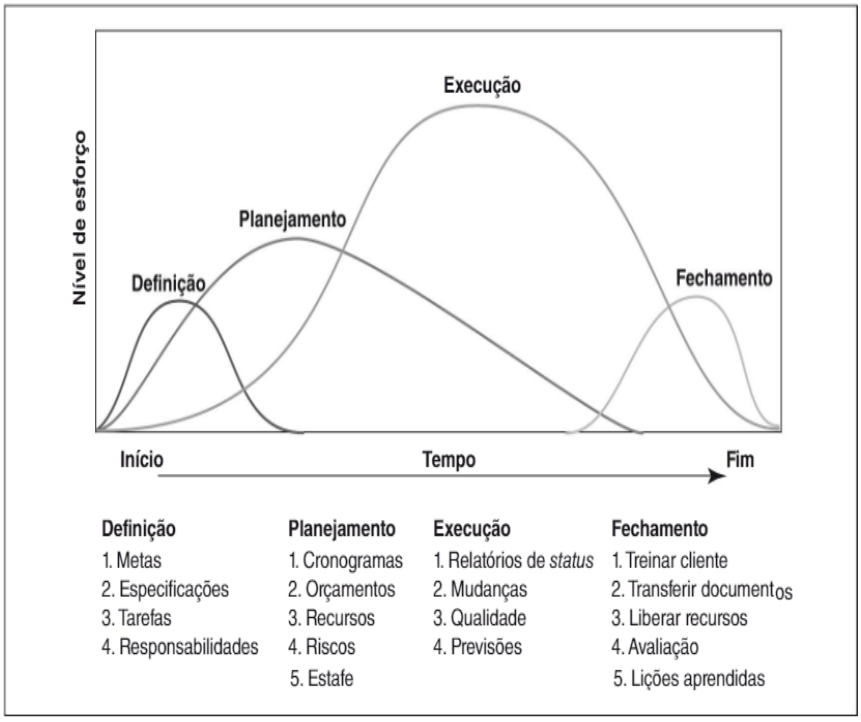
É importante frisar que os ciclos de vida do projeto são independentes dos ciclos de vida do produto resultante do projeto. O ciclo de vida do produto engloba, não só a fase de projeto e desenvolvimento, como também o crescimento, maturidade e descontinuação do produto (PMBOK, 2017).

Entender o ciclo de vida de um projeto é de extrema importância, alguns autores acreditam que o ciclo de vida de um projeto e a base para gerenciá-lo (LARSON & GRAY, 2016). Dentre a literatura de gerência de projetos existem inúmeros modelos de ciclo de vida, muitos deles são específicos para determinados tipos de projetos. Como exemplo, Larson e Gray (2016) apresentam um ciclo genérico que usualmente são formados por quatro estágios sequenciais: definição, planejamento, execução e entrega.

No estágio de definição são estabelecidas as especificações, objetivos, as equipes e suas principais responsabilidades. No estágio de planejamento é fechado o escopo do projeto, os beneficiários, o nível de qualidade e o orçamento destinado ao projeto. No estágio de

execução é onde o produto físico é produzido, esta é a etapa que mais exige esforço dos envolvidos, principalmente de quem está gerenciando o projeto, pois ele deve estar atento aos prazos, custo, especificações, revisões e possíveis mudanças que venham a surgir. Por fim o estágio da entrega, onde além de se entregar o produto desenvolvido, muitas vezes, é requerido o repasse de documentação e treinamento dos clientes. A fase de entrega também é onde se deve absorver a lições apreendidas durante todo o ciclo de vida, para se planejar projetos mais assertivos, evitando os erros cometidos anteriormente e evidenciando os acertos.

Logo abaixo, na Figura 1, segue um gráfico demonstrando o nível de esforço necessário em cada estágio durante todo o ciclo de vida de um projeto.



Fonte: Larson e Gray, 2016.

2.1.2 Exemplos de Ciclos de Vida

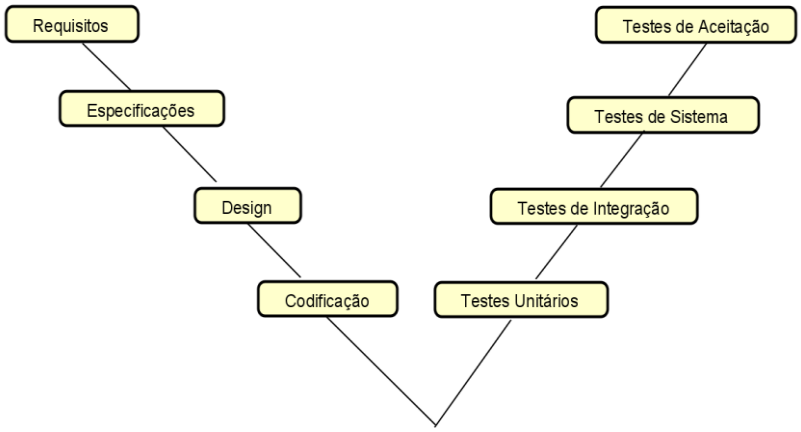
Nas sessões seguintes serão descritos três exemplos de ciclos de vida, o *Waterfall*, o *Spiral* o *Rational Unified Process*.

2.1.2.1Ciclo *Waterfall*

O modelo *Waterfall* começa com o levantamento e organização dos requisitos, seguido de especificações do sistema, definição dos requisitos funcionais e não-funcionais, design e implementação do software, testes unitários, testes de integração, testes do sistema e testes de aceitação. Cada fase do modelo só deve começar quando a fase anterior estiver concluída, logo, este modelo é utilizado em projeto onde os requisitos já são conhecidos pela equipe na fase inicial e o projeto não prever uma abertura para mudanças futuras (O'REGAN, 2017).

Abaixo segue uma Figura 2 ilustrando as fases do ciclo de vida *Waterfall* em formato de V.

Figura 2 - Modelo de Ciclo de Vida *Waterfall* em V



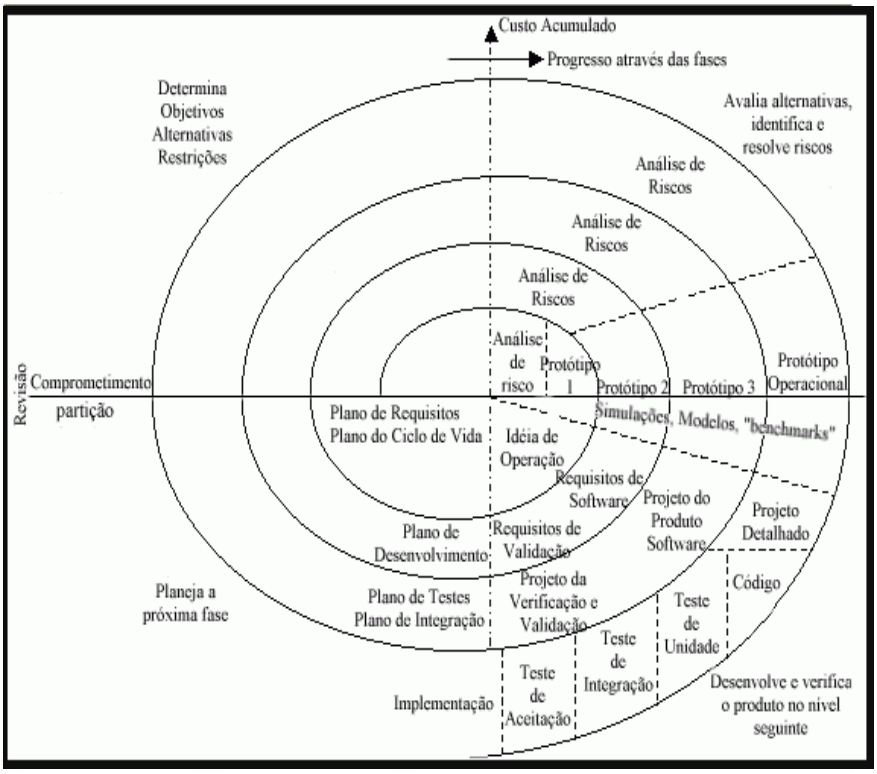
Fonte: Autoria Própria

2.1.2.2 Ciclo *Spiral*

Proposto na década de 1980 por Barry Boehm, o modelo *Spiral* é voltado para projetos onde a equipe não tem condições de definir todos os requisitos no início, projetos com requisitos que podem sofrer mudanças durante sua execução e projeto que tem requisitos que dependem de partes que serão colocadas durante as fases do projeto. O desenvolvimento decorre em um processo espiral onde cada volta envolve a definição dos objetivos, identificação e análises de risco, design, desenvolvimento e testes e o planejamento da próxima iteração levando em conta os resultados das anteriores (O'REGAN, 2017). O ponto forte desta abordagem em relação à anterior é a possibilidade da opinião do cliente sempre presente em cada volta na espiral, podendo eliminar a ocorrência de erros e ajustar os requisitos para que a versão final do software esteja em conformidade com as expectativas do cliente. Vale ressaltar que muitas das metodologias ágeis utilizam como base o modelo *Spiral* em seus processos.

