ESCOLA
SUPERIOR
DE TECNOLOGIA
E GESTÃO
POLITÉCNICO
DO PORTO

M

MESTRADO
ÁREA DE ESPECIALIZAÇÃO

A importância do uso das metodologias híbridas em gestão de projetos de desenvolvimento de software

Elisângela Loss Cunha

11/2021

A importância do uso das metodologias híbridas em gestão de projetos de desenvolvimento de software

Elisângela Loss Cunha

MESTRADO
ÁREA DE ESPECIALIZAÇÃO

ESCOLA
SUPERIOR
DE TECNOLOGIA
E GESTÃO
POLITÉCNICO
DO PORTO

Nome: Elisângela Loss Cunha Título: A importância do uso das metodologias híbridas em gestão de projetos de desenvolvimento de software

11/2021

ESCOLA
SUPERIOR
DE TECNOLOGIA
E GESTÃO
POLITÉCNICO
DO PORTO

M

MESTRADO
ÁREA DE ESPECIALIZAÇÃO

A importância do uso das metodologias híbridas em gestão de projetos de desenvolvimento de software

Nome: Elisângela Loss Cunha

Orientador: José Ângelo da Costa Pinto

RESUMO

Nos últimos anos, o setor de desenvolvimento de *software* apresentou grande crescimento em todo o mundo. Para gerenciar os projetos oriundos desta demanda crescente, mostraram-se necessários novos modelos de gestão, além do aprimoramento dos modelos existentes.

Os modelos existentes, também conhecidos como tradicionais, se tornaram-se muito burocráticos para o acompanhamento dos projetos e suas mudanças repentinas. Para solucionar este problema, foram inseridos os modelos ágeis, especialmente o Scrum, XP (*Extreme Programming)* e Kanban, como modelos menos burocráticos de gerenciar os projetos, já que uma das razões mais fortes para a adoção dos modelos ágeis é a velocidade de entrega dos projetos. Porém, nenhum dos modelos se mostraram 100% eficazes e muitas empresas têm adotado os modelos híbridos, que consistem na junção de mais de um modelo de gestão de projetos (especialmente ScrumBan e Scrum/XP (*Extreme Programming*)). Apesar disso, estes modelos também revelam alguns pontos negativos como a documentação insatisfatória e o âmbito pouco definido.

Com o intuito de solucionar problemas de gestão, modificação do âmbito e documentação ineficiente, foi criado o modelo híbrido VIPKS, que resultou da combinação entre os modelos ágeis Scrum, Kanban e o modelo clássico orientado pelo Guia PMBOK®, passível de ser aplicado em pequenas e médias empresas de desenvolvimento de *software* que desejam ter maior eficácia e agilidade na execução de seus projetos com constantes entregas e interação entre a equipe e o cliente, sem abdicar da gestão da mudança e documentação.

O modelo híbrido VIPKS utiliza o seguinte ciclo de vida como base para sua gestão: Análise e especificação de requisitos; Desenvolvimento; Validação do *software*; Entrega e Implantação.

A validação do modelo é feita através de um investigação-ação experimental em um caso real e com um questionário que foi respondido pela equipe em que se pôde constatar que o modelo híbrido VIPKS se revelou benéfico e adequado para o planejamento e execução de projetos de desenvolvimento de *software*.

Palavras-chave: Modelos Ágeis, Scrum, Guia PMBOK®, Modelo Híbrido VIPKS, Gestão de Projetos, Kanban.

ABSTRACT

In the last years, the *software* development sector has shown great growth all over the world. To manage the projects arising from this growing demand, new management models were needed, in addition to the improvement of existing models.

The existing models, also known as “traditional”, have become too bureaucratic for monitoring projects and their sudden changes. To solve this problem, agile models were introduced, especially Scrum, XP (Extreme Programming) and Kanban, as less bureaucratic models for managing projects since one of the strongest reasons for adopting agile models is the speed of delivery of projects.

However, none of the models proved to be 100% effective and many companies have adopted hybrid models, which consist of joining more than one project management model (especially ScrumBan and Scrum/XP (Extreme Programming)). Despite this, these models also reveal some negative points such as poor documentation and poorly defined scope.

In order to solve management problems, scope modification and inefficient documentation, the hybrid model VIPKS was created, which resulted from the combination of agile models Scrum, Kanban and the classic model guided by the PMBOK® Guide, which can be applied in small and medium-sized software development companies that wish to have greater efficiency and agility in the execution of their projects with constant deliveries and interaction between the team and the client, without opening up change management and documentation.

The hybrid model VIPKS uses the following lifecycle as the basis for its management: Analysis and specification of requirements; Development; Software validation; Delivery and Deployment.

The validation of the model is done through an experimental case study in a real case and with a questionnaire that was answered by the team, where it could be verified that the VIPKS hybrid model proved to be beneficial and suitable for the planning and execution of development projects of software.

Keywords: Agile Models, Scrum, PMBOK Guide®, VIPKS Hybrid Model, Project Management, Kanban.

ÍNDICE GERAL

ÍNDICE DE FIGURA

ÍNDICE DE TABELA

ABREVIATURAS E SIGLAS

XP	*Extreme Programming*
PMBOK	*Project Management Body of Knowledge*
ISO	*International Organization for Standardization*
PMI	*Project Management Institute*
TAP	Termo de Abertura do Projeto
FDD	*Feature Drive Development*
DSDM	*Dynamic System Development Model*
ASD	*Adpative Software Development*
JIT	*Just in time*
TPS	Sistema de Produção Toyota
WIP	*Work in progress*
RAD	Controle para desenvolvimento rápido de aplicações
VIPKS	Virtual Innovations PMBOK Kanban Scrum
GP	Gestor de Projeto
TL	*Team Leader*

1 Introdução

Para que seja feito o desenvolvimento de um *software,* é necessário um conjunto de tarefas e resultados associados, organizados de acordo com o ciclo de vida de um projeto. A principal função do ciclo de vida de um projeto é indicar as fases, atividades, entregas e responsabilidades de cada envolvido no processo de desenvolvimento de *software*.

Para Soares, o resultado do processo de desenvolvimento de *software* é um produto que reflete a forma como o processo foi conduzido (M. Soares, 2004).

Fowler diz que o desenvolvimento de um *software* é muitas vezes feito de forma caótica. Com isto, existe a necessidade de ter um modelo de gestão para o melhor desempenho dos projetos (Fowler, 2001).

Couto afirma que embora existam vários processos para o desenvolvimento de um *software*, existem várias atividades comuns em todos, são eles: análise e engenharia de sistemas, análise de requisitos de *software*, projeto, codificação, testes e manutenção (Couto et al., 2017).

Os modelos de gestão de projetos de *software*, tem como objetivo, fazer com que o desenvolvimento do mesmo, siga modelos específicos para cada aplicação desenvolvida (Taroco & Werner[1], 2015).

Silva, Souza e Camargo dizem que um modelo de desenvolvimento tem a função de formalizar a ordem de desenvolver um *software*, organizar a fase do ciclo de vida do projeto e ter o equilíbrio entre os processos e os dados. Há vários tipos de modelos de gerenciamento de projetos de *software* e cada empresa pode construir seu próprio modelo conforme as suas necessidades, porém todas elas têm o mesmo objetivo de formalizar a fase de desenvolvimento e organização das informações do projeto (Silva, Souza, & Camargo, 2013).

O grande objetivo dos modelos de gerenciamento de projetos de *software* é aumentar a probabilidade de entrega bem-sucedida dos resultados do projeto, através de um melhor controle do seu âmbito, tempo de entrega e redução de riscos, contribuindo para a satisfação do cliente e para o valor gerado no projeto.

Os modelos de gerenciamento de projetos são divididos em ágeis e clássicos. Neto afirma que o modelo ágil tem uma forte ligação com os projetos que possuem grandes indícios de mudanças e têm como principais características as frequentes interações do cliente e grande flexibilidade proporcionando práticas adaptativas fornecendo transparência aos envolvidos no processo, mas tem carência de alguns processos que estão presentes nos modelos clássicos (Neto, 2018).

Silva, Souza e Camargo afirmam que os modelos ágeis e entregas rápidas podem contribuir no desenvolvimento de projetos de *software* para qualquer tipo de empresa se forem agregados ao modelo clássico (Silva et al., 2013).

A utilização de mais de um modelo de gestão de projetos vem sendo cada vez mais utilizado nos projetos de desenvolvimento de *software*. Lendo o relatório da empresa VersionOne de 2016, podemos constatar que 10% dos entrevistados que utilizam os modelos ágeis usam os modelos híbridos Scrum/XP e 8% utilizam a combinação de diversos modelos e 7% utilizam o ScrumBan (VersionOne, 2016). Esta afirmação se confirma na pesquisa feita em 2020 pela 14ª pesquisa anual ágil, em que 10% utilizam o ScrumBan, 9% utilizam a combinação de diversos modelos e 8% utilizam o Scrum/XP (StateOfAgile, 2020).

1.1 Motivação e objetivos do trabalho

Atuando na gestão de projetos de desenvolvimento de *software* e consultoria nos últimos cinco anos, o autor tem encontrado grande dificuldade em utilizar os modelos ágeis devidos problemas na gestão do âmbito e na sua constante modificação.

Para Portillo, mudanças no âmbito do projeto podem impactar os custos e o cronograma do projeto de maneira diferente, dependendo de quando essas mudanças são implementadas no ciclo de vida do projeto (Portillo, 2010).

A empresa na qual o autor atua e é, atualmente, sócio, conta com cerca de 50 colaboradores sendo a maioria desenvolvedores. Atualmente é utilizado o modelo ágil Scrum ou parte dele na gestão dos projetos e o Kanban como ferramenta de gestão em quadro (trello ou zoho) (“https://trello.com,” n.d.)(“https://www.zoho.com,” n.d.).

Embora se utilizem estes modelos na gestão dos projetos, tem-se encontrado algumas dificuldades, como:

- Gestão do âmbito - como existe pouco tempo para definição do âmbito e documentação resumida (normalmente não se possuem os detalhes funcionais do projeto) pode existir divergência entre o que o cliente e a equipe envolvida no desenvolvimento do projeto esperam do produto. A ausência ou um inadequado controle de mudanças nos projetos pode causar problemas na gestão do âmbito, com problemas de prazos e custos.
- Modificação do âmbito - a falta de clareza na definição do âmbito, nos critérios de alterações e na validação da consistência das alterações solicitadas com os objetivos do projeto faz com que quando estamos na reta final do projeto o cliente consiga argumentar que alguma funcionalidade não desenvolvida estava prevista.
- Documentação - os projetos possuem uma documentação resumida. Com isto, identificamos dois problemas:
    - Quando é feito um pedido de alteração a uma funcionalidade antiga, nem sempre é possível, de forma rápida, compreender o seu impacto no projeto como um todo.

- Se alguma parte envolvida no projeto deixar o projeto, podem vir a existir problemas nas entregas, por não existir documentação suficiente para orientar o novo integrante do projeto, ocasionando perca de tempo e possíveis problemas nas entregas.

Com o intuito de mitigar as dificuldades encontradas anteriormente, o autor uniu a sua vivência com o mundo empresarial e o mundo acadêmico e se desafiou a estudar os modelos ágeis mais utilizados nas empresas e compará-los com os modelos clássicos com o objetivo de fazer o estudo de um modelo híbrido utilizando as melhores práticas dos modelos clássicos e ágeis.

Com base neste estudo será feito a aplicação de um modelo híbrido que será utilizado em projeto piloto de desenvolvimento de *software* na empresa que atua. O objetivo proposto da aplicação deste modelo híbrido é ajudar a solucionar os problemas mencionados acima.

1.2 Metodologia de investigação

Para o estudo proposto, será utilizado a metodologia de investigação-ação com uma análise de dados qualitativos e quantitativos.

A investigação-ação, tal como já foi referido anteriormente, é um processo contínuo da ação reflexiva ao desenvolvimento de conhecimentos, habilidades e atitudes em que todos participam, investigando as suas próprias práticas sociais a fim de conhecê-las e melhorá-las (Fonseca, 2012).

A investigação-ação exige uma estrutura de relação entre os pesquisadores e pessoas envolvidas no estudo da realidade do tipo participativo. A participação dos pesquisadores é explicitada dentro do processo do “conhecer” com os “cuidados” necessários para que haja reciprocidade/complementariedade por parte das pessoas e grupos implicados, que têm algo a “dizer e a fazer”. Não se trata de um simples levantamento de dados (Baldissera, 2012).

Baldissera afirma que a investigação-ação como metodologia de investigação agrega várias técnicas de pesquisa. Utiliza-se de técnicas de coleta e interpretação dos dados, de intervenção na solução de problemas e organização de ações (Baldissera, 2012).

Meirinhos salienta a relevância de utilizar, em alguns métodos de investigação, simultaneamente dados qualitativos e quantitativos. A utilização de dados qualitativos e quantitativos, na mesma investigação, vai no sentido de olhar para estas metodologias como complementares e não como opostas ou rivais (Meirinhos, 2010).

A investigação-ação proposta será analisada com uma abordagem qualitativa com uma perspectiva mais interpretativa e construtiva e uma análise quantitativa na coleta de informação do questionário de forma simples com média e percentual. Foi escolhido as duas abordagens de pesquisa para ter uma visão sistêmica e tem como objetivo abranger a máxima amplitude na descrição, explicação e compreensão com a análise do modelo de gestão de projeto de desenvolvimento de

software já utilizado atualmente na empresa estudada (Scrum) e com os resultados obtidos na aplicação do modelo híbrido VIPKS através de recolha de evidências.

Para análise da investigação-ação será levado em consideração as seguintes fases:

1 Planejamento - será definido o aspecto relacionado com a concepção da investigação.

2 Documentação formal - o levantamento da documentação será feito através da recolha de informações por questionário, relatórios de atividades, memorando de reuniões e análise dos dados.

3 Análise dos dados - os dados serão analisados através de observações diretas pelo autor que participará dos projetos e tomará notas de ações e descrições do contexto organizacional, citações e revisão de relatórios.

4 Resultados - os resultados serão apurados através de questionários estruturados sobre questões da problemática.

5 Análise de dados - os dados serão analisados levando em consideração as análises exploratórias através de construção da explicação, notas de campo e sequência lógica de evidências.

1.3 Estrutura da dissertação

Esta dissertação está dividida em cinco capítulos, nos quais pretende-se abordar diversos modelos de gestão de projeto de desenvolvimento de *software* e a criação e validação de um modelo híbrido através de aplicação em um projeto piloto.

No segundo capítulo serão abordados os modelos de desenvolvimento de um *software* e o ciclo de vida de um projeto. Estes modelos foram divididos em clássicos e ágeis que explicaremos o que é e como funciona os modelos mais conhecidos e mencionados na literatura. Será mencionado também o que são os modelos híbridos.

No terceiro capítulo será apresentada uma comparação entre os modelos ágeis com a combinação dos modelos Scrum e Kanban e os modelos Scrum e XP e uma comparação entre os modelos clássicos e ágeis com a combinação do modelo clássico orientado pelo Guia PMBOK® e o modelo ágil Scrum.

No quarto capítulo será explicado o modelo híbrido VIPKS e será apresentada a sua validação através de projeto piloto em um projeto de desenvolvimento de *software* numa empresa e a sua análise através de questionário respondido pela equipe envolvida no desenvolvimento do projeto e será ainda realizada uma comparação do modelo VIPKS com outros modelos estudados.