Abaixo segue uma figura ilustrando as fases do ciclo de vida espiral:

Figura 3 - Modelo de Ciclo de Vida Spiral



Fonte: O'Regan , 2017.

2.1.2.3 Ciclo *Rational Unified Process*

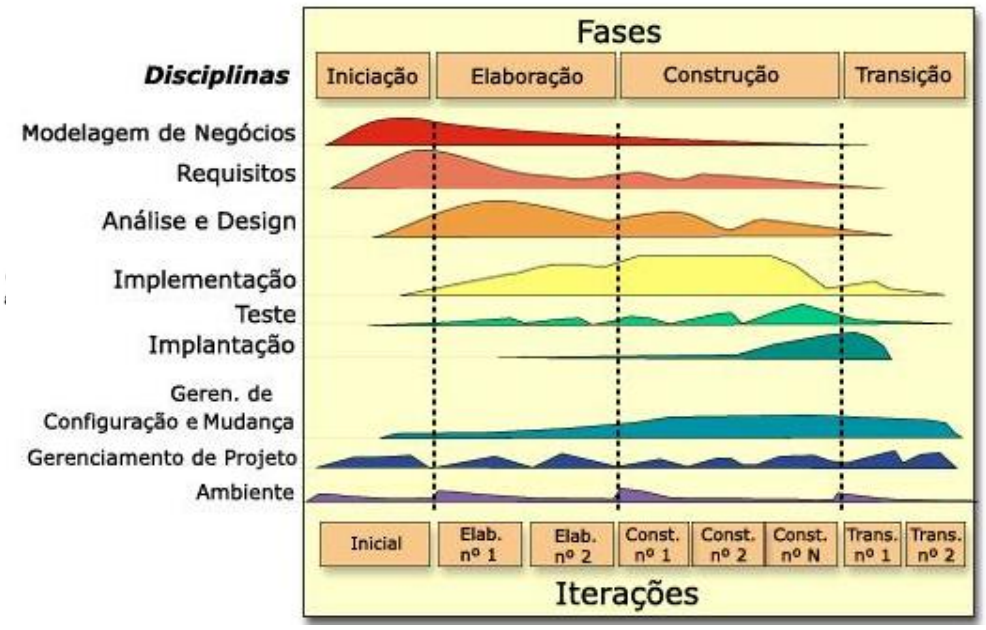
Por fim, o modelo *Rational Unified Process* (RUP), desenvolvido no final dos anos de 1990 dentro da *Rational Corporation*. Ele utiliza o, bastante conhecido pela engenharia de software, *Unified Modelling Language* (UML) como ferramenta de especificação e *design* (O'REGAN, 2017). O modelo é baseado em três perspectivas, a perspectiva dinâmica que vai apresentando as fases ao decorrer do tempo, a perspectiva estática, que apresenta as atividades

realizadas durante o processo, e a perspectiva prática, que faz a sugestão de boas práticas a serem utilizadas no processo. O que difere dos modelos anteriores, onde as fases dos processos são equalizadas com as atividades do processo, o modelo RUP tem suas fases diretamente ligadas ao negócio, tais fases são: concepção, elaboração, construção e transição.

O objetivo da fase de concepção é levantar todas as entidades que irão interagir com o sistema e definir os tipos de interações, ao fim desta fase, deve se analisar as informações e avaliar a contribuição do sistema para o negócio a fim de abortar o projeto caso o resultado não seja satisfatório. Na fase de elaboração, deve-se desenvolver uma compreensão a respeito do projeto, definir *frameworks* para arquitetura do sistema, desenvolver o plano de projeto e mapear os maiores riscos. Na fase de construção é onde entra o projeto, programação e testes. Ao fim da fase de construção deve se ter o sistema de software pronto, assim como a documentação associada. E por último, a fase de transição, que envolve a entrega do software ao usuário final em um ambiente real (SOMMERVILLE, 2011).

Abaixo segue uma figura mostrando as fases do ciclo de vida RUP, onde ondas são demostradas ilustradas para representar o nível de esforço dedicado em casa fase do ciclo.

Figura 1 - Modelo de ciclo de vida RUP



Fonte: RUP, 1998

2.2 *Framework Scrum*

À medida que a importância da gestão de projetos tornou-se uma prática comum, as empresas perceberam a necessidade de se desenvolver metodologias para lidar com isso. Sem uma metodologia de gestão de projetos, pode ser muito difícil alcançar a excelência, em gestão de projetos, a excelência é compreendida como um fluxo contínuo de projetos gerenciados com êxito. Vale ressaltar que a escolha de uma boa metodologia pode aumentar o desempenho durante a execução do projeto e

ampliar o nível de confiança com o cliente (KERZNER, 2020). Segundo Junior (2017) as metodologias de gestão de projetos são fundamentais para as empresas organizarem seus processos sistematizando melhor seus objetivos e estratégia. Junior (2017) também afirma que a utilização de uma boa metodologia irá garantir a execução correta das atividades inerentes ao projeto.

Segundo o guia PMBOK (2017), uma metodologia é definida como sendo um sistema de práticas, técnicas, procedimentos e regras usadas por aqueles que trabalham no projeto. Já Kerzner (2020) fala sobre metodologias como um conjunto de formulários, diretrizes, *templates* e lista de verificações que podem ser aplicadas a um projeto ou situação específica.

Dentre as inúmeras metodologias que podem ser utilizadas na gestão de projetos existem algumas, como *Scrum*, XP e *Kanban* que vem se destacando pelo fato de serem cada vez mais utilizadas em empresas de sucesso, e algo que elas tem em comum é o fato de serem ágeis (PWC, 2012). Com o mercado de software operando em cenários globalizados, tendo que lidar constantemente com mudanças e com o aumento da velocidade de entrega de projetos, tornou-se inevitável a adoção desse tipo de metodologia para gestão de seus projetos (SOMMERVILLE, 2011).

E foi pensando em desenvolver softwares sempre em velocidade maior e ainda mantendo a qualidade e a satisfação dos clientes, que no início de 2001, Kent Beck e outros 16 notórios da engenharia de software criaram o “Manifesto para desenvolvimento de software ágil”. O manifesto que em suma foi munido dos seguintes valores:

> Indivíduos e interações mais que processos e ferramentas.
>
> Software em funcionamento mais que documentação abrangente.
>
> Colaboração com o cliente mais que negociação de contratos.
>
> Responder a mudanças mais que seguir um plano. (BECK et al., 2001)

Algo importante que Beck e seus pares deixam claro no manifesto, é que os valores da direita também são importantes e devem ser levados em conta no processo, no entanto os da esquerda devem ser priorizados quando em choque com as da direita. Além dos quatro valores, o manifesto também nos apresenta aos doze princípios que devem ser seguidos no desenvolvimento ágil.

Nem todos os modelos ágeis façam uso de todos estes métodos, tais princípios definem um pensamento ágil.