Por fim, no quinto capítulo, será feito a conclusão e sugestões de estudos futuros.

2 Modelos de Gerenciamento de Projetos

Os modelos de gerenciamento de projetos ajudam os gerentes e demais membros envolvidos no projeto em seu planejamento, implantação e realização do projeto a fim de atingir o objetivo a qual foi proposto. Esses modelos podem ter um estilo clássico ou ágil.

Nos últimos anos vem crescendo a atenção em volta de modelos denominados híbridos, os quais são objetos de estudo da presente pesquisa. Esses modelos visam extrair as vantagens de ambas as abordagens através da combinação entre suas práticas, técnicas e ferramentas.

Neste capítulo pretende-se abordar qual é o ciclo da vida de um projeto e os modelos de gerenciamento de projetos de *software* mais utilizados segundo a literatura estudada, como eles funcionam e suas falhas.

2.1 Ciclo de vida de um projeto de desenvolvimento de *software*

O ciclo de vida de um projeto contém os processos, atividades e tarefas envolvidas no desenvolvimento, operação e manutenção de um *software* abrangendo toda a vida do sistema (“Norma NBR ISO/IEC 12207,” 1998).

Entre as atividades associadas no ciclo de vida de um projeto de desenvolvimento de *software*, existem quatro atividades principais, são elas: análise e especificação de requisitos, desenvolvimento, validação do *software* (testes técnicos e funcionais), entrega e implantação.

Abordaremos a seguir cada atividade para entendermos as etapas do ciclo de vida de um projeto.

1 Análise e especificação de requisitos

Esta é a primeira etapa do desenvolvimento de um *software*, que é a definição do produto, seus requisitos (âmbito), restrições e suas funcionalidades. Com base nestas informações o gestor do projeto pode fazer estimativas iniciais de recursos, custos e prazos.

Stoica, Mircea e Chilic-Micu afirmam que a análise de requisitos é a etapa mais importante do projeto (STOICA, MIRCEA, & GHILIC-MICU, 2013).

Segundo Sommerville , os incrementos iniciais do projeto, permitem aos clientes ainda durante o desenvolvimento, verificarem os requisitos solicitados na prática e requererem alterações a serem consideradas, já nos incrementos posteriores do projeto. Desta forma, já é possível identificar e priorizar as funcionalidades do projeto, podendo assim verificar falhas e solicitar novas alterações ou correções, não havendo necessidade de chegar ao final do desenvolvimento para conhecê-los e corrigi-los. (Sommerville, 2003)

O âmbito do projeto é o primeiro desafio de um projeto, no qual será feito o desenho de todo o projeto, delimitando suas fases e entendendo o que o cliente deseja com detalhes para que não existam grandes mudanças ao longo do projeto, o que na prática é muito comum de acontecer. Nesta etapa deve ser elaborado o plano de projeto configurando o processo a ser utilizado ao decorrer do projeto.

Este plano é revisado por todas as partes interessadas e a melhor abordagem é selecionada, com base em alguns parâmetros como: avaliação de risco, robustez do produto, design, orçamento e restrições de tempo.

As mudanças de âmbito devem ocorrer sempre na fase de iniciação e planejamento do projeto para que exista menos impacto, evitando desperdício de tempo com alteração do cronograma, custo adicionais com recursos e retrabalho.

Mudanças no âmbito do projeto podem impactar os custos e o cronograma do projeto de maneira diferente, dependendo de quanto essas mudanças são implantadas no ciclo de vida do projeto (Portillo, 2010).

Nesta etapa é preciso muita interação entre os envolvidos no projeto (gestor do projeto, desenvolvedores, designer, etc.) e o cliente para entender e validar tudo que o cliente deseja que o produto possua, e não pode existir nenhuma dúvida. Devemos garantir que os requisitos necessários funcionais e não funcionais de um projeto devem estar de acordo com a expectativa do cliente e garantir que toda a equipe envolvida no projeto esteja envolvida, validando e levantando dúvidas de todas as etapas do projeto a ser desenvolvido, desde a iniciação do projeto (levantamento dos requisitos necessários) e planejamento para que seja certificado que todas as questões técnicas, de entregas e cronogramas sejam respondidas.

O modelo do ciclo de vida a ser utilizado no projeto deve ser definido nesta etapa do projeto, pois é nesta etapa que será feito a modelagem dos processos e será definido como será gerido as etapas seguintes.

2 Desenvolvimento

Nesta etapa é feito todo o desenvolvimento do *software* segundo os requisitos definidos nas especificações. Esta etapa envolve atividades de *design* e codificação.

Nesta fase são propostos modelos que são implementados em alguma linguagem de programação.

Nesta etapa todo o *software* é testado, documentado e validado para garantir que todas as funcionalidades foram implementadas de acordo com o planejado.

Alguns modelos de processos (modelos ágeis) preveem a realização dos testes já nas primeiras etapas. Estes testes são divididos em técnicos e funcionais.

Nos testes técnicos é analisado e validado os códigos-fonte e integrações com outros sistemas. Segundo Souza e Gaparotto, os testes unitários validam a menor parte do código-fonte, no teste de integração é validado se a integração entre sistemas, componentes ou outras funcionalidades foi feita corretamente (K. P. de Souza & Gasparotto, 2013).

Nos testes funcionais é validado se o produto está funcionando como se espera e se está de acordo com a expectativa do cliente. Nesta fase, o teste é exercitado como um todo, a fim de encontrar discrepâncias entre os funcionamentos especificados nos requisitos e o construído (K. P. de Souza & Gasparotto, 2013).

Também é feito um teste de aceitação que é realizado pelo cliente no momento da entrega do sistema, ou de parte dele, este teste é executado para garantir que o *software* satisfaz os requisitos solicitados pelo cliente.

Caso seja encontrado algum erro (conhecido como *bugs*) em qualquer um dos testes, será feito os devidos acertos antes de seguirmos para última etapa.

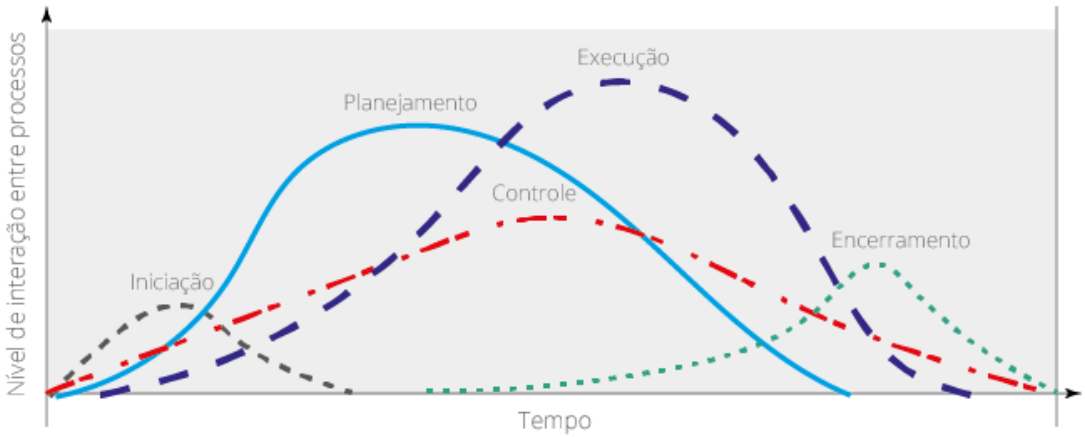
3 Entrega e Implantação

Nesta etapa o *software* deve ser instalado em ambiente de produção no qual deve ser feito a configuração, o treinamento dos usuários e se necessário a conversão da base dos dados.

Após validação de todo o *software* no ambiente de produção é feito as correções e manutenções adaptativas, corretivas e evolutivas.

Na Figura 1 conseguimos visualizar o ciclo de vida de um projeto de desenvolvimento de *software*.

Figura 1: Ciclo de vida de um projeto de *software*



Fonte: (Espinha, 2020)

2.2 Modelos clássicos de gerenciamento de projetos

Na década de 70, o desenvolvimento de *software* carecia de organização, planejamento e estrutura originando com alguma frequência, produtos de má qualidade que não correspondiam às reais necessidades dos clientes (Barry Boehm, 2003).

Modelos vindos da abordagem clássica, envolvem que propõem um levantamento e documentação de um conjunto completo de requisitos, seguido pelo planejamento de como esse projeto será desenvolvido ao longo do tempo, execução do que foi planejado, controle e acompanhamento do progresso do projeto a fim de assegurar que o que foi planejado está sendo seguido, e por fim, o encerramento desse. Esses modelos, portanto, são baseados em uma série sequencial de etapas (Mahdi JAVANMARD, 2015).

Nos modelos clássicos os projetos são desenhados desde o início em que são traçados diretrizes e planejamentos para o projeto como um todo. Franzen e Lutz afirmam que esse fluxo de trabalho obriga a empresa a ser bastante assertiva no planejamento inicial, visto que a sequência do projeto dependerá dele (Franzen & Lutz, 2018).

Segundo Trigo e Barreto, o modelo clássico trabalha todo o projeto como um projeto único. É necessário seguir todas as etapas para obter o *feedback* no final (Trigo & Barreto, 2019).

Existem diversos modelos clássicos no desenvolvimento de *software*. Segundo Bruce, os modelos clássicos mais usados são: modelo em cascata, modelo incremental e modelo em espiral (Bruce Maxim, 2014).

Podemos citar também como modelo clássico o Guia PMBOK® que é um modelo clássico muito seguido pelas empresas.

No capítulo seguinte pretende-se apresentar estes modelos e como eles funcionam, pormenorizando o modelo orientado pelo Guia PMBOK®.

2.2.1 Modelo em cascata

O modelo em cascata é um modelo clássico que foi criado em 1966, porém, formalizado somente por volta de 1970. Este modelo define que as fases serão sequenciais, na qual uma fase tem que estar completa antes de passar para a próxima. O modelo em cascata foi o primeiro a ser usado pela engenharia de *software* na década de 70 (M. D. S. Soares, 2004).

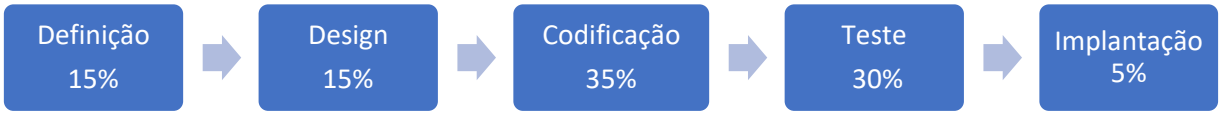
Soares afirma que o modelo em cascata é derivado de outras engenharias tradicionais como por exemplo: Civil, elétrica, naval, etc (M. D. S. Soares, 2004).

Neste modelo existe uma sequência de passos a serem seguidos e cada etapa deve ser concluída para que a próxima possa ser iniciada. O fato de uma etapa só iniciar após a conclusão de

outra, contribui para a detecção dos erros somente na etapa seguinte. Tal constatação reforça o ponto de que, apesar de ser um modelo bastante conceituado e amplamente utilizado mesmo atualmente, o processo de *software* denominado cascata apresenta pontos de melhoria na estrutura e até mesmo na organização de suas etapas (Franzen & Lutz, 2018).

O modelo em cascata é composto por atividades sequenciais de definição, design, codificação, teste e implantação como podemos ver na Figura 2.

Figura 2: Modelo em cascata



Fonte: (Mahdi JAVANMARD, 2015)

2.2.2 Modelo incremental

O modelo incremental surge para melhorar o modelo em cascata, pois permite entregar partes do projeto enquanto este se realiza. Embora seja dividido em incrementos, cada incremento passa pelas fases do modelo em cascata. Por norma, o primeiro incremento é o núcleo do sistema, implementando os requisitos mais básicos e aprendendo com este para os próximos incrementos.

Segundo Bruce, no modelo incremental prioriza-se a comunicação entre os desenvolvedores e os clientes, dando-se preferência a entrega do que a análise do projeto. Este modelo é realizado em diversas etapas por meio de entregas constantes, desenvolvendo de forma interativa, uma etapa de cada vez a ser incrementada, com a participação do cliente e constante *feedback* (Bruce Maxim, 2014).

Segundo Sommerville, os incrementos iniciais do projeto, permitem aos clientes ainda durante o desenvolvimento, verificarem os requisitos solicitados na prática e requererem alterações a serem consideradas, já nos incrementos posteriores do projeto (Sommerville, 2003).

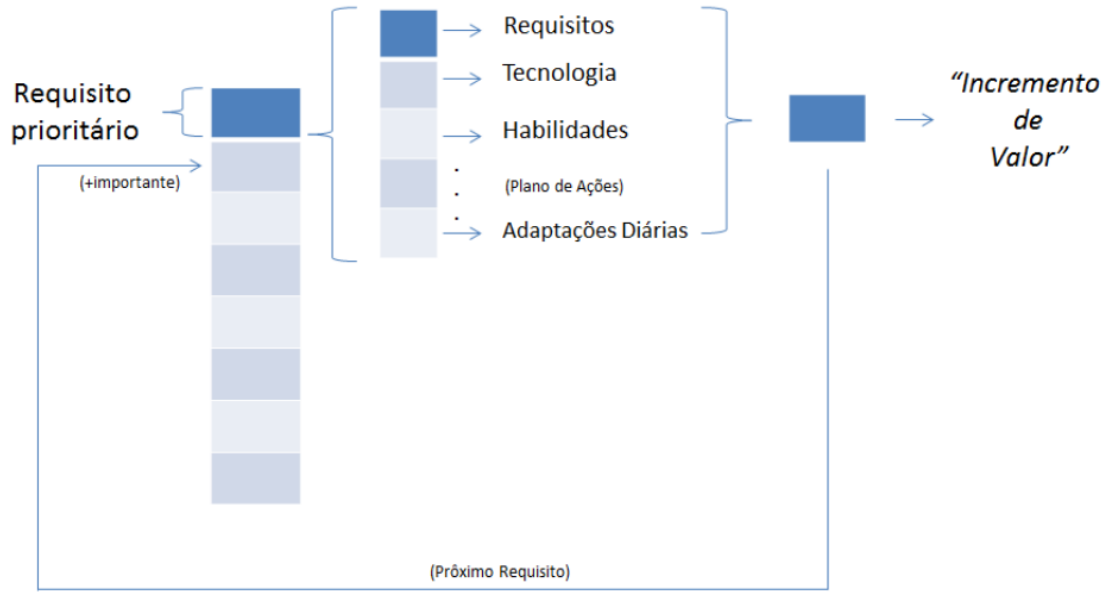
Desta forma, já é possível identificar e priorizar as funcionalidades do projeto, podendo assim verificar falhas e solicitar novas alterações ou correções, não havendo necessidade de chegar ao final do desenvolvimento para conhecê-los e corrigi-los. Todavia, diferente do modelo em cascata, cada etapa tem sua vez para acontecer e, ao término de todas, o projeto termina, no modelo incremental as atividades de especificação, projeto, implementação e validação são intercaladas, acontecendo em cada nova versão, com rápido *feedback* entre todas as atividades (Sommerville, 2003).

Os clientes podem estabelecer as prioridades das partes do sistema a serem desenvolvidas especificando as mais úteis primeiro. Após o cliente estabelecer as funcionalidades necessárias, são criadas as fases de entrega, em cada fase é fornecido um conjunto de funcionalidades do sistema.