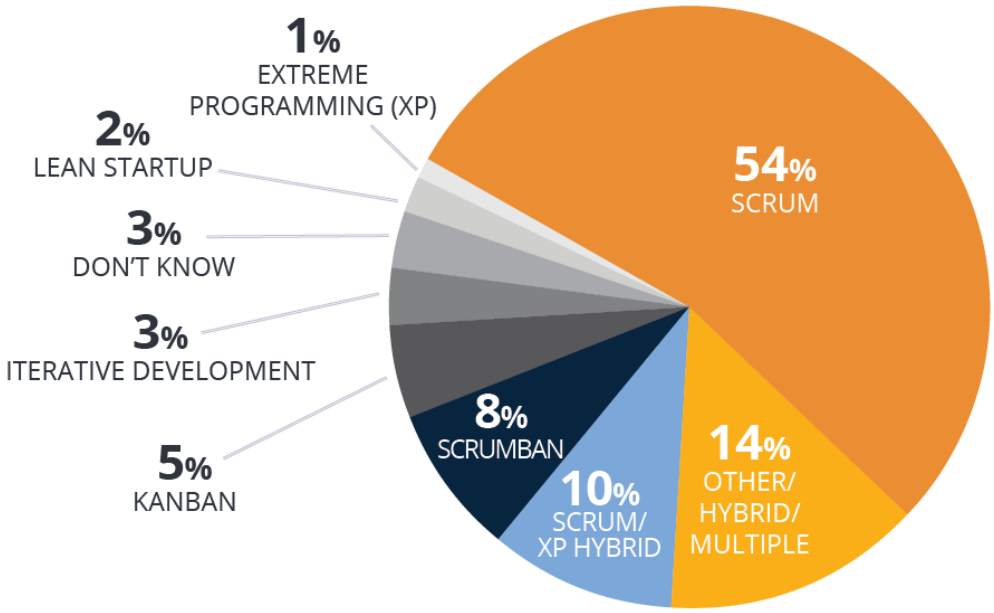
- Satisfação do cliente com entrega continua e adiantada de software com valor agregado.
- Mudanças contínuas nos requisitos em qualquer estágio do projeto, desde que sejam entendidas como algo que possa trazer vantagens ao cliente.
- Entrega frequente de software funcional em curtos espaços de tempo.
- Pessoas de negócio e desenvolvedores trabalhando juntos diariamente durante todo o projeto.
- Disponibilização de ambiente e o suporte necessário para aos indivíduos envolvidos no projeto.
- Priorização da comunicação direta como meio de transmissão das informações.

- Software funcionado como medida primaria do projeto.
- Capacidade de manter um ritmo constante indefinidamente entre os envolvidos no projeto.
- Excelência técnica e bom design em prol da agilidade.
- Simplicidade.
- Equipes auto-organizáveis.
- Refinamento e ajustes de processos e comportamentos em intervalos regulares.

Dentre as metodologias que carregam a mentalidade ágil podemos destacar a mais bem sucedida delas, o *Scrum* (PWC, 2012).

Segundo uma pesquisa de 2012 da *Price waterhouse Coopers* (PWC), o *Scrum* lidera o ranking de metodologia mais utilizada na organização, sendo escolhida por 45% dos entrevistados como a metodologia principal para o desenvolvimento e gestão de seus projetos, seguida de *Lean* e *Extreme Programming* (XP), com aproximadamente 10% cada, já de acordo com o *13th Annual State Of Aglie Report* publicado pela *CollabNet VersionOne* em 2019, o Scrum mantém a liderança com 54% de aderência entre os entrevistados, seguido por 14% que dizem utilizarem uma versão hibrida de múltiplas metodologias diferentes, e 10% que utilizam uma versão hibrida entre a metodologia XP e o Scrum, conforme ilustrado na Figura 5.

Figura 2 - Metodologias Ágeis



Fonte: COLLABNET VERSIONONE, 2019.

O *Scrum* foi elaborado por Jeff Sutherland em 1993, juntamente com Mike Beedle e Ken Schwaber, todos signatários do manifesto ágil, baseado em um artigo publicado por Takeuchi e Nonaka (1986) sobre as vantagens dos pequenos times no desenvolvimento de produtos voltado a indústria automotiva japonesa.

O *Scrum* tem como princípio para gerenciamento de projetos, práticas interativas e incrementais buscando sempre agregar valor ao negócio do cliente. O *Scrum* também é visto como uma abordagem empírica que torna possível o gerenciamento de projetos flexíveis e adaptáveis, tudo isso alinhado ao desempenho em mudanças constantes, que são muito presentes na engenharia de software atual (SILVA & LOVATO, 2016).

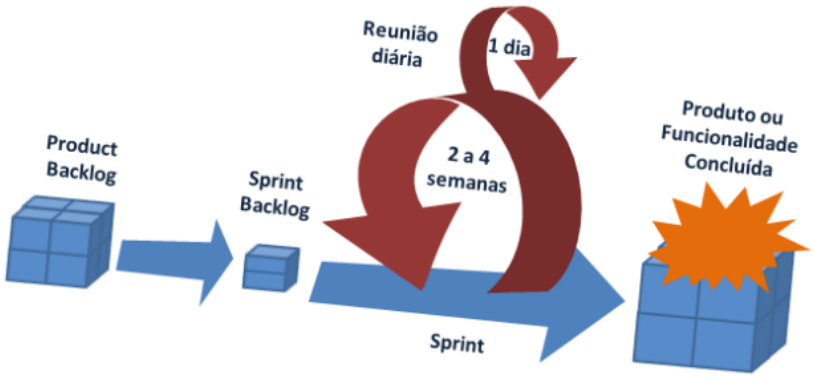
O *Scrum* não é um processo, nem uma técnica para construção de produtos, mas sim um *framework* estrutural voltado para o desenvolvimento de produtos complexos, dentro do qual emprega vários processos e técnicas para alcançar os seus objetivos (SCHWABER & SUTHERLAND, 2013).

Segundo Silva e Lovato (2016) o *Scrum* não é uma metodologia, pelo fato dele não dizer o que fazer, e sim, um *framework* voltado a soluções de problemas de software. Já para Terlizzi & Biancolino (2014) o *Scrum* não só é uma metodologia como também um *framework* estrutural onde se é possível empregar processos e técnicas para gerência de projetos.

No *Scrum* os projetos devem ter um princípio, meio e fim, também deve ser sempre focado em gerar valor através de entregas contínuas e interativas de *softwares* utilizáveis. O *framework* consiste em times ligados a papéis, eventos e artefatos. Cada componente desse aliado a um ciclo de vida iterativo e incremental é essencial para o sucesso do Scrum (SCHWABER & SUTHERLAND, 2013).

A Figura 6 representa de forma objetiva o ciclo de vida de um projeto gerenciado com *Scrum*. Na figura é possível visualizar os artefatos e alguns dos eventos pertencentes ao *framework*.

Figura 3 - Fluxo Scrum



Fonte: DESSOLDI, 2019.

Algo essencial para que o fluxo representado acima seja eficiente é ter um bom time Scrum.

2.3 Papeis *Scrum*

Um time Scrum é composto por indivíduos com os seguintes papéis *Product Owner*, *Development Team* e *Scrum Master*: o *Product Owner* tem a responsabilidade gerir o *Product backlog*, maximizar o valor do produto e do time de desenvolvimento. O *Development Team*, e os responsáveis diretos pelo desenvolvimento e entrega do produto em versões utilizáveis a cada ciclo de *Sprint*. E o *Scrum Master*, que tem a responsabilidade de garantir que o Scrum seja entendido e aplicado. O *Scrum Master* também tem o dever de ajudar aqueles que estão fora do time a entender quais de suas ações são úteis, ou não, para o time, caso entendidas como não úteis, o *Scrum Master* deve filtrá-las a fim de maximizar o valor criado pelo time *Scrum* (SCHWABER & SUTHERLAND, 2013).

2.3.1 Eventos *Scrum*

Os eventos do *Scrum*, ou *Time-boxes,* como também é conhecido, são eventos com um tempo limite definidos, podendo ser encerrados antes caso ele já tenha atingido seu objetivo. É de responsabilidade de o Scrum Master garantir que os eventos ocorram durante o projeto (DUARTE, 2019).

O principal evento do *Scrum* é o *Sprint*, o *Sprint* é um período limitado onde um incremento utilizável do produto deve ser entregue pela equipe, além do desenvolvimento do produto todos os outros eventos ocorrem durante o período do *Sprint*. Os períodos do *sprint* devem sempre ser fixos e constantes, alguns autores, entre eles Ken Schwaber e Jeff Sutherland (2013), citam que a duração do *sprint* deve ser de no máximo um mês, pois um período muito longo de *Sprint* pode acabar comprometendo a qualidade do processo.

Além do *Sprint*, temos como eventos do *Scrum* o *Sprint Planning*, onde em até 8 horas de reunião é apresentado as necessidades do produto pelo *Product Owner* e com base nisso e na capacidade produtiva da equipe é planejada e deliberada todas as ações do Sprint. *Daily Scrum*, uma reunião diária de no máximo 15 minutos com o objetivo de alinhar o status das atividades realizadas entre a equipe. *Sprint Review*, com duração máxima de 4 horas, é onde é apresentado o incremento utilizável do produto para apreciação do *Product Owner* e os demos *stakeholders*. E o *Sprint Retrospective*, com duração máxima de 3 horas, tem o objetivo de discutir os resultados obtidos no *Sprint* que passou e o que pode ser melhorado no processo com a lição aprendidas.