Segundo Libardi, como se trata de um modelo incremental, no início de cada iteração a equipe analisa o que deve ser feito e então seleciona aquilo que acreditam poder se tornar um incremento de valor ao produto ao final da iteração (Francischini & Crist, 2016).

Na Figura 3 conseguimos visualizar de forma resumida o funcionamento do modelo incremental no qual a cada interação a equipe analisa os requisitos, a tecnologia e suas habilidades e então se dividem para construir e entregar o melhor produto possível adaptando-se diariamente conforme surjam as complexidades e dificuldades.

Figura 3: Modelo incremental resumido



Fonte: (J. Souza, 2018) (Francischini & Crist, 2016) Adaptado de Libardi (2010)

2.2.3 Modelo em espiral

Criado por Barry Boehm em 1988, o modelo em espiral é uma melhoria do modelo incremental e possui esse nome por causa de sua representação, em cada volta no espiral percorre todas as fases do processo de *software*. As voltas devem ser repetidas quantas vezes forem necessárias até que o *software* possa ser completamente entregue.

O modelo em espiral foi definido por Barry Boehm, com base na experiência com vários refinamentos do modelo em cascata aplicado a grandes projetos de *software*.

Segundo Sommerville, o modelo em espiral combina prevenção e tolerância a mudanças, assume que mudanças são um resultado de riscos de projeto e inclui atividades explícitas de gerenciamento de riscos para sua redução (Sommerville, 2003).

Pressman também diz que o modelo em espiral é uma abordagem realista do desenvolvimento de sistemas e *softwares* de grande porte usando a prototipagem como mecanismo de redução de riscos (Bruce Maxim, 2014).

No modelo em espiral, os requisitos ainda possuem um detalhamento inicial muito forte. Uma passagem no ciclo em espiral destina-se a compreender os requisitos e realizar algumas validações desses antes de começar o desenvolvimento propriamente dito. Posteriormente, o modelo passa por outro ciclo em espiral maior, destinada a desenvolver a solução em etapas sequenciais, de *design*, codificação, implantação e teste (Strode & Strode, 2016).

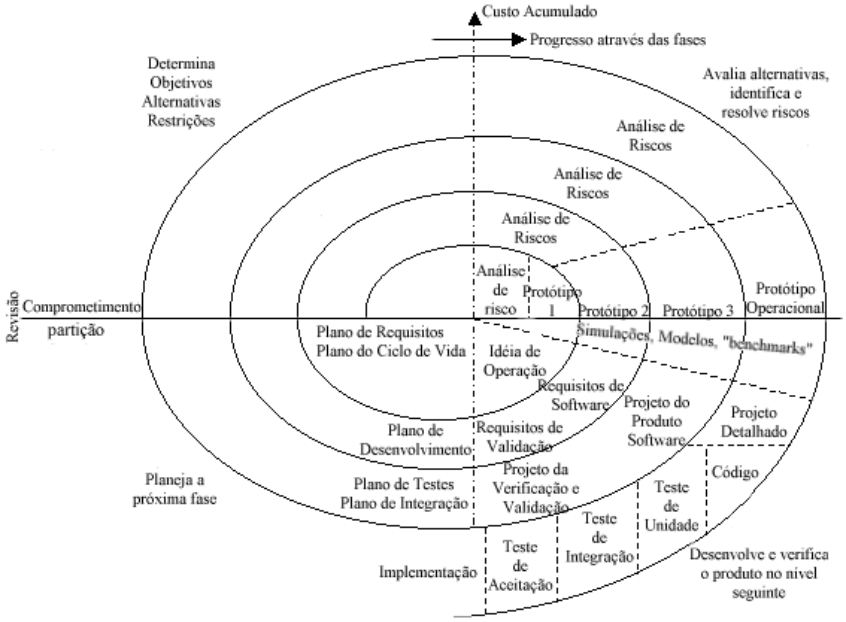
No espiral que representa o modelo, a volta mais interna pode preocupar-se com a viabilidade do sistema; o ciclo seguinte, com definição de requisitos; o seguinte, com o projeto do sistema, e assim por diante (Sommerville, 2003).

Os modelos em espirais são interativos. Inicialmente é realizado um levantamento das necessidades do cliente, e através de diversas interações são implementados pequenos conjuntos de requisitos, proporcionando ao cliente acompanhar a evolução do projeto. Gasparotto e Souza definem seis etapas para os modelos em espirais, são eles (K. P. de Souza & Gasparotto, 2013)(J. Souza, 2018):

1 Planejamento - determina os objetivos específicos para a fase do projeto.
2 Análise de risco - identifica e analisa informações para reduzir e resolver riscos.
3 Realização de protótipos - afirmando-se como uma das maiores desvantagens deste modelo uma vez que o resultado pode ser diferente e o cliente muitas vezes, julga ter já um produto e afinal tem um protótipo.
4 Desenvolvimento do produto.
5 Testes e disponibilização do produto.
6 Entrega e avaliação feita pelo cliente.

Na Figura 4 criada por Boehm em 1988, conseguimos visualizar todas as etapas do processo em espiral.

Figura 4: Modelo em espiral



Fonte: (Barry Boehm, 2003)

2.2.4 Guia PMBOK®

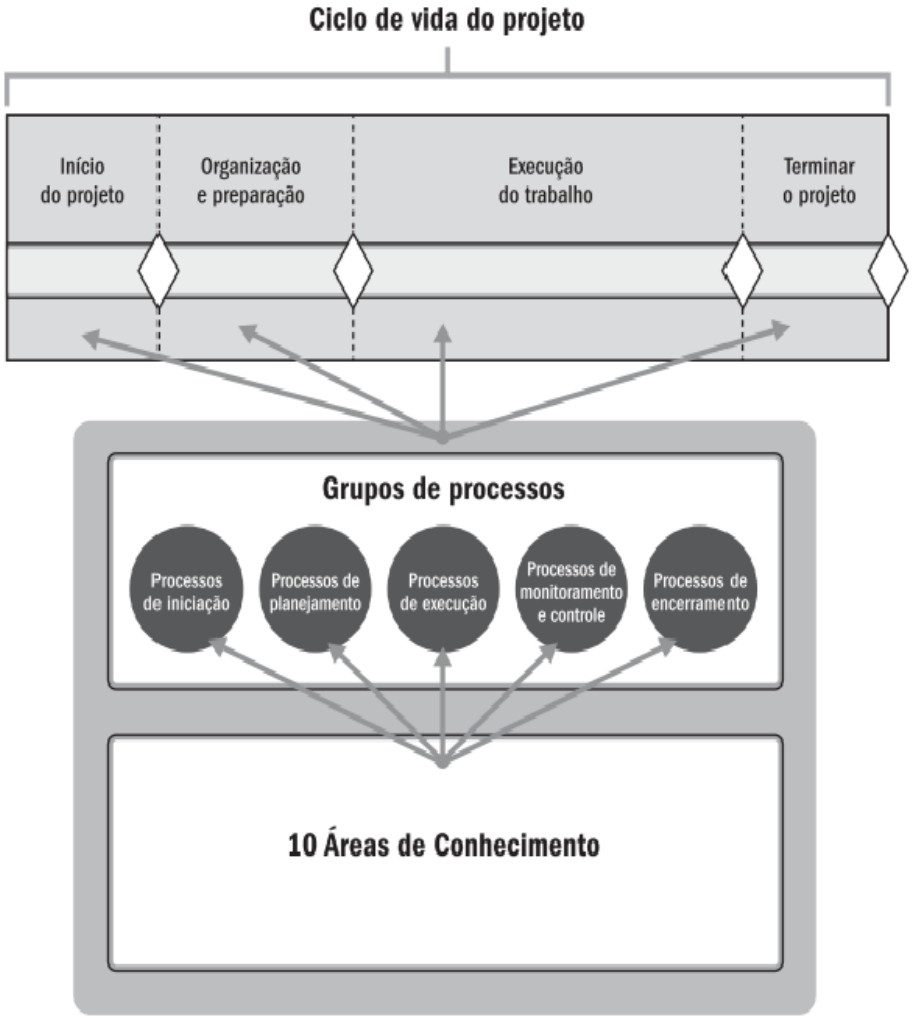
O Guia PMBOK® (*Project Management Body of Knowledge* - Conjunto de Conhecimentos em Gerenciamento de Projetos) foi definido e desenvolvido pelo PMI (*Project Management Institute*), um instituto americano focado em trabalhar ações da área de gerenciamento de projetos.

O modelo clássico orientado pelo Guia PMBOK® trabalha com vários grupos de processos que abrangem todas as fases de um projeto, escrito de forma genérica, descrevendo as boas práticas de gerenciamento sem determinar a área de conhecimento para sua aplicação (*Guia PMBOK®*, 2017).

O Guia PMBOK® utiliza cinco grupos de processos, são eles: Iniciação, Planejamento, Execução, Monitoramento e Controle e Encerramento (*Guia PMBOK®*, 2017). Estes processos são usados para categorizar as operações de gerenciamento de projetos necessárias para administrar uma empresa ou supervisionar um projeto.

Na Figura 5 conseguimos visualizar os cinco grupos de processos e os passos para o ciclo de vida de um projeto orientado pelo Guia PMBOK®.

Figura 5: Ciclo de vida de um projeto e grupos de processos



Fonte: (*Guia PMBOK®*, 2017)

Todo o processo de desenvolvimento é orientado pelo gerente do projeto. Segundo Bomfin, Nunes e Hastenreiter, o gerenciamento de projetos é, normalmente, o campo de responsabilidade de um gerente de projeto individual. Esse indivíduo raramente participa nas atividades que produzem o resultado, mas se esforça para manter o progresso e a interação produtiva das várias partes, reduzindo o risco geral de fracasso (Bomfin, Nunes, & Hastenreiter, 2012).

A organização dos processos em grupos e áreas de conhecimento é o que caracteriza o Guia PMBOK®, tendo sido escrito a partir das experiências positivas e negativas dos voluntários autores do PMI. É considerado pela literatura como um modelo clássico, tradicional e burocrático de gerenciamento de projetos (Vargas, 2016).

Existem dez áreas de conhecimentos em gestão de projetos. Para Camargo elas caracterizam os principais aspectos envolvidos em um projeto e no seu gerenciamento, são elas (Camargo, 2019):

1 Gerenciamento de integração de projetos

A integração é referente ao processo de combinar ou unir as várias partes móveis de qualquer projeto. Assim, será mais fácil trabalhar em direção a um objetivo comum.

Integração significa unificação, consolidação e articulação. O gerenciamento da integração requer que sejam feitas escolhas sobre alocação de recursos, concessões entre objetivos e alternativas conflitantes, além do gerenciamento de dependências mútuas entre áreas de conhecimento e processos. Aqui se destaca a importância que tem a comunicação com os participantes do projeto.

Gerenciar a integração do projeto é garantir que os componentes do projeto precisam trabalhar juntos e é papel do gerente de projetos fazer que isso aconteça. Exige habilidades em negociação e gerenciamento de conflitos de interesses. Também exige habilidades gerais de gerenciamento, boa comunicação, organização, familiaridade técnica com o produto, etc.

2 Gerenciamento do âmbito do projeto

O Gerenciamento do âmbito inclui processos necessários para assegurar que o projeto inclui todo o trabalho necessário e somente o trabalho necessário para concluir o projeto com sucesso. O objetivo é definir e controlar o que faz parte do projeto, assim, evita-se que o âmbito do projeto se expanda conforme o tempo passa.

3 Gerenciamento do cronograma

Também chamado de gerenciamento de tempo em edições anteriores à sexta edição do Guia PMBOK®, o cronograma visa manter uma sequência de eventos precisa e atualizada. Dessa forma, busca-se o cumprimento de prazos e responsabilidade, porém, podem ocorrer ajustes dos prazos, se necessário.

O gerenciamento do cronograma inclui processos necessários para estimar as tarefas, seus recursos e durações, de modo a gerenciar o projeto para o término pontual.

4 Gerenciamento de custos

A gestão de custos inclui processos envolvidos em estimativas, orçamentos e controle dos custos, de modo que o projeto possa ser terminado dentro do orçamento aprovado.

5 Gerenciamento da qualidade

O gerenciamento da qualidade inclui processos e atividades da organização executora que determinam as políticas de qualidade, objetivos, requisitos e responsabilidades de modo que o projeto satisfaça às necessidades para as quais foi empreendido. Implementa o sistema de gerenciamento da qualidade e atividades para a melhoria contínua dos processos.

6 Gerenciamento de recursos do projeto

Também chamada de recursos humanos em algumas edições, ela inclui processos que organizam e gerenciam a equipe do projeto. Faz parte desta área do conhecimento descrever as necessidades de pessoal e suas respectivas capacidades e habilidades.

7 Gerenciamento de comunicações

O gerenciamento das comunicações inclui todos os processos necessários para assegurar que as informações do projeto sejam geradas, coletadas, distribuídas, armazenadas, recuperadas e organizadas de maneira oportuna e apropriada.

8 Gerenciamento de riscos

O gerenciamento de riscos inclui processos de planejamento, identificação, análise, estabelecendo também um plano de resposta para tratar de problemas que possam surgir, bem como o monitoramento e controle de riscos de um projeto. Os objetivos do gerenciamento de riscos são aumentar a probabilidade e o impacto dos eventos positivos e reduzir a probabilidade e o impacto dos eventos negativos no projeto.

9 Gerenciamento de aquisições do projeto

O gerenciamento das aquisições do projeto inclui os processos necessários para comprar ou adquirir produtos, serviços ou resultados externos ao projeto e abrange o gerenciamento de contratos. A organização pode ser tanto o comprador como vendedor dos produtos, serviços ou resultados de um projeto. Na ótica do PMI, abordamos o gerenciamento das aquisições do ponto de vista do comprador.

10 Gestão de partes interessadas do projeto

O Gerenciamento das partes interessadas ou stakeholders inclui processos de identificação, planejamento, engajamento e gerenciamento das partes interessadas. Os objetivos do gerenciamento das partes interessadas é aumentar o suporte e comprometimento dos stakeholders ao projeto. Para isso, são utilizadas estratégias para identificar e gerenciar as expectativas das partes interessadas.

Na Figura 6 conseguimos visualizar as dez áreas de conhecimento orientado pelo Guia PMBOK®.