2.3.2 Artefatos Scrum

Dentre os artefatos do *Scrum* temos o *Product Backlog*, que é uma lista ordenada de tarefas que são necessárias para o uso do produto. Essa lista serve de fonte única para os requisitos do sistema em qualquer etapa do projeto e deve ser gerenciado pela *Product Owner. Sprint Backlog*, uma versão reduzida do *Product Backlog*, que engloba apenas a atividades do *Sprint*. E o Incremento que é a junção de todos os itens do *Sprint Backlog* entregues durante um *Sprint*, somado ao valor dos incrementos de todos os *Sprints* anteriores. Por fim os Cada incremento deve resultar em uma parte utilizável do produto para ser avaliado pelo *Product Owner*, que decidirá se ele será liberado em produção ou não (SCHWABER & SUTHERLAND, 2013).

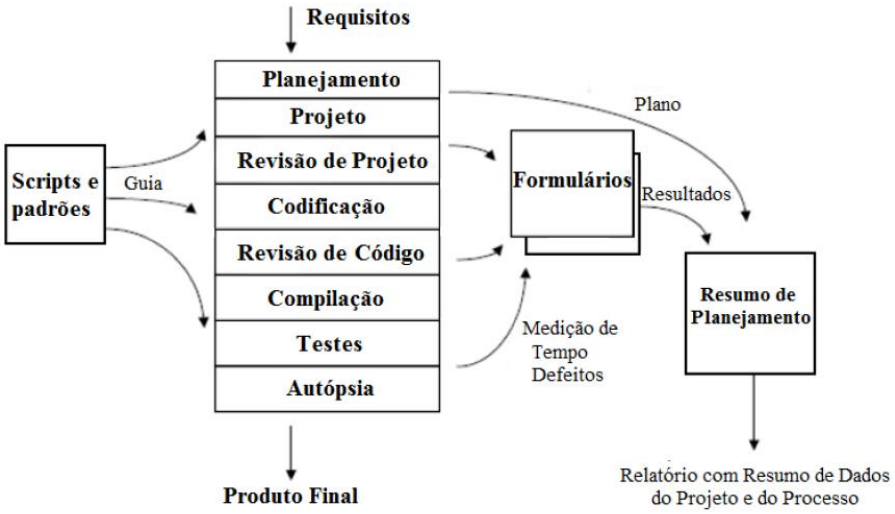
2.4 Personal Software Process

Personal Software Process (PSP) é um método que busca, através de um processo pessoal, a melhoria na qualidade do trabalho de um engenheiro de Software. Segundo Watts Humphrey (2015), PSP é um processo de melhoria pessoal projetado para ajudar o engenheiro de software a controlar, administrar e melhorar o modo como ele lida com seus projetos de softwares. O PSP é uma estrutura de formulários, diretrizes e procedimentos para o desenvolvimento de software, que se usado de forma correta, provê os dados históricos que o desenvolvedor de software precisa para executar e conhecer melhor os seus compromissos tornando os elementos rotineiros de seu

trabalho mais previsíveis e eficientes. O PSP baseia-se no princípio do conhecimento, avaliação e melhorias contínuas do processo individual de desenvolvimento de software, com foco no perfil de erros cometidos individualmente com mais frequência e em sua minimização (HUMPHREY, 2005).

No PSP o processo de desenvolvimento de software é dividido em fases, durante as quais devem ser aplicados seus componentes. A Figura 7 expõe o fluxo PSP com seus componentes ao longo de suas fases.

Figura 4 - Fluxo PSP



Fonte: (ESTECA et al., 2015)

Assim como indicado na figura 7, durante as fases do PSP todo o processo deve ser guiado pelos scripts e padrões, usando-se de formulários para registrar os resultados de medições de tempo e defeitos encontrados no processo (ESTECA et al., 2015).

Nas próximas subseções será apresentado de forma mais breve as fases do PSP, seus componentes e seus níveis.

2.4.1 Fases do PSP

Com o objetivo de estabelecer um processo bem definido de qualidade de software, o PSP se divide em três fases:

- Planejamento: onde será definido o que fazer e com fazer através do desenvolvimento de um plano que guiará todo o processo de desenvolvimento do software;
- Desenvolvimento: a fase de desenvolvimento é composta pelas seguintes etapas: definição de requisitos, projeto do software, revisão do projeto, codificação, revisão do código, compilação e testes;
- *Postmortem*: documenta e estabelece um comparativo entre o que foi planejado e o que foi seguido durante o projeto.

2.4.2 Os Componentes do PSP

O PSP é composto por quatro componentes, *scripts* ou roteiros, formulários, medições e padrões.

Os *scripts* ou roteiros são descrições compostas pelo objetivo do processo, critérios de entradas, diretrizes gerais, passos a serem executados, critérios de qualidade e condições para encerramento. Estes *scripts* têm o objetivo de direcionar o desenvolvedor na realização de um processo específico.

Os formulários definem os dados que devem ser coletados em cada nível do PSP.

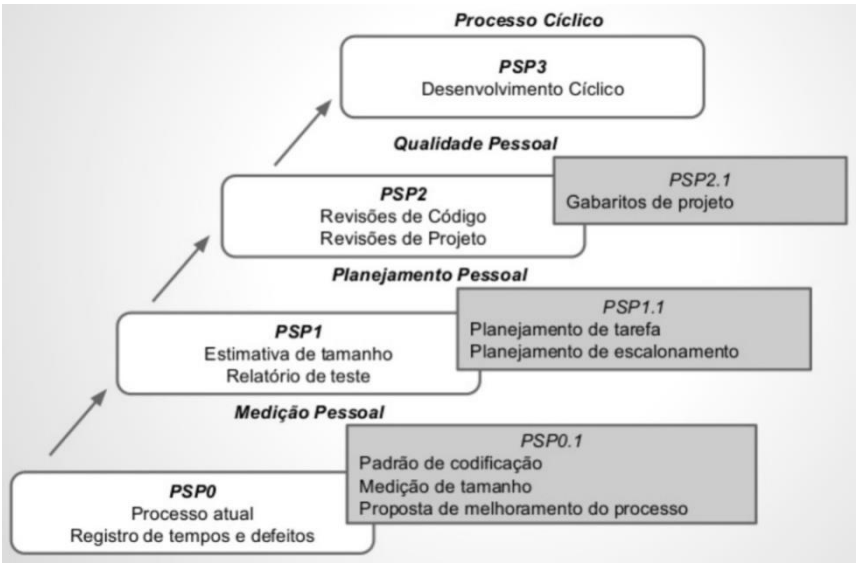
As medições quantificam os processos e produtos com dados analíticos.

E os padrões oferecem definições precisas e imutáveis que servirão de guia para o trabalho durante o projeto.

2.4.3 Os Níveis PSP

O PSP é composto por sete níveis de competências evolucionárias, e sua implantação é feita de forma incremental. Na Figura 8 é ilustrada a estrutura do PSP, sumarizando os elementos incorporados a cada nível.

Figura 5 - Níveis do PSP



Fonte: Humphrey, 2005.

O PSP é composto por sete níveis de competências evolucionários e sua implantação é feita de forma incremental. Na Figura 8 é ilustrada a estrutura do

PSP, sumarizando os elementos incorporados a cada nível.

- PSP0: o processo base, neste nível é introduzido as primeiras medições e relatórios padronizados a rotina do desenvolvedor. Este estágio provê uma base para medições e obtenções de dados de seu trabalho;
- PSP0.1: do PSP0 incrementa-se para o PSP0.1 adicionando um padrão de codificação, medições de tamanho e proposta de desenvolvimento de processos através de um formulário que provê uma forma estruturada para registro de problemas no processo, experiências positivas e negativas e sugestões de melhorias;
- PSP1: E o processo de planejamento pessoal, neste nível é adicionado passos de planejamento ao nível anterior, um relatório de testes e estimativa de recursos e tamanho;
- PSP1.1: são introduzidos planejamento de horários e tarefas através da elaboração de cronogramas de projeto e emprego de técnicas para monitoramento de progresso e detecção de atrasos em relação ao planejado.
- PSP2: o nível com o foco em gerenciamento de qualidade pessoal. No PSP2 é acrescentado técnicas de revisão ao PSP1.
- PSP2.1: é estabelecido critérios de verificação de finalização de projeto e examina várias técnicas para verificação de consistência de projetos.
- PSP3: neste nível a abordagem procura subdividir o programa em partes menores chamadas de módulos. A construção será feita com base em ciclos de iteração e a cada repetição será executado um PSP2 completo incluindo projeto, compilação e teses.

3. RESULTADOS

Nas sessões seguintes será apresentado o *framework Scrum* adaptado para o desenvolvimento *standalone*. E dentro dos eventos e artefatos será descrito os elementos do PSP que podem ser aplicados para apoiar e melhorar a adaptação do *Scrum*, gerando uma metodologia pessoal híbrida entre *Scrum* e PSP. Sendo assim, iremos chamar a metodologia apresentada a seguir de P-Scrum.

3.1 Papéis

Quanto aos papeis do *P-Scrum*, não faz sentido manter as figuras do *Product Owner*, *Development Team* e *Scrum Master* presentes no *Scrum*, tendo em vista, que eles foram concebidos levando em consideração que a equipe seria composta por vários membros. Os papeis considerados nessa adaptação serão os de desenvolvedor e cliente.

3.1.1 Desenvolvedor

O desenvolvedor será responsável por desenvolver e gerenciar o produto de *software* ao longo do projeto e garantir que os eventos e artefatos destacados nas sessões 3.2 e 3.3 sejam respeitados durante o projeto, ou seja, absorverá os papeis do *Scrum*. O desenvolvedor também deverá garantir que seja feito um planejamento de seu trabalho, garantir que haja interações com o cliente, obtenção de *feedbacks* contínuos e que sejam registrados os erros e acertos no

processo de desenvolvimento a fim de refiná-lo a cada iteração.

3.1.2 Cliente

O cliente deve acompanhar os P-Sprints reviews, validar os incrementos do produto, realizar testes de usuário, dar opinião, fazer sugestões de alteração no *Product Backlog* e garantir que o desenvolvedor tenha ciência do valor que cada incremento esteja gerando ao produto. Por fim, vale ressaltar que cabe ao desenvolvedor deixar claro o papel do cliente no início do projeto

3.2 Eventos

Nas sessões seguintes iremos apresentar os eventos *P-Scrum* que serão utilizados para o *framework* proposto neste trabalho: *P-Sprint, P-Sprint Planning, P-Sprint Review* e *P-Sprint Retrospective*.

3.2.1 *P-Sprint*

O *P-Sprint* segue as mesmas premissas do *Sprint* padrão do Scrum, ou seja, é um ciclo de desenvolvimento onde um incremento do produto pronto é gerado pelo desenvolvedor a partir dos itens selecionados para as *P-Sprint* no *product backlog*. Além do desenvolvimento do produto em si, todos os outros eventos, com exceção do *P-Sprint Planning* inicial, são realizados dentro do *timebox* do P-*Sprint*. Conforme o padrão no Scrum, a duração das P-*Sprints* são em *timebox* fixas e constantes.

Durante a P-*Sprint*, com o objetivo de aplicar as primeiras medições advindas do PSP0, deverão ser adotado os formulários de Log de Registro de Tempo (ANEXO A), Log de Registro de Defeito (ANEXO B) e Tipo de Defeito Padrão (ANEXO C).Eles auxiliarão no registro e medições dos P-*Sprints* com o objetivo de avaliaras práticas seguidas em *P-Sprints* anteriores, e se necessário, reformular as posteriores.

Outro formulário do PSP0 que pode ser adotados na *P-Sprint* é o PSP0 - Script de desenvolvimento (ANEXO D), pois ele apoiará o desenvolvedor no processo de desenvolvimento trazendo quatros passos padrões a serem seguidos, e critérios de entradas e saídas que nortearam o desenvolvedor em seu processo de desenvolvimento.

Por fim, para garantir a qualidade do código produzido no *P-Sprint,* deve ser adotado os conceitos de revisão de código do PSP2, com a aplicação de revisões como inspeções, walk-throughs e revisões pessoais. De auxílio prático, o PSP fornece o PSP - Script de Revisão de Código (ANEXOI) e PSP - Checklist de Revisão de Código (ANEXOJ).

3.2.2 *P-Sprint Planning*

Todo trabalho que será realizado nos *P-Sprint* será planejados nos *P-Sprint Planning*, porém diferente do *Scrum*, ao invés de ser uma reunião entre a equipe, será um momento no projeto onde o desenvolvedor deve planejar e definir quais atividades do *P-product backlog* devem ser incorporadas ao Sprint, e o que deve ser entregue no próximo incremento.

Para auxiliar no planejamento do *Sprint*, deverão ser adotados os scripts PSP0 – Script de Planejamento (ANEXO E). Com esse script, o desenvolvedor terá uma visão mais clara do que será desenvolvido, tal como uma formalização do tempo estimado na atividade.

Para que o *P-Sprint Planning* seja efetivo, as estimativas de esforços por tarefas no *backlog* devem estar o mais assertivas possíveis. Com isso um artefato importante do PSP0 a ser utilizado é o PSP0 - Resumo de Plano de Projeto (PSP0-RPP) (ANEXO F). O PSP0-RPP dará ao desenvolvedor uma referência de estimativa baseada em experiências passadas levando em consideração a incidência de erros e correção deles. Vale ressaltar que o PSP0-RPP deverá ser alimentado e refinado durante todas as fases do projeto.

Outro relatório importante a ser consultado nesta fase é o de Log de Registro de Tempo (ANEXOA), relatório introduzido no P-Sprint.

No PSP0. 1 também é nos fornecido premissas importantes que auxiliarão no planejamento do P-Sprint, tais como alguns dos passos estabelecidos por Humphrey (2005) para planejamento de processos de software:

- Começar uma declaração explicita do trabalho que será feito e assegurar que é o que o cliente espera.
- Quebrar atividades que levem mais que um dia para serem feitas e dividi-las em múltiplas tarefas estimando-as separadamente.
- Realizar as estimativas baseada em dados históricos de atividades similares.
- Documentar as estimativas e depois compará-las com os resultados atuais.

Por fim, o PSP1 introduz a utilização do método *Proxy-Based Estimating* (PROBE), método estatístico

avançado para estimar tamanho e tempo de desenvolvimento (HUMPHREY, 2005). Ele pode ser adotado como método que guiará o desenvolvedor em suas estimativas, no entanto o desenvolvedor pode adotar o método de estimativa que mais o convém, tendo em vista que o método sugerido pelo PSP requer um nível elevado de conhecimento matemático, o que pode não ser a realizada de todos os desenvolvedores que farão uso desse *framework*.

3.2.3 *P-Sprint* Review

O *P-Sprint Review* é executado no final de cada *P-Sprint* para inspecionar o incremento desenvolvido. A cerimônia deverá ter a presença do cliente e será feita com o objetivo de apresentar o incremento, receber *feedbacks* do produto, refinar o *Backlog* e analisar a linha do tempo do projeto relacionando o que foi entregue e ao que será entregue no próximo ciclo de *P-Sprint*.

Com os elementos PSP inseridos nos demais eventos o desenvolvedor terá uma base documentada mais solida para discutir sobre o incremento entregue e refinar o *Product Backlog* de acordo com o *feedback* do cliente.

3.2.4 *P-Sprint Retrospective*

Assim como o Sprint Retrospective do framework Scrum, o *P-Sprint Retrospective* dará a oportunidade do desenvolvedor de inspecionar a si próprio e criar um plano para melhorias a serem aplicadas na próxima P-Sprint. Este evento deve ocorrer após o Sprint Review e deverá

ter como objetivo o refinamento do processo de desenvolvimento na próxima P-Sprint.

O primeiro elemento PSP que deve ser adotado nesta fase é o PSP0 - Script de Postmortem (ANEXOH), ele trará visibilidade dos defeitos encontrados no desenvolvimento durante o P-Sprint.

Outro elemento que deve ser utilizado são os conceitos de revisão de projetos, as revisões devem ser usadas em qualquer elemento do produto, desde histórias de usuário, processos e o projeto como um todo. Pode ser adotado PSP2 - Script de Revisão de Projeto (ANEXO K) e o PSP2 - Checklist de Revisão de Projeto (ANEXO L). Estes artefatos construíram uma base solida para análise e criação de um plano de melhoria para os próximos ciclos de desenvolvimento e garantirão a qualidade do projeto.