Figura 6: Dez áreas de conhecimento segundo Guia PMBOK®

Áreas de conhecimento	Grupos de processos de gerenciamento de projetos				
	Grupo de processos de iniciação	Grupo de processos de planejamento	Grupo de processos de execução	Grupo de processos de monitoramento e controle	Grupo de processos de encerramento
4. Gerenciamento da integração do projeto	4.1 Desenvolver o Termo de Abertura do Projeto	4.2 Desenvolver o Plano de Gerenciamento do Projeto	4.3 Orientar e Gerenciar o Trabalho do Projeto 4.4 Gerenciar o Conhecimento do Projeto	4.5 Monitorar e Controlar o Trabalho do Projeto 4.6 Realizar o Controle Integrado de Mudanças	4.7 Encerrar o Projeto ou Fase
5. Gerenciamento do escopo do projeto		5.1 Planejar o Gerenciamento do Escopo 5.2 Coletar os Requisitos 5.3 Definir o Escopo 5.4 Criar a EAP		5.5 Validar o Escopo 5.6 Controlar o Escopo	
6. Gerenciamento do cronograma do projeto		6.1 Planejar o Gerenciamento do Cronograma 6.2 Definir as Atividades 6.3 Sequenciar as Atividades 6.4 Estimar as Durações das Atividades 6.5 Desenvolver o Cronograma		6.6 Controlar o Cronograma	
7. Gerenciamento dos custos do projeto		7.1 Planejar o Gerenciamento dos Custos 7.2 Estimar os Custos 7.3 Determinar o Orçamento		7.4 Controlar os Custos	
8. Gerenciamento da qualidade do projeto		8.1 Planejar o Gerenciamento da Qualidade	8.2 Gerenciar a Qualidade	8.3 Controlar a Qualidade	
9. Gerenciamento dos recursos do projeto		9.1 Planejar o Gerenciamento dos Recursos 9.2 Estimar os Recursos das Atividades	9.3 Adquirir Recursos 9.4 Desenvolver a Equipe 9.5 Gerenciar a Equipe	9.6 Controlar os Recursos	
10. Gerenciamento das comunicações do projeto		10.1 Planejar o Gerenciamento das Comunicações	10.2 Gerenciar as Comunicações	10.3 Monitorar as Comunicações	
11. Gerenciamento dos riscos do projeto		11.1 Planejar o Gerenciamento dos Riscos 11.2 Identificar os Riscos 11.3 Realizar a Análise Qualitativa dos Riscos 11.4 Realizar a Análise Quantitativa dos Riscos 11.5 Planejar as Respostas aos Riscos	11.6 Implementar Respostas aos Riscos	11.7 Monitorar os Riscos	
12. Gerenciamento das aquisições do projeto		12.1 Planejar o Gerenciamento das Aquisições	12.2 Conduzir as Aquisições	12.3 Controlar as Aquisições	
13. Gerenciamento das partes interessadas do projeto	13.1 Identificar as Partes Interessadas	13.2 Planejar o Engajamento das Partes Interessadas	13.3 Gerenciar o Engajamento das Partes Interessadas	13.4 Monitorar o Engajamento das Partes Interessadas	

Fonte: (Guia PMBOK®, 2017)

Com isto, podemos concluir que o Guia PMBOK® se propõe a ser um manual de boas práticas a ser executado em conjunto com os modelos de gestão de projetos, não descreve “como fazer” projetos e sim “o que deve ser feito” através de processos que abrangem diversas áreas de conhecimento (Vargas, 2016).

No próximo capítulo pretende-se abordar algumas falhas nos modelos clássicos mais mencionados na literatura.

2.3 Falhas nos modelos clássicos

As abordagens clássicas seguem um conjunto de processos pré-determinados, em que é produzido documentações. O sucesso alcançado com os projetos com estas abordagens, depende do conhecimento de todos os requisitos no início, o que não é simples nos dias de hoje. Um problema apontado reside no facto dos requisitos serem todos fechados antes do início do desenvolvimento, sendo difícil a implantação de mudanças ao longo do ciclo de vida do projeto (Cockburn, 2003) (Juliani, Juliani, Alves Bello, & De Souza, 2012) (J. Souza, 2018).

Taroco & Werner afirmam que o ciclo de vida dos modelos clássicos são pesados e requer detalhamento antes de iniciar o desenvolvimento de projeto, isso leva a empresa a utilizar mais tempo detalhando o *software* do que desenvolvendo, e que o maior objetivo do projeto é o seu desenvolvimento e não o seu planejamento (Taroco & Werner[1], 2015).

Soares defende que os modelos clássicos são pesados e que devem ser aplicados apenas em situações em que os requisitos do *software* são estáveis e requisitos futuros são previsíveis. Estas situações são difíceis de serem atingidas, uma vez que os requisitos para o desenvolvimento de um *software* são mutáveis (M. D. S. Soares, 2004).

Segundo Adriele, a realidade dos projetos de desenvolvimento de *software* é que os clientes e outras partes interessadas normalmente não têm certeza sobre o que desejam no início dos projetos e que quando uma etapa de um projeto de *software* foi inteiramente concluída, a opção de voltar atrás e refazer parte do trabalho como é feito nos modelos clássicos implica em custos elevados (Portugal, 2020).

Segundo Cohn, algumas das razões para os modelos clássicos falharem são (Rasnacis & Berzisa, 2016):

- As atividades não terminam antes do tempo planejado, ou seja, o responsável pela execução tende a estender o tempo inicialmente estipulado para cada atividade. Em algumas empresas, acabar antes do planejado pode levar o gestor de projeto a ponderar que fez um mau planejamento ou que todas as atividades irão terminar antes.
- O atraso de uma atividade implica que as suas sucessoras possam iniciar mais tarde, isto se a sua folga for inferior ao atraso. Mesmo que uma tarefa termine mais cedo, mas o seu início dependa de outras, só pode iniciar quando todas terminarem.
- As atividades não são desenvolvidas por prioridades, uma vez que o cliente só terá acesso ao projeto quando este estiver terminado. Um problema frequente é o momento do fechamento de um projeto, ou seja, quando não foi possível realizar todo o âmbito, a tendência é ajustá-lo ao âmbito realizado. Estas situações

determinam em muitos casos, a não serem entregues as funcionalidades mais importantes para o cliente.

- A gestão clássica ignora a incerteza ao assumir que os requisitos, inicialmente definidos, levam a uma satisfação total por parte dos clientes.
- A atribuição de várias tarefas, em simultâneo, a um recurso tende a diminuir a produtividade, uma vez que o foco do colaborador tende a dispersar-se por diversas atividades.

No próximo capítulo pretende-se abordar o que são os modelos ágeis na gestão de projetos e como eles funcionam.

2.4 Modelos ágeis

Os modelos ágeis de gestão de projetos, são modelos muito utilizados nos projetos de desenvolvimento de *software* com pouca documentação, transparência e muito dinamismo, uma vez que os projetos são entregues em etapas aumentando assim a adaptabilidade às mudanças e evitando surpresas no final dos projetos.

Segundo Tomás, os modelos ágeis estão cada vez mais a ser discutidos pelas empresas, sendo que muitas têm receio de adotar estes modelos devido às dificuldades na reestruturação organizacional (Tomás, 2009).

Para Franzen e Lutz, os modelos ágeis surgiram para oferecer dinamicidade aos projetos, propondo novas técnicas de execução das etapas de trabalho, seguindo fluxos alternativos e flexíveis (Franzen & Lutz, 2018).

Nos modelos ágeis a fase de planejamento inicial é minimizada, de modo que os desenvolvedores se concentram em entregar o produto ao fim de cada interação, ao invés de traçar diretrizes e planejamentos para o projeto como um todo.

Segundo Moniruzzaman, Hossain, Gautam e Kumar, o termo *agile* refere-se a fácil e rápida movimentação, encontro rápido e resposta rápida. Um processo ágil ajuda a dividir o trabalho em vários números de subtarefas para adaptar planos utilizáveis que podem ser usados para reavaliações frequentes (Moniruzzaman, Hossain, Gautam, & Kumar, 2013).

Estes modelos exigem maior participação do cliente e de todas as pessoas envolvidas no projeto, garantindo um nível de assertividade maior tanto para a empresa quanto para o cliente uma vez que em cada etapa do projeto é feito o levantamento dos requisitos, desenvolvimento de código, testes e documentação com o intuito de existir no final de cada iteração uma entrega ao cliente, que inclua um conjunto de novas funcionalidades e uma nova versão de *software*. Após essa entrega há

um novo processo de comunicação com o cliente e então são definidas quais deverão ser as novas entregas.

Segundo Tomás, nos modelos ágeis as pessoas têm um papel fundamental no desenvolvimento dos projetos, sendo essencial que exista uma boa comunicação e que cada indivíduo se preocupe com a qualidade dos projetos. É valorizada a entrega de um produto funcional e adequado ao que o cliente realmente deseja (Tomás, 2009).

Para Trigo e Barreto, o modelo ágil permite aos clientes visualizarem o *software* com todas as funcionalidades prontas em cada etapa do projeto devido as entregas constantes, o que pode não ocorrer no modelo clássico pela distância natural que ocorre entre o cliente e a equipe de desenvolvimento, no final do projeto o cliente pode receber um *software* com funções adicionais não solicitadas e não agregando valores aos negócios (Trigo & Barreto, 2019).

Os modelos ágeis têm despertado um grande interesse na comunidade de desenvolvimento de *software*. Acredita-se que, devido à esta demanda, uma considerável quantidade de modelos, apresentando características ágeis, têm surgido nos últimos anos.

Na Figura 7 podemos verificar os modelos ágeis por data de publicação.

Figura 7: Modelos ágeis por data de publicação

	Método Agile		Fonte Primária Jornal	
			artigo	Livro
1	método dinâmico de Desenvolvimento de Sistemas	DSDM DSDM_	Consortium (1995)	Stapleton (1997)
2	métodos de cristal	Cristal	Cockburn (1998)	Cockburn (2002)
3	RUP (configurado)	dX	Martin (1998)	
4	Programação extrema	XP	Beck (1999)	Beck (2000)
5	Adaptive Desenvolvimento de Software	ASD		Highsmith (2000)
6	Scrum	Scrum Beedle,	Devos, Sharon, Schwaber, e Sutherland (1999)	Schwaber & Beedle (2002)
7	Programação pragmática	PP		Hunt e Thomas (2000)
8	Desenvolvimento velocidade gratuita	ISD	Cusumano e Yoffie (1999) Baskerville & Pries-Heje (2001)	
9	Modelagem ágil	SOU		Ambler (2002)
10	Feature Driven Development	FDD		Palmer & Felsing (2002)
11	Open Source Software Development	OSS	Sharma, Sugumaran, & Rajagopalan (2002)	
12	Desenvolvimento magra	LD	Charette (2002)	Poppendiek & Poppendiek (2003)

Fonte: (Strode & Hunt, 2006)

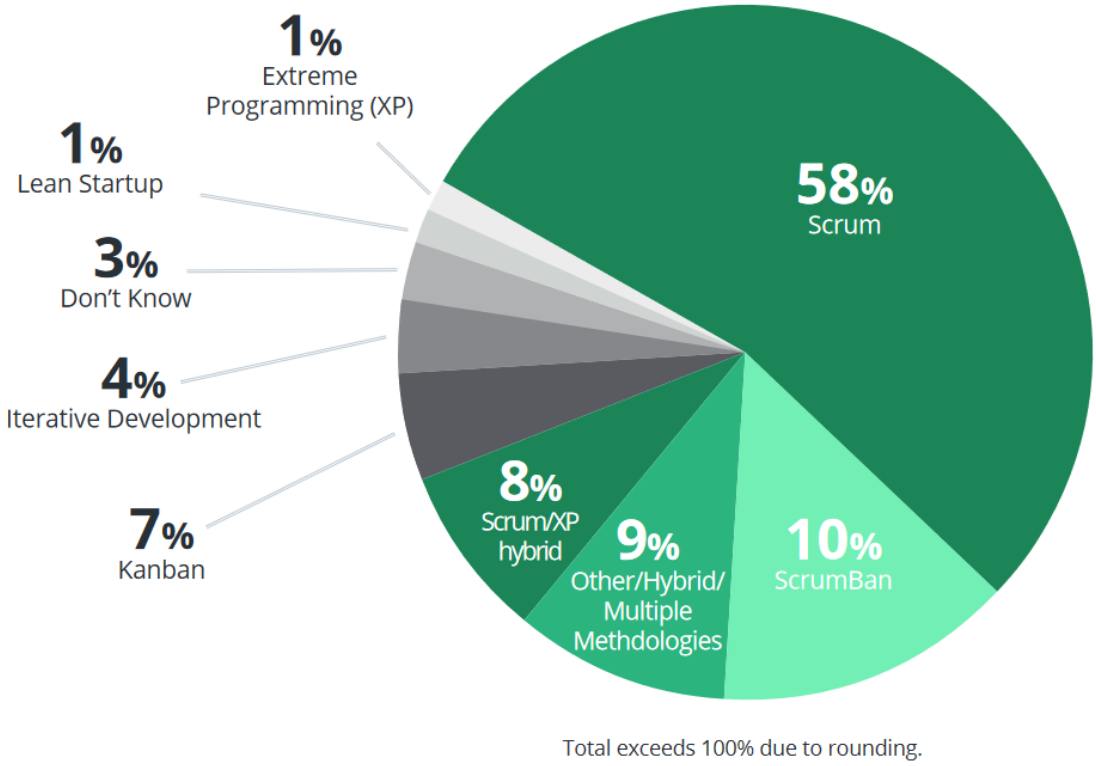
Segundo Sharma e Aggarwal, muitos estudos foram realizados sobre esses modelos ágeis, alguns dos modelos mais mencionados nas literaturas são: *XP-Extreme Programming*, *FDD-Feature Driven Development*, Kanban, Scrum e *ASD-Adaptive Software Development* (Sharma & Aggarwal, 2013).

Em 2020, a 14ª pesquisa anual ágil, deixa claro que o desenvolvimento ágil de *software* se tornou cada vez mais popular na última década. A participação na pesquisa cresceu mais de três vezes nos últimos dez anos. O número de grandes empresas que estão adotando o modelo ágil continua a aumentar a cada ano. Mais de 24% dos entrevistados trabalhavam para empresas com mais de 20.000

funcionários. Esta pesquisa levantou os modelos ágeis mais utilizados pelas empresas em todo mundo em 2020 (StateOfAgile, 2020).

Na Figura 8 conseguimos visualizar que o Scrum é o modelo ágil mais utilizado pelas empresas com 58% dos entrevistados, com pelo menos 75% dos entrevistados praticando Scrum ou um modelo híbrido que inclui Scrum.

Figura 8: Modelos ágeis e sua utilização



Fonte: (StateOfAgile, 2020)

Nos capítulos seguintes pretende-se abordar alguns destes modelos, pormenorizando o modelo Scrum por ser o modelo mais utilizado segundo a literatura.

2.4.1 XP-Extreme Programming

Segundo Soares, o *XP-Extreme Programming* é o modelo ágil mais conhecido (M. D. S. Soares, 2004).

O XP usa uma abordagem orientada a objetos como seu paradigma de desenho. Para Tomás, o processo é composto por quatro atividades, são elas (Tomás, 2009):

- Planejamento: consiste em decidir o que é necessário ser feito e o que pode ser adiado no projeto.
- Projeto: deve ser o mais simples possível e satisfazer os requisitos atuais, sem a preocupação de requisitos futuros.
- Codificação: a implementação do código é feita em dupla, ou seja, dois desenvolvedores trabalham em um único computador.
- Teste: focaliza a validação do projeto durante todo o processo de desenvolvimento, que são repetidas iteração a iteração.

Para Soares, o XP é conduzido por quatro valores: comunicação, simplicidade, feedback e coragem. Estas são as principais diferenças deste modelo em relação aos outros modelos ágeis (M. D. S. Soares, 2004).