3.3 Artefatos

Os artefatos do Scrum são de fundamental importância para a metodologia proposta, pois eles são a representação do trabalho ou do valor obtido no projeto. Eles possibilitam a transparência e oferece oportunidades para inspeção e adaptações durante o projeto. Quanto aos artefatos do *P-Scrum*, não se vê a necessidade de adaptá-los, visto que, os artefatos propostos por Schwaber e Sutherland (2013) no Scrum são plenamente aplicáveis a metodologia proposta neste trabalho, com isso apenas alteramos seus nomes para seguirmos o padrão estabelecido anteriormente, onde *P-Product Backlog*, *P-Sprint Backlog* e *P-Increment* seguem os padrões de *Product Backlog, Sprint* e *Increment* respectivamente.

4. CONSIDERAÇÕES FINAIS

Conforme comprovado na literatura, para projetos de software manter a qualidade e obterem sucesso, eles devem ser bem gerenciados (PMBOK, 2017). E para o bom gerenciamento de projeto de software é imprescindível que seja adotado uma boa metodologia (Junior, 2017).

Com o intuito de alcançar os objetos específicos desse trabalho, foi feito um estudo aprofundado dos principais conceitos de gerência de projeto, *Scrum* e *Personal Software Process.*

Posteriormente foram sugeridas, dentro dos processos Scrum, adaptações que podem adequar o framework a realidade de desenvolvedores standalone, e adoções de elementos PSP, que conforme apresentado, são de fundamental importância para garantir a qualidade de processos pessoais de desenvolvimento de softwares. Estas adaptações resultaram na proposta do P-Scrum, um *framework* para gerência de projetos de software híbrido entre *Scrum* e PSP aplicável em projetos de desenvolvedores *standalone.*

Recomenda-se como trabalhos futuros a elaboração de um estudo de caso com a aplicação do P-Scrum em projetos de softwares reais. A execução de uma análise comparativa entre projetos de softwares desenvolvidos com P-Scrum e projetos de softwares desenvolvidos com outras metodologias ou *frameworks* de gerência de projeto. E a reestruturação do framework levando em consideração os níveis de maturidade do PSP, ou seja, implementar o framework para que ele evolua junto ao desenvolvedor a medida em que se é

alcançado um maior nível de maturidade dentro dos processos PSP.

REFERÊNCIAS

BECK, K. et al. **Manifesto ágil**. Manifesto para Desenvolvimento Ágil de Software, 2001. Disponível em: <https://agilemanifesto.org/iso/ptbr/manifesto.html>. Acesso em: 06 mai. 2020.

COLLABNET VERSIONONE. 13th Annual State Of Agile Report, mai. de 2019. Disponível em: <https://www.stateofagile.com/#ufh-i-521251909-13th-annual-state-of-agile-report/473508>. Acesso em: 07 mai. 2020.

DESSOLDI, Flávia. **Método SCRUM — Um resumo de tudo o que você precisa saber**. Medium. 03jun. 2019. Disponível em: <https://medium.com/reprogramabr/scrum-um-breve-resumo-f051e1bc06d9>.Acesso em: 27 mai. 2020.

DUARTE, Luiz. **Scrum e Métodos Ágeis: Um Guia Prático**: 1. ed. Gravataí, Rio Grande do Sul: ISBN, 2019.

ESTECA, Antonio Marcos Neves. et al. **Personal Software Process: Uma visão geral sobre o processo e o seu impacto na indústria de software**. Perspectivas em Ciências Tecnológicas, v. 4, n. 4, p.34-45, mai. 2015.

GERHARDT, Tatiana Engel; SILVEIRA, Denise Tolfo. **Métodos de Pesquisa**: 1. ed. Rio Grande do Sul: UFRGS, 2009.

HUMPHREY, Watts S**. PSP:** a self-improvement process for software engineers. Massachusetts: Pearson Education, Inc., 2005.

JUNIOR, Carlos. **Gestão de projetos:** 12 principais metodologias: Framework Ágil para projetos complexos. Project Builder. 31 mai. 2017 Disponível em: < https://www.projectbuilder.com.br/blog/metodologias-de-gestao-de-projetos/>. Acesso em: 25 abr. 2020.

JÚNIOR, José Wilson da Silva. **Uma Disciplina Para a Engenharia de Software: Estudo do Personal Software Process (PSP), Proposto Por Watts Humphrey (A Profissionalização do Desenvolvedor de Software):** Universidade Federal de Pelotas, 2000.

KERZNER, Harold. Gestão de Projetos: as melhores práticas: 4. ed. Porto Alegre: Bookman, 2020.

LARSON, Erick W; GRAY, Clifford F. **Gerenciamento de projetos: o processo gerencial**: 6. ed. Porto Alegre: AMGH, 2016.

O'REGAN, Gerard. **Concise Guide to Software Engineering: From Fundamentals to Application Methods**. Cham: Springer, 2017.

PMBOK. **Um Guia de Conjunto de Conhecimentos em Gerenciamento de Projetos (Guia PMBOK®)**. Project Management Institute, Four Campus Boulevard, Newton Square, Pennsylvania, USA, Sixth Edition, 2017.

PWC. **Idéias e tendências:** Práticas atuais de gestão de projetos, portfólios e programas. PWC. 2012. Disponível em: < https://www.pwc.com.br/pt/publicacoes/servicos/assets/consultoria-negocios/insights-and-trends-12.pdf >. Acesso em: 28 abr. 2020.

RUP. **Rational Unified Process:** Best Practices for Software Development Teams, 1998. Disponível em: < https://www.ibm.com/developerworks/rational/library/content/03July/1000/1251/1251_bestpractices_TP026B.pdf>. Acesso em: 18 mai. 2020.

SCHWABER, Ken; SUTHERLAND, Jeff. **Guia do Scrum:** Um guia definitivo para o Scrum: As regras do jogo. Jul. 2013. Disponível em: <https://www.scrum.org/resources/scrum-guide/>. Acesso em: 23 mai. 2020.

SILVA, Edson Coutinho da; LOVATO, Leandro Alvares. **FRAMEWORK SCRUM:** EFICIÊNCIA EM PROJETOS DE SOFTWARE. Revista de Gestão e Projetos -gep, São Paulo, v. 7, n. 2, p.1-5, mar. 2016.

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

STACK OVERFLOW. **Developer Survey Results 2019**, 2019. Disponível em: <https://insights.stackoverflow.com/survey/2019>. Acesso em: 26 mai. 2020.

TERLIZZI, M. A; BIANCOLINO, C. A. **Projeto de Software no Setor Bancário: Scrum ou Modelo V**. Rio de Janeiro, 4(1), 46-58.

UOL. Home Office deve crescer 30% no país após coronavirus, diz professor da FGV. **OUL**. São Paulo, 06 abr. 2020. Disponível em: <https://economia.uol.com.br/empregos-e-carreiras/noticias/redacao/2020/04/06/home-office-coronavirus.htm>.Acesso em: 27 mai. 2020.