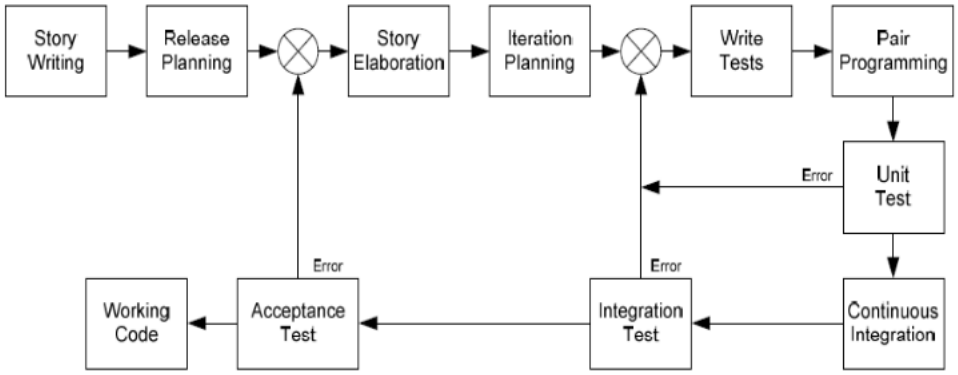
Soares afirma que o modelo XP é diferenciado das demais modelos ágeis por conter um feedback constante, uma abordagem incremental e pela comunicação entre as pessoas ser encorajada. Algumas regras deste modelo podem causar conflitos e outras não fazem sentido se aplicadas isoladamente, sendo assim, revoluciona o desenvolvimento de *software* (M. Soares, 2004).

No XP, os gerentes, desenvolvedores e clientes, são todos parte de uma mesma equipe. Ao longo do projeto os feedbacks são constantes, tanto para o cliente quanto para a equipe. Os desenvolvedores recebem retorno constante trabalhando em pares e testando o código conforme é escrito. Os gerentes recebem o reporte do progresso e dos obstáculos nas reuniões diárias. Os clientes recebem feedback sobre o progresso com os resultados dos testes de aceitação e demonstrações a cada iteração (Franzen & Lutz, 2018).

No XP o código do projeto pertence a todos os membros da equipe. Isto significa que qualquer pessoa que percebe que pode adicionar valor a um código visto que o código é sempre comentado e feito em conjunto com outras pessoas (Qasaimeh, Mehrfard, & Hamou-Lhadj, 2008).

Na Figura 9 conseguimos visualizar e entender melhor as fases do *XP-Extreme Programming*.

Figura 9: Fases do modelo XP-*Extreme Programming*



Fonte: (Qasaimeh et al., 2008)

2.4.2 Kanban

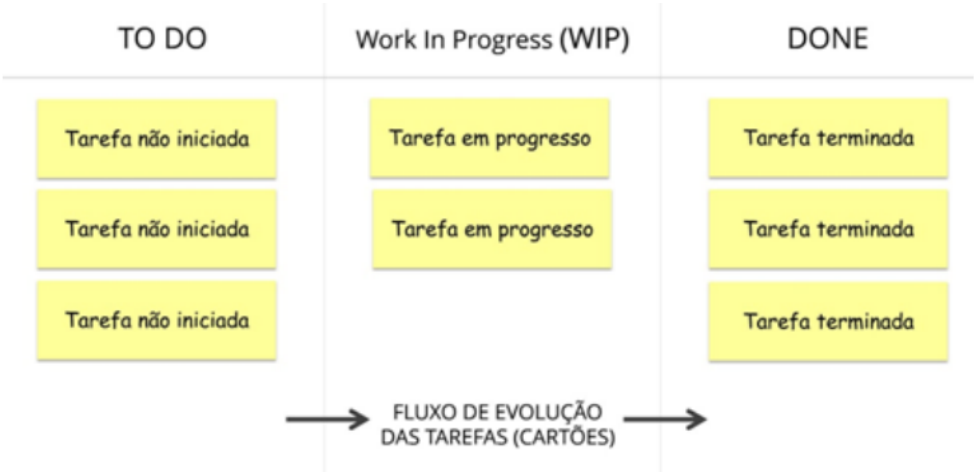
O Kanban surgiu no Japão e significa cartão ou sinalização (Bruno, 2019).

Uma das formas mais frequentes de Kanban utilizadas desde o momento de sua criação, segundo Taiichi Ohno, é a de um pedaço de papel dentro de um envelope de vinil retangular, contendo informações essenciais para o funcionamento correto do sistema produtivo, com a finalidade de atingir o Just-in-time (Ohno, 1997).

O sistema *Just-in-time* (JIT) como é conhecido o modelo Kanban, é um subsistema do Sistema de Produção Toyota (TPS) que foi desenvolvido pela *Toyota Motors Company* durante os anos 1950-1960 com o objetivo de eliminar os elementos desnecessários referentes à produção, com consequente redução de custos. A filosofia do Kanban é que partes e materiais devem ser fornecidos exatamente quando são necessários no processo de produção fabril, a fim de eliminar estoques, reduzir custos e aumentar a produtividade (Thielmann & Rodrigues, Gustavo Alves, Raphael Lima, 2015) (Ohno, 1997).

Esse modelo consiste em utilizar um quadro Kanban, que pode ser feito em um quadro branco, mural, parede ou folha nos quais são colados cartões que demonstram o fluxo do trabalho. À medida que o trabalho vai sendo realizado, os cartões avançam dentre os estágios pré-estabelecidos: a fazer, em andamento e finalizado. Na Figura 10 conseguimos visualizar esta idéia.

Figura 10: Quadro Kanban



Fonte: (Bruno, 2019)

Também existem diversos *softwares* que simulam o quadro Kanban e são muito utilizados. Dentro das empresas nas quais o autor atuou, podemos citar o *software* Trello e o Zoho (https://trello.com)("https://www.zoho.com," n.d.).

As principais vantagens deste modelo é que ela permite que se visualize rapidamente a situação atual de trabalho e a sua transparência, que melhora a comunicação e integração das equipes. Um elemento importante do Kanban é a priorização das tarefas, de modo que o trabalho que entregar maior valor fica posicionado na parte superior do quadro. Além disso, esse modelo evidencia a importância da adaptação para o atingimento da melhoria contínua (Bruno, 2019).

Segundo Souza, os princípios e práticas do Kanban são (J. Souza, 2018) (https://kanbanize.com/pt/recursos-kanban/primeiros-passos/o-que-e-kanban):

- Começar com o que existe - o modelo Kanban é iniciado com os processos já existentes na empresa e estimula a mudança contínua, incremental e evolutiva do sistema.
- Concordar com a mudança contínua - tem de se concordar com a mudança contínua e incremental.
- Respeitar o processo atual - o que funciona bem deve-se manter. Respeitar o que se encontra a correr bem, como papéis e responsabilidades no local de trabalho, gera um maior consenso no apoio às mudanças necessárias.
- Liderança - são incentivados atos de liderança em todos os níveis da organização, uma vez que as melhorias são esperadas em qualquer nível de uma organização.
- Ver o trabalho - ao ser criado um modelo visual de todo o trabalho e seu fluxo, podemos observar o fluxo em movimento através do sistema Kanban. Tornar o trabalho visível junto

com todos os constrangimentos, instantaneamente, contribui a uma maior compreensão e colaboração.

- *Work-in-Progress* (WIP) - ao ser limitado o número máximo de pedidos por estado do fluxo, reduz-se o tempo que um item demora a percorrê-lo e auxilia também na priorização dos pedidos, evitando problemas de trocas.
- Gerir o fluxo - é necessário monitorar como os pedidos percorrem o fluxo. Deve ser rápido e suave, ou seja, criar rapidamente valor para o negócio, minimizar o risco e custos de atraso.
- Regras bem definidas - todas as normas de trabalho devem ser bem compreendidas no seio da equipa, caso contrário é impossível haver entendimentos ou processos de melhoria.

2.4.3 Desenvolvimento Guiado por Funcionalidade (FDD-Feature Driven Development)

O FDD-*Feature Driven Development* é um *software* ágil desenvolvido por Jeff De Luca e Peter Code, dois profissionais de gerenciamento de projetos de desenvolvimento de *software* (Hislop et al., 2002).

A abordagem FDD concentra-se nos recursos de *software* do sistema como o principal condutor do processo de desenvolvimento. Ele difere significativamente de outros processos ágeis, colocando uma forte ênfase no planejamento e no *design* antecipado (Qasaimeh et al., 2008).

Conforme mostrado na Figura 11, Qasaimeh analisa a figura da seguinte forma (Qasaimeh et al., 2008):

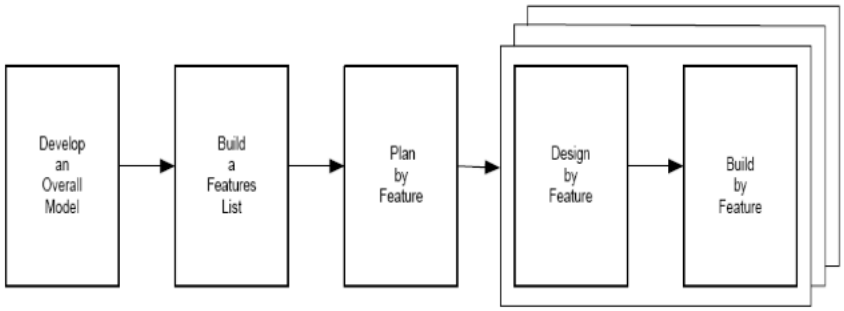
A primeira etapa do processo de FDD é construir um modelo detalhado do sistema a ser desenvolvido, que captura as premissas e requisitos das partes interessadas. Uma vez que o modelo de domínio é construído, os membros da equipe imprimem uma lista dos recursos do sistema. Cada recurso deve ser desenvolvido em algumas horas ou dias, mas não mais do que duas semanas.

Usando o FDD, as equipes de desenvolvimento são formadas especificamente para projetar e implementar um determinado recurso. O trabalho é geralmente executado em paralelo em vários recursos. As equipes são dissolvidas assim que o recurso é concluído e verificado. Cada equipe é liderada por um proprietário do recurso que é responsável pelo segmento de código que implementa o recurso.

O processo FDD utiliza diretrizes de inspeção rigorosas para encontrar defeitos no sistema. Ele também impõe padrões de codificação. Ele também incentiva compilações regulares em um diário ou semanalmente, a fim de adicionar recursos recém-projetados ao sistema de linha de base. Devido ao fato de que os recursos são desenvolvidos em paralelo, é importante ter uma configuração de um sistema de gestão que permite a integração adequada das alterações efetuadas no sistema.

Exclusivo para a abordagem FDD é um mecanismo de rastreamento e papel que avalia o status do projeto com base no número de recursos que foram implementados, bem como o progresso geral do *design*, codificação e testes.

Figura 11: FDD - *Feature Driven Development*



Fonte: (Malik Qasaimeh, 2008)

2.4.4 Scrum

O Scrum foi desenvolvido inicialmente por Jeff Sutherland e por sua equipe no início da década de 1990 com o intuito de atender projetos com prazos apertados e requisitos que mudam frequentemente.

Segundo Soares, o foco do Scrum é encontrar uma forma de trabalho dos membros da equipe para produzir o *software* de forma flexível e em um ambiente em constante mudança (M. D. S. Soares, 2004).

O Scrum é desenvolvido em equipes em que todos trabalham com requisitos pré-estabelecidos para entrega de cada fase que são chamados de *sprints*. No Scrum existem pequenas reuniões diárias de acompanhamento nos quais são discutidos pontos como o que foi feito desde a última reunião e o que precisa ser feito até a próxima.

Para Tomás, os *sprints* consistem em unidades de trabalho que são necessárias para satisfazer um requisito definido, num determinado período de tempo (Tomás, 2009).

O processo de desenvolvimento utilizando Scrum divide o projeto em fases. Em cada fase, uma parte é totalmente desenvolvida, testada e fica pronto para ir para a produção. A equipe não se move para uma nova fase até a fase atual ser concluída (Sharma & Aggarwal, 2013).

Como o Scrum é amplamente citado nos projetos estudados, alguns termos referentes a este modelo são aqui descritos, são eles (J. J. Sutherland (Autor), 2020):

- Definição - *framework* (um processo de engenharia de *software*) estrutural utilizado em projetos de desenvolvimento de produtos complexos.
- Equipe - são auto-organizáveis e multifuncionais. Escolhem a melhor maneira de completar seu trabalho em vez de serem dirigidos por outros de fora da equipe. Fazem parte da equipe:
    - *Product owner* - o dono do produto, responsável por maximizar o valor do produto e do trabalho da equipe do desenvolvimento. É responsável pelo *backlog* do produto.
    - Equipe de desenvolvimento - profissionais responsáveis por entregar uma versão usável para incrementar o produto, ao final de cada *sprint*.
    - Scrum *Master* - pessoa com bastante conhecimento do Scrum, responsável por garantir que o *framework* seja entendido e aplicado.
- Eventos - são acontecimentos rotineiros que devem acontecer ao utilizar-se Scrum.
    - *Sprint* - entrega parcial, que ocorre aproximadamente a cada duas ou quatro semanas e agrega valor ao produto final do projeto.
    - Reunião de planejamento do *Sprint* - reunião em que se planeja o trabalho a ser realizado durante o próximo *sprint*.
    - Reunião diária - reuniões no qual a equipe se reúne diariamente, por 15 minutos, para sincronizar as atividades e criar um plano para as próximas 24 horas.
    - Retrospectiva do *sprint* - ao final de cada *sprint*, é feita uma revisão do que foi entregue, para inspecionar e, se for necessário, ajustar o *backlog* do produto.
- Artefatos - representam o trabalho, e são projetados para maximizar a transparência nos projetos:
    - *Backlog* do produto - lista ordenada e priorizada com tudo o que deve ser entregue no produto.
    - *Backlog* do *sprint* - itens do *backlog* do produto selecionados para a implementação no *sprint* atual.
    - Incremento - todos os itens do *backlog* do produto que foram entregues durante o *sprint*.

Soares afirma que o ciclo de vida do Scrum é baseado em três fases principais, são elas (M. D. S. Soares, 2004):

1 Pré-planejamento - os requisitos são descritos em um documento chamado *backlog*. Posteriormente eles são priorizados e são feitas estimativas de esforço para o desenvolvimento de cada requisito. O planejamento inclui também, entre outras atividades, a definição da equipe de desenvolvimento, as ferramentas a serem usadas, os possíveis riscos do projeto e as necessidades de

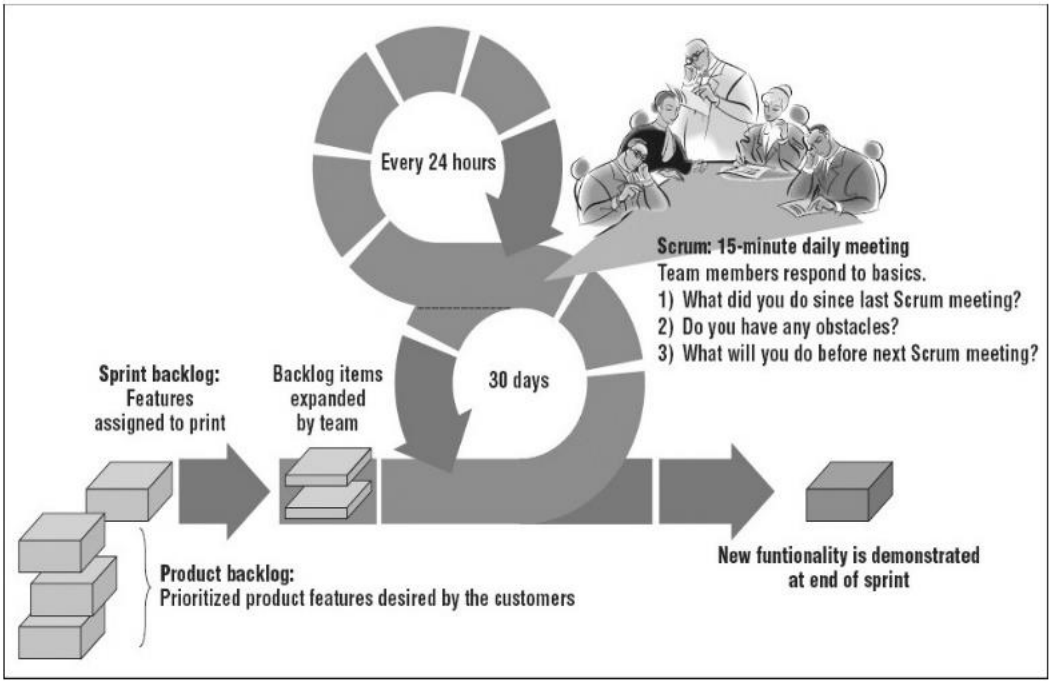
treinamento. Finalmente é proposta uma arquitetura de desenvolvimento. Eventuais alterações nos requisitos descritos no *backlog* são identificadas, assim como seus possíveis riscos.

2 Desenvolvimento - as muitas variáveis técnicas e do ambiente identificadas previamente são observadas e controladas durante o desenvolvimento. Ao invés de considerar essas variáveis apenas no início do projeto, como no caso dos modelos clássicos, no Scrum o controle é feito continuamente, o que aumenta a flexibilidade para acompanhar as mudanças. Nesta fase o *software* é desenvolvido em ciclos (*sprints*) em que novas funcionalidades são adicionadas. Cada um desses ciclos é desenvolvido de forma tradicional, ou seja, primeiramente faz-se a análise, em seguida o projeto, implementação e testes. Cada um desses ciclos é planejado para durar de uma semana a um mês.

3 Pós-planejamento - após a fase de desenvolvimento são feitas reuniões para analisar o progresso do projeto e demonstrar o *software* atual para os clientes. Nesta fase são feitas as etapas de integração, testes finais e documentação.

Na Figura 12 conseguimos visualizar o ciclo de vida do Scrum e como ele funciona.

Figura 12: Ciclo de vida do Scrum



Fonte: (Moniruzzaman et al., 2013)

No próximo capítulo iremos abordar de forma resumida outros modelos ágeis de gestão de projetos.

2.4.5 Outros modelos ágeis de gestão de projetos

Este capítulo pretende-se abordar de forma resumida alguns modelos ágeis mencionados na literatura. Porém, com pouco uso atualmente.

2.4.5.1 DSDM - Método de desenvolvimento de sistema dinâmico

O DSDM foi desenvolvido no Reino Unido em meados de 1990. A ideia fundamental por trás do DSDM é fixar o tempo e os recursos e, em seguida, ajustar a quantidade de funcionalidade de acordo, em vez de fixar a quantidade de funcionalidade em um produto e, em seguida, ajustar o tempo e os recursos para alcançar essa funcionalidade (Mahdi JAVANMARD, 2015).

Segundo Stoica, O DSDM divide os projetos em 3 fases: pré-projeto, ciclo de vida do projeto e pós-projeto (STOICA et al., 2013).

O DSDM é baseado em nove princípios. Stoica e Hislop o definem como (STOICA et al., 2013)(Hislop et al., 2002):

1 Envolvimento dos usuários - os usuários devem estar ativamente envolvidos em todo o processo de desenvolvimento.

2 Capacitação da equipe do projeto - as equipes (incluindo usuários e desenvolvedores) devem ter poderes para tomar decisões sem a aprovação explícita da administração superior.

3 Necessidades atuais do negócio - a entrega frequente de produtos tem maior prioridade.

4 Entregas - as entregas são avaliadas principalmente no que diz respeito ao negócio, para fins comerciais.

5 Iterativo e desenvolvimento incremental - iterações rápidas e entrega incremental são fundamentais para convergir para soluções de negócios aceitáveis.

6 Mudanças - retrocesso ou reconstruir versões anteriores deve ser possível.

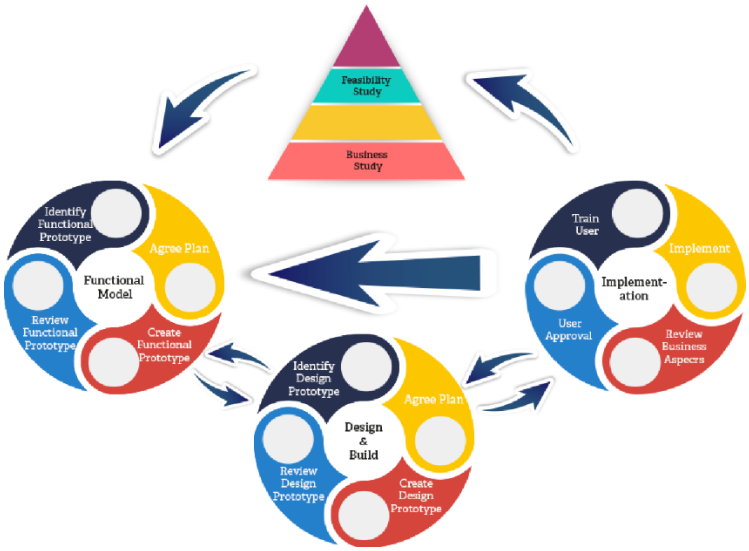
7. Objetivo final - o objetivo final é estabelecido antes do início do projeto. Requisitos de alto nível são definidos cedo para permitir uma investigação detalhada de suas consequências.

8. Teste - o teste é feito em todo o processo de desenvolvimento.

9. Comunicação eficiente - a colaboração entre todas as partes interessadas é a chave para o sucesso.

Na Figura 13 conseguimos visualizar as fases e estágios de um fluxo de processo do modelo DSDM.

Figura 13: Fases e estágios de um fluxo de processo do Modelo DSDM



Fonte: (Coimbra, 2020)

2.4.5.2 ASD – Adpative Software Development

O ASD é um modelo ágil proposta por Jym Highsmith e focou-se na colaboração humana e na auto-organização. Este modelo adapta-se a ambientes que existem mudanças constantes e pouco planeamento. Por isso, o ciclo de desenvolvimento assenta em três fases, sendo elas especulação, colaboração e aprendizagem.

Qasaimeh as definem como (Qasaimeh et al., 2008):

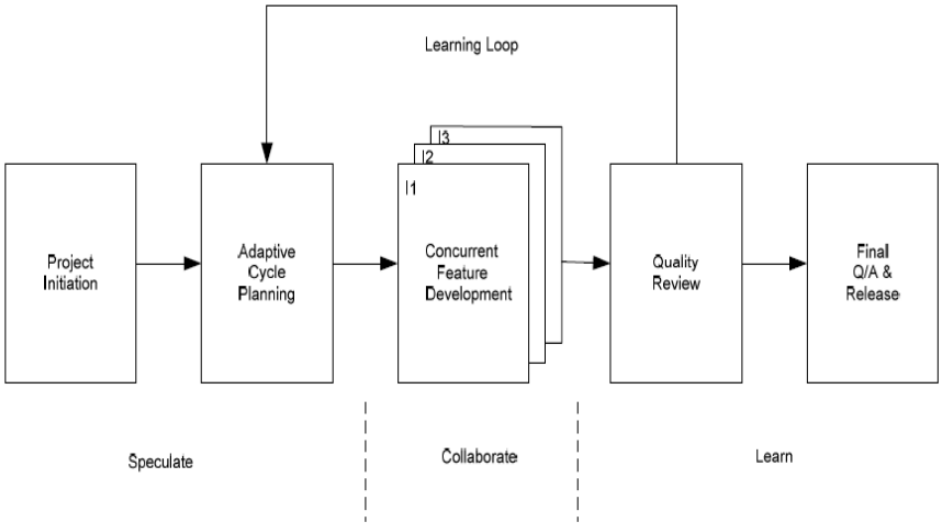
1 Especulação - o que alcançar em cada iteração é determinada.

2 Colaboração - se destaca a importância do trabalho em equipe, compartilhando o conhecimento entre os desenvolvedores de *software*.

3 Aprendizagem - é realizada após cada iteração, a fim de melhorar a experiência do desenvolvedor, bem como melhorar a qualidade do trabalho.

Na Figura 14 conseguimos visualizar as principais fases do processo ASD segundo Qasaimeh (Qasaimeh et al., 2008).

Figura 14: Modelo ASD



Fonte: (Qasaimeh et al., 2008)

2.4.5.3 Crystal

O modelo *Crystal* concentra-se na comunicação entre pequenas equipes que desenvolvem *software* não crítico.

Stoica afirma que o desenvolvimento do *Crystal* possui sete características: entrega frequente, melhora reflexiva, osmótica comunicação, segurança pessoal, concentração, fácil acesso a usuários experientes e requisitos para ambiente técnico (STOICA et al., 2013).

O modelo *Crystal* coloca ênfase em um conjunto de padrões de política que governam a maneira como o projeto é gerenciado. Esses padrões são comuns entre todos os modelos de *Crystal* e incluem entrega incremental de lançamentos, rastreamento de progresso, envolvimento direto do usuário direto, etc (Qasaimeh et al., 2008).

Na Figura 15 conseguimos visualizar o funcionamento do modelo *Crystal* e sua explicação segundo Qasaimeh.

O eixo X indica o tamanho da equipe, enquanto o eixo Y representa a criticidade do sistema. Quanto mais crítico o projeto, mais rigorosos e processos formais está requerido. O modelo *Crystal* define quatro níveis de crítica:

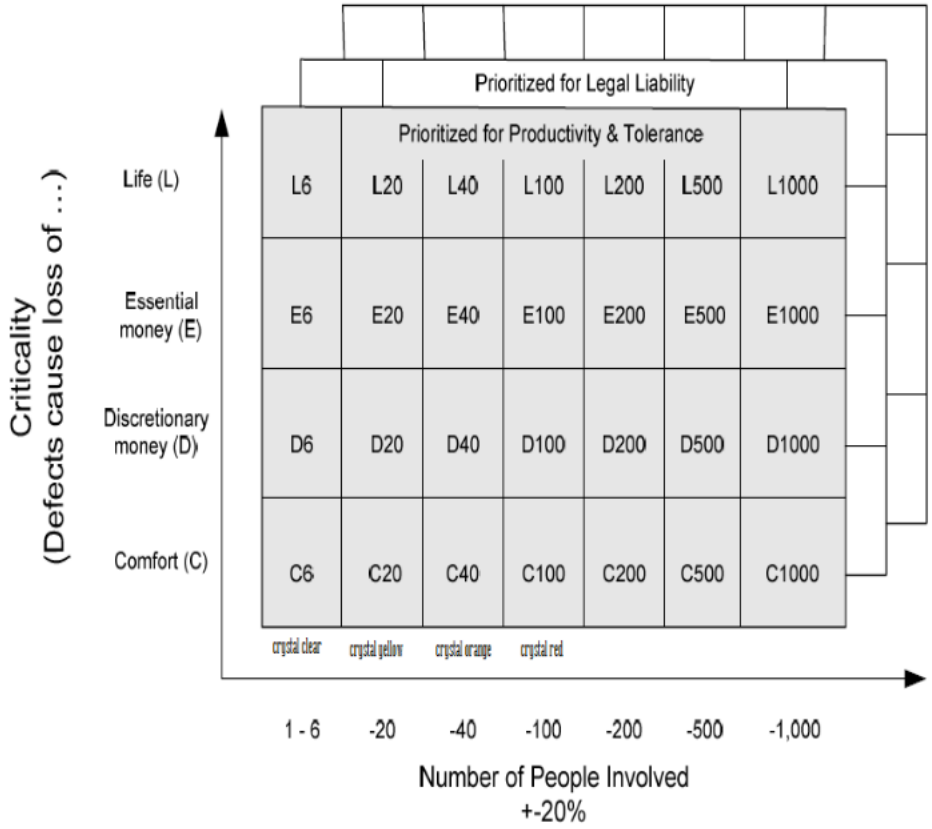
Vida (L): Uma falha do sistema é crítica e pode causar perda de vida.

Dinheiro essencial (E): Uma falha no sistema pode causar perda de dinheiro.

Dinheiro discricionário (D): Uma falha do sistema pode causar perda de dinheiro, mas pode ser corrigida consultando o manual do usuário do sistema.

Conforto (C): Uma falha no sistema pode causar perda de conforto do cliente.

Figura 15: Modelo *Crystal*



Fonte: (Qasaimeh et al., 2008)

No próximo capítulo pretende-se abordar algumas falhas nos modelos ágeis mais mencionados na literatura.

2.5 Falhas nos modelos ágeis

Os modelos ágeis, são os modelos de gestão de projetos mais utilizados atualmente nas empresas de desenvolvimento de *software* segundo a literatura. Esta informação se confirma, pela *14th annual State of Agile* em pesquisa feita com mais de 40.000 executivos, praticantes e consultores ágeis desde o seu início, nos quais 95% dos entrevistados praticam os modelos de desenvolvimento ágil nas empresas que atuam (StateOfAgile, 2020).

Kumar afirma que nada neste mundo é sem qualquer deficiência ou limitação e cita algumas limitações e deficiências do modelo ágil. São elas (Kumar & Bhatia, 2012):

- A ênfase principal do modelo ágil está no desenvolvimento, em vez de *design* e do utilizador. Basicamente se concentra em processos para obter requisitos e desenvolvimento de código e não se concentra em *design* de produto.
- Altos tempos de execução de teste e baixa cobertura de teste.

- Muitas equipes que exigem alta coordenação e comunicação dos gerentes de projeto.
- Não se adapta bem a grandes projetos, como numerosos são necessárias iterações para completar a desejada funcionalidade.
- Muito tempo pode ser dedicado a algo pequeno.
- Em um projeto de grande escala, custo de oportunidade para empregar agilidade podem ser muito altos para uma produção perdida em projetos mais lucrativos e enxutos.
- A sobrecarga de gerenciamento é aumentada porque um sucesso a aplicação de um modelo ágil depende muito do forte trabalho em equipe, o gerente do projeto deve permanecer envolvidos na dinâmica da equipe.

Soares afirma que o desafio dos modelos ágeis é encontrar maneiras de eliminar alguns de seus pontos fracos, como a falta de análise de riscos, sem torná-las modelos pesados como acontece com os modelos clássicos. Como riscos acontecem normalmente em projetos de desenvolvimento de *software*, este é um ponto negativo dos modelos ágeis (M. Soares, 2004).

Os modelos ágeis só funcionam com eficiência quando todos os membros da equipe estão realmente comprometidos com o projeto. Isso ocorre porque envolve a colaboração da equipe e reuniões diárias que podem consumir muito tempo (Portugal, 2020).

No próximo capítulo pretende-se abordar o que são os modelos híbridos na gestão de projetos de *software*.

2.6 Modelos híbridos

Mesmo diante de vários modelos existentes, percebe-se que, muitas vezes, os *softwares* ainda são desenvolvidos sem um planejamento e acompanhamento adequado (Santos, Santos, Reis, & Costa, n.d.).

O uso sistematizado de melhores práticas reunidas de vários modelos e *frameworks* pode ser uma opção para aumentar a produtividade e melhorar a qualidade do produto final (Santos et al., n.d.).

Eder, Conforto, Amaral e Silva por sua vez, definem que os modelos híbridos são a combinação de princípios, práticas, técnicas e ferramentas de diferentes abordagens em um processo sistemático que visa adequar a gestão para o contexto de negócio e tipo específico de projetos. Têm como objetivo maximizar o desempenho do projeto e produto, proporcionar um equilíbrio entre previsibilidade e flexibilidade, reduzir os riscos e aumentar a inovação, para entregar resultados melhores de negócio e valor agregado para o cliente (Eder, Conforto, Amaral, & Silva, 2014).