ANEXO

ANEXO A – *PSP TIME RECORDING LOG*

Project	*Phase*	*Start Date and Time*	*Int. Time*	*Stop Date and Time*	*Delta Time*	*Comments*

INSTRUCTIONS

Purpose	*Use this form to record the time you spend on each project activity.* *For the PSP, phases often have only one activity; larger projects* *usually have multiple activities in a single process phase.* *These data are used to complete the Project Plan Summary.* *Keep separate logs for each program.*
General	*Record all of the time you spend on the project.* *Record the time in minutes.* *Be as accurate as possible.* *If you need additional space, use another copy of the form.* *If you forget to record the starting, stopping, or interruption time for an activity, promptly enter your best estimate.*
Header	*Enter your name and the date.* *Enter the program name and number.* *Enter the instructor's name and the programming language you are using.*
Project	*Enter the program name or number.*
Phase	*Enter the name of the phase for the activity you worked on, e.g.,* *Planning, Design, Test.*
Start Date and Time	*Enter the date and time when you start working on a process activity.*
Interruption Time	*Record any interruption time that was not spent on the process activity.* *If you have several interruptions, enter their total time.* *You may enter the reason for the interrupt in comments.*

Stop Date and Time	*Enter the date and time when you stop working on that process activity.*
Delta Time	*Enter the clock time you actually spent working on the process activity, less the interruption time.*
Comments	*Enter any other pertinent comments that might later remind you of any unusual circumstances regarding this activity.*

ANEXO B – *PSP DEFECT RECORDING LOG*

Project	***Date***	***Number***	***Type***	***Inject***	***Remove***	***Fix Time***	***Fix Ref***
Descripiton:							
Project	***Date***	***Number***	***Type***	***Inject***	***Remove***	***Fix Time***	***Fix Ref***
Descripiton:							
Project	***Date***	***Number***	***Type***	***Inject***	***Remove***	***Fix Time***	***Fix Ref***
Descripiton:							
Project	***Date***	***Number***	***Type***	***Inject***	***Remove***	***Fix Time***	***Fix Ref***
Descripiton:							
Project	***Date***	***Number***	***Type***	***Inject***	***Remove***	***Fix Time***	***Fix Ref***
Descripiton:							

INSTRUCTIONS

Purpose	*Use this form to hold data on the defects you find and correct.* *These data are used to complete the project plan summary.*
General	*Record each defect separately and completely.* *If you need additional space, use another copy of the form.*

Header	*Enter your name and the date.* *Enter the program name and number.* *Enter the instructor's name and the programming language you are using.*
Project	*Give each program a different name or number.* *For example, record test program defects against the test program.*
Date	*Enter the date you found the defect.*
Number	*Enter the defect number.* *For each program or module, use a sequential number starting with 1 (or 001, etc.).*
Type	*Enter the defect type from the defect type list summarized in the* *top left corner of the form.* *Use your best judgment in selecting which type applies.*
Inject	*Enter the phase when this defect was injected.* *Use your best judgment.*
Remove	*Enter the phase during which you fixed the defect.* *This will generally be the phase when you found the defect.*
Fix Time	*Enter the time you took to find and fix the defect.* *This time can be determined by stopwatch or by judgment.*
Fix Ref.	*If you or someone else injected this defect while fixing another defect, record the number of the improperly fixed defect.* *If you cannot identify the defect number, enter an X.*
Description	*Write a succinct description of the defect that is clear enough to later remind you about the error and help you to remember why you made it.*

ANEXO C – *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*

ANEXO D – *PSP0 DEVELOPMENT SCRIPT*

Purpose	*To guide the development of small programs*	
Entry Criteria	*• Requirements statement* *• Project Plan Summary form with estimated program development time* *• Time and Defect Recording logs* *• Defect Type standard*	
Step	***Activities***	***Description***
1	*Design*	*• Review the requirements and produce a design to meet them.* *• Record in the Defect Recording log any requirements defects found.* *• Record time in the Time Recording log.*
2	*Code*	*• Implement the design.* *• Record in the Defect Recording log any requirements or design defects found.* *• Record time in the Time Recording log.*
3	*Compile*	*• Compile the program until there are no compile errors.* *• Fix all defects found.* *• Record defects in the Defect Recording log.* *• Record time in the Time Recording log.*
4	*Test*	*• Test until all tests run without error.* *• Fix all defects found.* *• Record defects in the Defect Recording log.* *• Record time in the Time Recording log.*
Exit Criteria	*• A thoroughly tested program* *• Completed Time and Defect Recording logs*	

ANEXO E – *PSP0 PLANNING SCRIPT*

Purpose	*To guide the PSP planning process*	
Entry Criteria	*• Problem description* *• Project Plan Summary form* *• Time Recording log*	
Step	***Activities***	***Description***
1	*Program Requirements*	*• Produce or obtain a requirements statement for the program.* *• Ensure that the requirements statement is clear and unambiguous.* *• Resolve any questions.*
2	*Resource Estimate*	*Make your best estimate of the time required to develop this program.*
Exit Criteria	*• Documented requirements statement* *• Completed Project Plan Summary form with estimated development time data* *• Completed Time Recording log*	

ANEXO F – PSP0 *PROJECT PLAN SUMMARY*

Time in Phase (min)	***Plan***	***Actual***	***To Date***	***To Date %***
Planning				
Design				
Code				
Compile				
Test				
Postmortem				
Total				
Defects Inject	***Plan***	***Actual***	***To Date***	***To Date %***
Planning				
Design				
Code				
Compile				
Test				
Postmortem				
Total Development				
Defects Removed	***Plan***	***Actual***	***To Date***	***To Date %***
Planning				
Design				

Code			
Compile			
Test			
Postmortem			
Total Development			
After Development			

INSTRUCTIONS

Purpose General	*To hold the plan and actual data for programs or program parts* *"To-Date" is the total actual to-date values for all products developed.*
Header	*Enter your name and the date.* *Enter the program name and number.* *Enter the instructor's name and the programming language you* *are using.*
Time in Phase	*Enter the estimated total time.* *Enter the actual time by phase and the total time.* *To Date: Enter the sum of the actual times for this program plus* *the To-Date times from the most recently developed program.* *To Date %: Enter the percentage of To-Date time in each phase.*
Defects Injected	*Enter the actual defects by phase and the total actual defects.* *To Date: Enter the sum of the actual defects injected by phase* *and the To-Date values for the most recent previously developed* *program.* *To Date %: Enter the percentage of the To-Date defects injected* *by phase.*
Defects Removed	*To Date: Enter the actual defects removed by phase plus the To-* *Date values for the most recent previously developed program.* *To Date %: Enter the percentage of the To-Date defects removed* *by phase.* *After development, record any defects subsequently found during* *program testing, use, reuse, or modification.*

ANEXO G – *PROBE ESTIMATING SCRIPT*

Purpose	*To guide the size and time estimating process using the PROBE method*
Entry Criteria	*• Requirements statement* *• Size Estimating template and instructions* *• Size per item data for part types* *• Time Recording log* *• Historical size and time data*
General	*• This script assumes that you are using added and modified size data as the size-accounting types for making size and time estimates.* *• If you choose some other size-accounting types, replace every "added and modified" in this script with the size-accounting types of your choice.*

Step	***Activities***	***Description***
1	*Conceptual Design*	*Review the requirements and produce a conceptual design.*
2	*Parts Additions*	*Follow the Size Estimating template instructions to estimate the parts additions and the new reusable parts sizes.*
3	*Base Parts and Reused Parts*	*• For the base program, estimate the size of the base, deleted, modified, and added code.* *• Measure and/or estimate the side of the parts to be reused.*
4	*Size Estimating Procedure*	*• If you have sufficient estimated proxy size and actual added and modified size data (three or more points that correlate), use procedure 4A.* *• If you do not have sufficient estimated data but have sufficient plan added and modified and actual added and modified size data (three or more points that correlate), use procedure 4B.* *• If you have insufficient data or they do not correlate, use procedure 4C.* *• If you have no historical data, use procedure 4D.*

4A	*Size Estimating Procedure 4A*	*Using the linear-regression method, calculate the β0 and β1 parameters from the estimated proxy size and actual added and modified size data.* *• If the absolute value of β0 is not near 0 (less than about 25% of the expected size of the new program), or β1 is not near 1.0 (between about 0.5 and 2.0), use procedure 4B.*
4B	*Size Estimating Procedure 4B*	*• Using the linear-regression method, calculate the β0 and β1 parameters from the plan added and modified size and actual added and modified size data.* *• If the absolute value of β0 is not near 0 (less than about 25% of the expected size of the new program), or β1 is not near 1.0 (between about 0.5 and 2.0), use procedure 4C.*
4C	*Size Estimating Procedure 4C*	*If you have any data on plan added and modified size and actual added and modified size, set β0 = 0 and β1 = (actual total added and modified size to date/plan total added and modified size to date).*
4D	*Size Estimating Procedure 4D*	*If you have no historical data, use your judgment to estimate added and modified size.*
5	*Time Estimating*	*• If you have sufficient estimated proxy size and actual development time data (three or more points that correlate), use procedure 5A.* *• If you do not have sufficient estimated size data but have sufficient plan added and modified size and actual development time data (three or more points that correlate), use procedure 5B.* *• If you have insufficient data or they do not correlate, use procedure 5C.* *• If you have no historical data, use procedure 5D.*
5A	*Time Estimating Procedure 5A*	*• Using the linear-regression method, calculate the β0 and β1 parameters from the estimated proxy size and actual total development time data.* *• If β0 is not near 0 (substantially smaller than the expected development time for the new program), or β1 is not within 50% of 1/(historical productivity), use procedure 5B.*