Existem muitas diferenças entre os modelos clássicos e ágeis, nomeadamente na forma como lidam com o projeto. Os clássicos usam uma abordagem preditiva e as equipas têm de seguir um plano predefinido para todas as tarefas e atividades do ciclo de vida. Por sua vez, os modelos ágeis são adaptativos e os requisitos encontram-se em constante mudança, sendo definidos muito próximo da sua implementação.

Podemos dizer que os modelos clássicos possuem maior foco na geração de documentação sobre o projeto e em seu cumprimento rígido de processos. Já os modelos ágeis concentram as atenções na entrega constante do produto e nas interações entre as pessoas envolvidas nos projetos.

Segundo Soares, uma característica dos modelos ágeis é que eles são adaptativos ao invés de serem preditivos. Com isso, eles se adaptam a novos fatores decorrentes do desenvolvimento do projeto, ao invés de procurar analisar previamente tudo o que pode acontecer no decorrer do desenvolvimento (M. Soares, 2004).

Diante das necessidades que variam em cada projeto, o mercado passou a falar em modelos híbridos, que possibilitam o cumprimento de importantes etapas de planejamento e documentação, mas também se aproveitam do ciclo de soluções rápidas proporcionado pelos modelos ágeis, em que as entregas de valor são feitas de forma contínua, através de processos interativos. O conceito deste modelo é de associar boas práticas clássicas como planejamento, controle de riscos e de processos para o foco em um âmbito desejado, com as boas práticas do modelo ágil na solução de cenários dinâmicos enfrentados cotidianamente (Francischini & Crist, 2016).

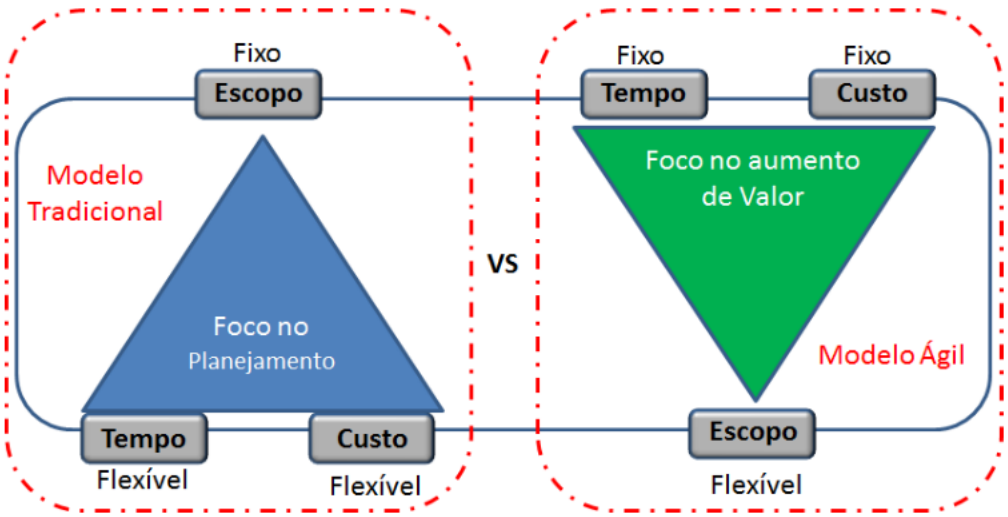
O modelo ágil permite aos clientes visualizarem o *software* com todas as funcionalidades ou parcialmente de acordo com as etapas, devido à proximidade com as equipes de desenvolvimentos e através das aceitações frequentes o cliente recebe o *software* com as funções solicitadas o que pode não ocorrer no modelo clássico pela distância natural que ocorre entre o cliente e a equipe de desenvolvimento e alterações no entendimento dos requisitos fornecidos pelo cliente, no final do projeto o cliente pode receber um *software* com funções adicionais não solicitadas e não agregando valores aos negócios (Trigo & Barreto, 2019).

É perceptível que o desenvolvimento de modelos híbridos de gestão de projetos é um desafio estratégico das empresas, a fim de criar ambientes dinâmicos e eficazes no desenvolvimento de soluções competitivas (Francischini & Crist, 2016).

Francischini afirma ainda que no quesito desenvolvimento de modelos híbridos, o entendimento entre as diferenças de cada abordagem é de suma importância para a construção de modelos robustos e eficientes. Modelos que atendam a necessidade de cada projeto e assim obter resultados esperados (Francischini & Crist, 2016).

Na Figura 16 conseguimos entender como funciona o modelo clássico em comparação com o modelo ágil.

Figura 16: Modelo clássico vs Modelo ágil



Fonte: (Francischini & Crist, 2016)

2.7 Sumário

Neste capítulo foi abordado diversos modelos clássicos e ágeis de gestão de projetos de desenvolvimento de *software*. O que são, como eles funcionam, seus valores e práticas.

Existem muitas diferenças entre as modelos clássicos e ágeis, nomeadamente na forma como lidam com o projeto. Os clássicos usam uma abordagem preditiva e as equipes têm de seguir um plano predefinido para todas as tarefas e atividades do ciclo de vida de um projeto. Por sua vez, os modelos ágeis são adaptativos e os requisitos encontram-se em constante mudança.

Após ser abordado as falhas na utilização dos modelos clássicos e ágeis, foi abordado o que são os modelos híbridos e a necessidade de utilização deles nos projetos para a melhor gestão de desenvolvimento dos *softwares*.

Apesar de ambos os modelos terem abordagens diferentes, ambos têm o mesmo objetivo final que é satisfazer o cliente e entregar o *software* de acordo com as suas necessidades.

No próximo capítulo pretende-se abordar algumas possibilidades de modelos híbridos comparando os modelos de gestão de projetos.

3 Análise entre os Modelos de Gestão de Projetos de Desenvolvimento de *Software*

Neste capítulo pretende-se fazer uma comparação dos modelos de gestão de projetos, atendendo à literatura.

Este encontra-se dividido em três seções. Na primeira seção será feito uma comparação entre os modelos ágeis dando ênfase ao modelo híbrido ScrumBan (combinação do Scrum e Kanban) e a comparação entre os modelos Scrum vs XP uma vez que são os modelos híbridos mais utilizados segundo literatura estudada (StateOfAgile, 2020).

Seguidamente, será feito uma comparação entre os modelos clássicos vs ágeis, dando ênfase ao Guia PMBOK® vs Scrum, pois além de serem os mais usados, existe muitos estudos utilizando a comparação e junção dos dois como um modelo híbrido de gestão de projetos de desenvolvimento de *software*.

Finalizando, teremos o sumário.

3.1 Comparação dos modelos ágeis

Os modelos ágeis possuem semelhanças em muitas das suas práticas e princípios, uma vez que todas foram construídas com base na mesma perspectiva ágil (Begel & Nagappan, 2007).

Com base na pesquisa e análise refeita por Abrahamsson foi analisado os pontos-chave, pontos fortes e pontos negativos de cada modelo de gestão que demonstramos na Tabela 1 (Abrahamsson, Warsta, Siponen, & Ronkainen, 2003).

Tabela 1: Comparação entre os modelos ágeis

Modelos Ágeis	Pontos-Chave	Pontos Fortes	Pontos Negativos
Scrum	- Processos simples; - Equipes de desenvolvimento organizadas; - Ciclos de entrega curtos (15 a 30 dias).	- Paradigma de orientação ao cliente.	- Requer outras abordagens para complementar o ciclo de vida; - Focado na gestão do projeto; - Ausência de práticas e técnicas de desenvolvimento.

Extreme Programming (XP)	- Desenvolvimento orientado ao cliente; - Equipes pequenas; - Ciclos de entrega curtos.	- Desenvolvimento orientado a testes; - Programação em par.	- Poucas práticas de gestão; - Ausência de documentação; - Focado essencialmente no desenvolvimento; - Indicado para equipas pequenas.
Kanban	- WIP – *Work in Progress;* - Começar com o que existe; - Ver o trabalho.	- Visualização do fluxo de trabalho num quadro.	- Focado na gestão do projeto; - Desenhado para manutenção; - Ausência de práticas e técnicas de desenvolvimento.
Scrumban	- Visualização do fluxo de trabalho num quadro; - WIP – *Work in progress;* - Ver o trabalho.	- Desenhado para as constantes mudanças de trabalho.	- Focado na gestão do projeto; - Ausência de práticas e técnicas de desenvolvimento.
Scrum & XP	- Equipes de desenvolvimento organizadas; - Ciclos de entrega curtos (15 a 30 dias).	- Desenvolvimento orientado a testes; - Programação em par.	- Ausência de documentação; - Indicado para equipas pequenas.
(ASD) *Adpative Software Development*	- Cultura Adaptativa; - Desenvolvimento baseado em componentes iterativos.	- As empresas são vistas como sistemas adaptativos.	- Encontra-se fortemente ligado a cultura e conceitos.

(DSDM) *Dynamic Systems Development Method*	- Controle para desenvolvimento rápido de aplicações (RAD); - Tempo estabelecido por interação; - Equipes autónomas; - Consórcio ativo para manter a modelo.	- Utilização de prototipagem; - Diversos papéis do utilizador.	- Sugere que os requisitos devem estar estáveis; - Rigidez nos princípios.
(FDD) *Feature Driven Development*	- Processo dividido em cinco passos; - Desenvolvimento baseado em componentes orientadas a objetos; - Pequenas interações.	- Método simplista; - *Design* e implementação por funcionalidades; - Modelagem de objetos.	- Foco em *design* e implementação; - Requer outras abordagens para complementar o ciclo de vida.

Fonte: Comparação entre modelos ágeis Adaptado de Qumer e Henderson-Sellers, 2008 e Harma, e Bawa, 2017 (J. Souza, 2018)

No estudo feito pela 14th conferência anual ágil, foi constatado que 10% dos entrevistados utilizam o modelo híbrido ScrumBan e que 8% utilizam o modelos híbridos com a junção do modelo Scrum e XP uma vez que os aspectos negativos de uma abordagem são complementados pela outra (StateOfAgile, 2020).

Por ser um dos modelos mais usados segundo a pesquisa, abordaremos mais profundamente nos próximos capítulos.

3.1.1 ScrumBan

O ScrumBan resulta da combinação dos modelos ágeis Scrum e Kanban.

Segundo pesquisa feita pela 14th conferência ágil o ScrumBan é o segundo modelo ágil híbrido mais utilizado pelos entrevistados, ficando atrás apenas de modelos híbridos que resultam da combinação de vários modelos ágeis (StateOfAgile, 2020).

O ScrumBan reúne as propriedades básicas do Scrum e a capacidade de melhorar o processo e flexibilidade do modelo Kanban permitindo que as equipes se movam rumo ao desenvolvimento ágil e à melhoria constante de seus processos.

O ScrumBan é um modelo híbrido que torna o Scrum mais enxuto e flexível para mudanças rápidas, utilização do planejamento e revisão de *sprint*, combinando com partes do modelo Kanban que utiliza os quadros de fluxo de trabalho, WIP, regras de equipe, etc (Lutfiani, Harahap, Aini, Ahmad, & Rahardja, 2020).

Acredita-se que seja especialmente adequado para projetos de manutenção ou projetos com histórias de usuário frequentes e inesperadas ou erros de programação (Rebaiaia & Rodrigues Vieira, 2014).

Lutfiani enumera os sete princípios usados no ScrumBan, são eles (Lutfiani et al., 2020):

- Visualização do trabalho - esta é uma das ferramentas tiradas do Kanban e aplicada ao ScrumBan, local em que o fluxo de trabalho pode ser visualizado do início ao fim.
- Trabalho de fila - no ScrumBan o trabalho é colocado em uma fila e, ao contrário do Scrum, todo o trabalho deve ser concluído em *sprints*.
- Limitar o trabalho em andamento (WIP) - a coisa mais importante no ScrumBan é aplicar restrições ao trabalho em andamento em cada capacidade da equipe.
- Tornar explícitas as regras da equipe - no Scrum cada equipe se auto-organiza e vai trabalhando na coordenação, mas na realidade sempre há lacunas que ocorrem na equipe, como se organizam e funcionam.
- Reuniões de planejamento - ao contrário do Scrum que têm um planejamento curto.
- Revisões, retrospectivas e reuniões diárias - são as mais importantes no ScrumBan. Fornecendo feedback diretamente do proprietário do produto ou de quem tem um relacionamento com a equipe, como gerentes e clientes.
- Métricas e estimativas opcionais no ScrumBan - no Scrum é estimado usando métricas como histórico e tarefas tomadas e inseridas no *sprint*.

Algumas vantagens deste modelo são o aumento da qualidade, diminuição do prazo de execução, decisões tomadas quando necessário (*just-in-time*), melhoria contínua, minimização do desperdício e melhoria do processo (Bruno, 2019).

Abrahamsson afirma que um dos pontos-chave do modelo ScrumBan é a visualização do fluxo de trabalho através de quadro em que toda equipe envolvida no projeto e o cliente podem visualizar a qualquer momento todo fluxo de trabalho num quadro. Outro benefício do ScrumBan é que ele é desenhado para as constantes mudanças de trabalho. Porém, ele é focado na gestão do projeto e falha devido à ausência de práticas e técnicas de desenvolvimento (Abrahamsson et al., 2003).

3.1.2 Scrum e XP

Uma vez que os modelos ágeis Scrum e XP, são as uns dos modelos híbridos mais utilizados pelas empresas conforme pesquisa feita pela 14th conferencia ágil, foi feito uma comparação os dois modelos ágeis (StateOfAgile, 2020).

No XP não existe a preocupação formal em fazer a análise e o planejamento de riscos. Como riscos acontecem normalmente em projetos de desenvolvimento de *software*, este é um ponto negativo da XP (SOARES, 2009).

No Scrum o escopo é definido como funcionais e não funcionais o que auxilia na execução por não possuir muita documentação o que pode ocasionar constantes mudanças e aumento dos riscos no decorrer do projeto.

Souza comparou os modelos, tendo em conta os seguintes aspetos: documentação; interação com o cliente; tamanho e complexidade do projeto; requisitos e construção do *software*. Este estudo foi feito com base nos autores Kiran Hiwarkar, Sriram Rajagopalan e Fahad (J. Souza, 2018).

Na Tabela 2 conseguimos visualizar as principais diferenças entre os dois modelos.

Tabela 2: Comparação Scrum vs XP

Abordagens	Scrum	XP
Documentação	- Produz a documentação estritamente necessária.	- Produz a documentação estritamente necessária
Interação com o cliente	- Necessita de grande envolvimento e compromisso por parte do cliente durante o processo de desenvolvimento.	- Necessita de grande envolvimento e compromisso por parte do cliente durante o processo de desenvolvimento.
Tamanho e complexidade do projeto	- Adapta-se melhor a projetos de dimensão mais reduzida e é um dos modelos mais adequados para projetos que exigem mudanças por considerar a comunicação permanente com o cliente.	- Adapta-se a qualquer tipo de projeto.
Requisitos do *software*	- Existe um conjunto de práticas que ajudam no levantamento de requisitos como o jogo do planeamento, a	- Todos os requisitos são colocados na lista de *Product Backlog*, priorizados e

	criação das *user stories* e a identificação dos intervenientes. - Não é realizada nenhuma distinção entre requisitos funcionais e não funcionais. - Um dos pontos mais fortes relaciona-se com a possibilidade de ser elaborada a descrição dos testes funcionais com a colaboração do cliente.	classificados por todas as partes envolventes.
Construção do *software*	- Encontra-se muito orientado para a implementação do *software*. - Conceção simples, o código é propriedade de todos os programadores, *pair programming, refctoring* e integração contínua. - Testes presentes em todas as fases do processo de desenvolvimento.	- É omisso, uma vez que o seu foco é a gestão de *software*.