5B	*Time Estimating Procedure 5B*	*• Using the linear-regression method, calculate the β0 and β1 regression parameters from the plan added and modified size and actual total development time data.* *• If β0 is not near 0 (substantially smaller than the expected development time for the new program), or β1 is not within 50% of 1/(historical productivity), use procedure 5C.*
5C	*Time Estimating Procedure 5C*	*• If you have data on estimated—added and modified size and actual development time, set β0 = 0 and β1 = (actual total development time to date/estimated – total added and modified size to date).* *• If you have data on plan – added and modified size and actual development time, set β0 = 0 and β1 = (actual total development time to date/plan total added and modified size to date).* *• If you only have actual time and size data, set β0 = 0 and β1 = (actual total development time to date/actual total added and modified size to date).*
5D	*Time Estimating Procedure 5D*	*If you have no historical data, use your judgment to estimate the development time from the estimated added and modified size.*
6	*Time and Size Prediction Intervals*	*• If you used regression method A or B, calculate the 70% prediction intervals for the time and size estimates.* *• If you did not use the regression method or do not know how to calculate the prediction interval, calculate the minimum and maximum development time estimate limits from your historical maximum and minimum productivity for the programs written to date.*
Exit Criteria	*• Completed estimated and actual entries for all pertinent size categories* *• Completed PROBE Calculation Worksheet with size and time entries* *• Plan and actual values entered on the Project Plan Summary*	

ANEXO H – *PSP0 POSTMORTEM SCRIPT*

Purpose		*To guide the PSP postmortem process*
Entry Criteria		*• Problem description and requirements statement* *• Project Plan Summary form with development time data* *• Completed Time and Defect Recording logs* *• A tested and running program*
Step	***Activities***	***Description***
1	*Defect Recording*	*• Review the Project Plan Summary to verify that all of the defects found in each phase were recorded.* *• Using your best recollection, record any omitted defects.*
2	*Defect Data Consistency*	*• Check that the data on every defect in the Defect Recording log are accurate and complete.* *• Verify that the numbers of defects injected and removed per phase are reasonable and correct.* *• Using your best recollection, correct any missing or incorrect defect data.*
3	*Time*	*• Review the completed Time Recording log for errors or omissions.* *• Using your best recollection, correct any missing or incomplete time data.*
Exit Criteria		*• A thoroughly tested program* *• Completed Project Plan Summary form* *• Completed Time and Defect Recording logs*

ANEXO I – *PSP CODE REVIEW SCRIPT*

Purpose		*To guide you in reviewing programs*
Entry Criteria		*• A completed and reviewed program design* *• Source program listing* *• Code Review checklist* *• Coding standard* *• Defect Type standard* *• Time and Defect Recording logs*
General		*While following this process* *• do the review from a source-code listing; do not review on the screen!* *• record all time spent in the Time Recording log (LOGT)* *• record all defects found in the Defect Recording log (LOGD)*
Step	***Activities***	***Description***
1	*Review*	*• Follow the Code Review checklist.* *• Review the entire program for each checklist category; do not try to review for more than one category at a time!* *• Check off each item as it is completed.* *• For multiple procedures or programs, complete a separate checklist for each.*
2	*Correct*	*• Correct all defects.* *• If the correction cannot be completed, abort the review and return to the prior process phase.* *• To facilitate defect analysis, record all of the data specified in the Defect Recording log instructions for every defect.*
3	*Check*	*• Check each defect fix for correctness.* *• Re-review all design changes.* *• Record any fix defects as new defects and, where you know the number of the defect with the incorrect fix, enter it in the fix defect space.*
Exit Criteria		*• A fully reviewed source program* *• One or more Code Review checklists for every program reviewed* *• All identified defects fixed* *• Completed Time and Defect Recording logs*

ANEXO J – *PSP CODE REVIEW CHECKLIST*

Propose	*To guide you in conducting an effective code review*				
General	*• Review the entire program for each checklist category; do not attempt to review for more than one category at a time!* *• As you complete each review step, check off that item in the box at the right.* *• Complete the checklist for one program or program unit before reviewing the next.*				
Complete	*Verify that the code covers all of the design*				
Includes	*Verify that the includes are complete.*				
Initialization	*Check variable and parameter initialization.* *• at program initiation* *• at start of every loop* *• at class/function/procedure entry*				
Calls	*Check function call formats.* *• pointers* *• parameters* *• use of '&'*				
Names	*Check name spelling and use.* *• Is it consistent?* *• Is it within the declared scope?* *• Do all structures and classes use '.' reference?*				
Strings	*Check that all strings are* *• identified by pointers* *• terminated by NULL*				
Pointers	*Check that all* *• pointers are initialized NULL* *• pointers are deleted only after new* *• new pointers are always deleted after use*				
Output Format	*Check the output format.* *• Line stepping is proper.* *• Spacing is proper.*				
() Pairs	*Ensure that () are proper and matched.*				

Logic Operators	*• Verify the proper use of ==, =, ||, and so on.* *• Check every logic function for ().*				
Line-by-line check	*Check every line of code for* *• instruction syntax* *• proper punctuation*				
Standards	*Ensure that the code conforms to the coding standards.*				
File Open and Close	*Verify that all files are* *• properly declared* *• opened* *• closed*				

ANEXO K –*PSP2 DESIGN REVIEW SCRIPT*

Purpose		*To guide you in reviewing detailed designs*
Entry Criteria		*• Completed program design* *• Design Review checklist* *• Design standard* *• Defect Type standard* *• Time and Defect Recording logs*
General		*Where the design was previously verified, check that the analyses* *• covered all of the design and were updated for all design changes* *• are correct, clear, and complete* *Where the design is not available or was not reviewed, do a complete* *design review on the code and fix all defects before doing the code* *review.*
Step	***Activities***	***Description***
1	*Review*	*Examine the program and checklist and decide on a review strategy.*
2	*Correct*	*• Follow the Design Review checklist.* *• Review the entire program for each checklist category; do not try to* *review for more than one category at a time!* *• Check off each item as you complete it.* *• Complete a separate checklist for each product or product segment* *reviewed.*

3	*Check*	*• Check each defect fix for correctness.* *• Re-review all changes.* *• Record any fix defects as new defects and, where you know the defective defect number, enter it in the fix defect space.*
Exit Criteria		*• A fully reviewed detailed design* *• One or more Design Review checklists for every design reviewed* *• All identified defects fixed and all fixes checked* *• Completed Time and Defect Recording logs*

ANEXO L – *PSP2 DESIGN REVIEW CHECKLIST*

Propose	*To guide you in conducting an effective design review*				
General	*• Review the entire program for each checklist category; do not attempt to review for more than one category at a time!* *• As you complete each review step, check off that item in the box at the right.* *• Complete the checklist for one program or program unit before reviewing the next.*				
Complete	*Verify that the design covers all of the applicable requirements.* *• All specified outputs are produced.* *• All needed inputs are furnished.* *• All required includes are stated.*				
External Limits	*Where the design assumes or relies upon external limits, determine if behavior is correct at nominal values, at limits, and beyond limits.*				

Logic	*• Verify that program sequencing is proper.* *• Stacks, lists, and so on are in the proper order.* *• Recursion unwinds properly.* *• Verify that all loops are properly initiated, incremented, and terminated.* *• Examine each conditional statement and verify all cases.*				
Internal Limits	*Where the design assumes or relies upon internal limits, determine if behavior is correct at nominal values, at limits, and beyond limits.*				
Special Cases	*• Check all special cases.* *• Ensure proper operation with empty, full, minimum, maximum, negative, zero values for all variables.* *• Protect against out-of-limits, overflow, underflow conditions.* *• Ensure "impossible" conditions are absolutely impossible.* *• Handle all possible incorrect or error conditions.*				
Functional Use	*• Verify that all functions, procedures, or methods are fully understood and properly used.* *• Verify that all externally referenced abstractions are precisely defined.*				

System Considerations	*• Verify that the program does not cause system limits to be exceeded.* *• Verify that all security-sensitive data are from trusted sources.* *• Verify that all safety conditions conform to the safety specifications.*				
Names	*Verify that* *• all special names are clear, defined, and authenticated* *• the scopes of all variables and parameters are self-evident or defined* *• all named items are used within their declared scopes*				
Standards	*Ensure that the design conforms to all applicable design standards.*				