Fonte: (J. Souza, 2018)

Podemos constatar que os modelos ágeis mesmo que combinados carecem de documentação.

3.2 Modelos clássicos vs ágeis

Os modelos clássicos dependem completamente da análise de requisitos e do planejamento cuidadoso no início do ciclo de vida de desenvolvimento de *software* (STOICA et al., 2013).

Stoica diz que o modelo ágil usa uma abordagem adaptativa em que não há planejamento detalhado e apenas tarefas futuras claras são aquelas relacionadas às características que devem ser desenvolvidas. A equipe se adapta às mudanças dinâmicas nos requisitos do produto (STOICA et al., 2013).

Cockburn afirma que os modelos ágeis não possuem nada de novo, o que os diferencia dos modelos clássicos é o enfoque e os valores (Cockburn, 2003).

A Tabela 3 conseguimos visualizar a diferença de forma resumida entre os modelos clássicos vs ágeis.

Tabela 3: Diferença entre os modelos clássicas vs ágeis

Abordagem	Modelos Clássicos	Modelos Ágeis
Hipóteses fundamentais	- Sistemas são totalmente especificáveis; - Desenvolvido por pequenas equipes.	- Software adaptativo de alta qualidade; - Sistemas são totalmente especificáveis; - Baseado em *feedback* rápido e mudança.
Estilo de gestão	- Comando e controle.	- Liderança e colaboração.
Gestão do conhecimento	- Explicito.	- Tático.
Comunicação	- Formal.	- Informal.
Modelo de desenvolvimento	- Modelo de entrega evolucionária.	- Modelo de entrega evolucionária.
Estrutura organizacional	- Mecânico (burocrático, alto estrutura formalização), visando grandes participativo, incentiva o social organização.	- Orgânico (flexível e Organizacional, participativo, incentiva o social), visando pequenas e médias empresas.
Controle de qualidade	- Planejamento difícil e controle rígido; - Teste difícil e tardio.	- Controle permanente ou controle de qualidade, requisitos, *design* e soluções; - Teste permanente.
Requisitos do usuário	- Detalhado e definido antes da codificação / implementação.	- Entrada interativa.
Custo de reinicialização	- Alto.	- Baixo.
Direção de desenvolvimento	- Fixo.	- Facilmente alterável.
Teste	- Depois que a codificação for concluída.	- Cada interação.

Envolvimento do cliente	- Baixo.	- Alto.
Habilidades adicionais	- Nada em particular.	- Habilidades interpessoais.
Escala apropriada do projeto	- Larga escala.	- Pequena e média escala.
Desenvolvedores	- Orientado no plano; - Com habilidades; - Acesso ao externo conhecimento.	- Ágil; - Desenvolvedores com habilidades avançadas; - Conhecimento cooperativo.
Clientes	- Com acesso ao conhecimento; - Cooperativos.	- Clientes dedicados e bem-informados; - Representante e cooperativo.
Requisitos	- Muito estáveis; - Conhecidos antecipadamente.	- Com mudanças rápidas.
Arquitetura	- Projetar para requisitos atuais e previsíveis.	- Projetar para os requisitos atuais.
Remodelação	- Cara.	- Barata.
Tamanho	- Times e projetos grandes.	- Times e projetos pequenos.
Objetivos primários	- Alta segurança.	- Valor rápido.

Fonte: (STOICA et al., 2013)

No relatório de 2015 do *Chaos Report*, lançado pelo *Standish Group*, realizou um estudo em mais de 10 mil projetos, em que é possível concluir que existe uma maior taxa de sucesso em projetos que adotaram modelos ágeis uma vez que os resultados de todos os projetos mostram que os projetos ágeis têm quase quatro vezes a taxa de sucesso dos projetos clássicos (The Standish Group International, 2015).

Na Figura 17 conseguimos visualizar que os resultados também são divididos por tamanho do projeto: grande, médio e pequeno. Os resultados gerais mostram claramente que os projetos clássicos não são bem escaláveis, enquanto os projetos ágeis têm uma escalabilidade muito melhor. No entanto, observe que quanto menor for o projeto, menor será a diferença entre o processo ágil e o processo em modelos clássicos.

Conforme declaramos no relatório *Chaos Report* de 2015, identificamos dois pontos que, juntos, criam uma mão vencedora. Os pontos são o processo ágil e os pequenos projetos. Conforme medido pelas métricas modernas, pequenos projetos que usam um processo ágil têm apenas uma taxa de falha de 4% (The Standish Group International, 2015).

Figura 17: Resolução CHAOS por modelo ágil vs modelo clássico

SIZE	METHOD	SUCCESSFUL	CHALLENGED	FAILED
All Size Projects	Agile	39%	52%	9%
	Waterfall	11%	60%	29%
Large Size Projects	Agile	18%	59%	23%
	Waterfall	3%	55%	42%
Medium Size Projects	Agile	27%	62%	11%
	Waterfall	7%	68%	25%
Small Size Projects	Agile	58%	38%	4%
	Waterfall	44%	45%	11%

Fonte: (The Standish Group International, 2015)

3.2.1 Guia PMBOK® vs Scrum

De acordo com o Guia PMBOK®, este modelo trabalha com vários grupos de processos que abrangem todas as fases de um projeto, escrito de forma genérica, descrevendo as boas práticas de gerenciamento sem determinar a área de conhecimento para sua aplicação (*Guia PMBOK®*, 2017).

Nesse sentido, Cruz afirma que o Guia PMBOK® não descreve “como fazer” projetos e sim descreve “o que deve ser feito” através de processos que abrangem diversas áreas de conhecimento. Este descritivo de boas práticas tem por objetivo flexibilizar os processos de gerenciamento permitindo a equipe determinar o grau de profundidade a ser atingido conforme a complexidade do projeto, dessa forma as peculiaridades de cada área de conhecimento são alcançadas pelo modelo (CRUZ, 2013).

Desta forma, a perspectiva de análise adotada no presente estudo considera que o Guia PMBOK® se propõe a ser um manual de boas práticas a ser executado em conjunto com um modelo ágil de gestão, sendo a última responsável pelo engessamento ou não dos projetos (Vargas, 2016).

Abordaremos na Tabela 4 uma comparação entre o Guia PMBOK® e o Scrum elaborado por Vargas com adaptação do autor com o intuito de identificar a diferença entre os modelos e analisar como elas podem se complementar para a criação do modelo híbrido proposto que será utilizado em estudo de um caso (Vargas, 2016).

Tabela 4: Comparação Guia PMBOK® vs Scrum

Abordagens	Guia PMBOK®	Scrum
Gerenciamento de projetos	- Foco no planejamento.	- Foco no produto.
Documentação	- Principal instrumento de controle; - Toda etapa do projeto deve ser documentada. Sendo este um fator impeditivo para a mudança de etapa; - Extensa e desenvolvida pelo gerente de projetos.	- Produzida conforme os ciclos são realizados; - O foco do trabalho está na implementação do produto com qualidade, a documentação é secundária; - Existe uma flexibilidade na produção; - Simples, resumida e com linguagem amigável.
Controle	- Necessita de um número maior de certezas para estabelecer as métricas de trabalho (custo, dimensionamento de equipe e tempo).	- Trabalha com incertezas e com priorizações durante o desenvolvimento do produto.
Conceito	- Âmbito claro e detalhado do que deverá ser realizado no projeto; - Devem ser descritos os elementos principais que servem de base para as estimativas de custo e tempo.	- Abstração de funcionalidades, estabelecendo um âmbito inicial que servirá de base para o detalhamento no decorrer dos ciclos de vida.
Mudanças	- São considerados riscos ao sucesso do projeto, não são impedidas, mas são permitidas apenas após análise de impacto e custos, os benefícios forem	- São consideradas inerentes ao processo de amadurecimento da equipe e devem ser implementadas no próximo ciclo possível.

	superiores aos custos de alteração de âmbito.	
Custos	- São estimados com base do âmbito e análise dos requisitos do projeto; - Existem processos específicos que gerenciam os custos do projeto; - A inexistência do âmbito inviabiliza a estimativa de custo do projeto.	- Os custos são estimados através do *product backlog*, agilizando o início da execução pois os detalhamentos são realizados a cada iteração; - Não há um processo específico que trate do assunto.
Orientação	- A processos: processos bem definidos devem ser executados obrigatoriamente para garantir o desenvolvimento de qualidade; - O gerente deve distribuir as atividades e montar o cronograma conforme os talentos das pessoas envolvidas no projeto.	- A pessoas: as pessoas são tratadas como indivíduos e não recursos. - O ritmo de trabalho é respeitado, cada membro da equipe escolhe quais e estimam o tempo a ser gasto das tarefas que irá desenvolver no próximo *sprint*.
Burocracia	- Burocrática: exigem o desenvolvimento da solução completa, gera sobrecarga da equipe e compromete a velocidade de desenvolvimento.	- Simplicidade: parte do princípio que realizar algo simples e útil é melhor do que gastar tempo em algo complexo que poderá não ser utilizado.
Equipes	- Grandes, selecionados conforme a estrutura da empresa; - Há obrigatoriamente um gerente de projetos como líder; - Os membros são alocados no projeto conforme a necessidade organizacional, podendo ter dedicação parcial ou total dos envolvidos, além de ser modificada durante a execução do projeto;	- Pequenas, multifuncionais, sem distinção de títulos profissionais, e, enquanto for economicamente viável, devem permanecer iguais durante todo o projeto; - Não há liderança designada, são auto-organizáveis; - O *Scrummaster* e o *product owner* não são considerados no número de membros da equipe pois podem participar de mais de um projeto.

	- Não há restrição quanto a formação de equipes geograficamente separadas.	
Ciclo de vida	- São definidos três modelos de ciclo de vida: preditivo, interativo/incremental e adaptativo; - Os dois últimos são mais utilizados e permitem o planejamento detalhado em ondas, o âmbito é planejado em uma visão macro no início do projeto e pormenorizado conforme as etapas de desenvolvimento se repetem.	- Ciclo de vida baseado em *sprints*, o projeto tem seu âmbito detalhado ao longo do desenvolvimento; - As reuniões de planejamento e revisão da *sprint* definem o detalhamento do âmbito do período e as possíveis mudanças necessárias.
Processo de gerenciamento de qualidade	- Deve ser aplicado a todo o projeto e garantir que os requisitos sejam cumpridos e validados com a melhor qualidade possível; - Os padrões de qualidade devem ser compatibilizados com as normas ISO.	- Está em atender o cliente através do cumprimento dos requisitos e do *feedback* constante do cliente ao longo do projeto; - A cada ciclo são retornados os pontos de melhoria que devem ser trabalhados o mais breve possível; - O foco está no produto com a melhor qualidade possível.
Comunicação	- Planejada através da identificação das necessidades das partes e determinação dos meios adequados; - Devem ser gerados documentos formais de comunicação para registrar e divulgar as conclusões as partes interessadas.	- Interpessoal, realizada nas reuniões de planejamento e revisão do *sprint*, e reuniões diárias; - São reuniões objetivas, rápidas e com pauta preestabelecida.
Cliente	- O cliente está presente em poucos momentos de planejamento do âmbito ou de gerenciamento de riscos.	- Participação ativa do cliente como parte do projeto.

Fechamento	- Ao final do projeto é realizada reunião para arquivamento de documentos e retomada das lições aprendidas durante o projeto; - Gerenciamento de aquisições.	- O processo de retomada das lições aprendidas é realizado ao final de cada *sprint* durante a reunião de retrospectiva.

Fonte: (Márques-Vargas, 2016) (Márques-Vargas, 2016) com adaptação do autor.

3.3 Sumário

Neste capítulo, constatou-se que nos modelos ágeis não existe a preocupação formal em fazer a análise e o planejamento de riscos e que a maioria dos modelos ágeis carecem de documentação, pois encontram-se essencialmente focados na gestão do projeto.

Também foi feita uma comparação entre os modelos ágeis e clássicos e podemos concluir que a junção dos dois modelos utilizando a documentação dos modelos clássicos e a gestão de projeto (desenvolvimento) dos modelos ágeis seria uma junção ideal para um modelo híbrido de gestão de projetos de desenvolvimento de *software*.

No próximo capítulo será criado e validado um modelo híbrido ágil através de um teste piloto.

4 Criação e Validação do Modelo Híbrido

Neste capítulo pretende-se apresentar um modelo híbrido, que vai ser criado e que procura responder à necessidade de um modelo ágil que permita ter mais documentação, práticas de desenvolvimento e que possua um âmbito bem definido no início do projeto.

Com isto, será proposto o modelo VIPKS, que será constituído utilizando como inspiração os modelos ágeis Scrum e Kanban e o modelo clássico orientado pelo Guia PMBOK®.

Primeiro pretende-se apresentar a criação do VIPKS, procedendo a sua validação em um projeto prático de desenvolvimento de *software* na empresa Virtual Innovations e será validado por questionário com a equipe envolvida no projeto.

4.1 Criação do modelo híbrido VIPKS

O nome do modelo VIPKS faz referência as iniciais da empresa na qual o modelo será testado e validado e as iniciais dos modelos que servirá de inspiração para criação deste modelo. São eles:

V – Virtual

I – Innovations

P – Guia PMBOK®

K - Kanban

S – Scrum

Este modelo pretende contribuir para a melhoria da gestão de projetos de *software*, passível de ser aplicada em pequenas e médias empresas de desenvolvimento de *software*.

O ambiente empresarial que originou este estudo utiliza grande parte do Scrum como seu modelo de gestão de projeto principal.

A ausência de documentação faz com que, frequentemente, os projetos iniciem sem que possuam uma definição detalhada do âmbito, e quando chega o momento de encerrar um projeto, o cliente consegue argumentar que ainda falta desenvolver mais funcionalidades para considerar o projeto concluído. Com base nesta constatação, criamos um modelo híbrido ágil no qual iremos unir práticas dos modelos ágeis e clássicos com o intuito de melhorar o desempenho das gestões de projetos de desenvolvimento de *software*.

O modelo VIPKS será dividido da seguinte forma:

- Guia PMBOK® – Início do projeto, criação do TAP (Termo de Abertura do projeto) e a parte formal da documentação (relatórios e reuniões).
- Scrum – Gerenciamento do projeto, desenvolvimento e entregas por *sprints*.
- Kanban – Ferramenta de gestão do projeto.

O modelo híbrido VIPKS possui uma abordagem ágil, que resulta da intersecção entre o modelo ágeis Scrum e Kanban com as boas práticas do modelo clássico recomendadas pelo Guia PMBOK®.

No próximo capítulo pretende-se abordar a base do modelo híbrido VIPKS.

4.2 Base do modelo híbrido VIPKS

Neste capítulo será estabelecido a base do modelo híbrido proposto VIPKS. Para isto, será estabelecido os valores, eventos, artefatos, estrutura organizacional e responsabilidades.

4.2.1 Valores

Neste capítulo serão estabelecidos os valores do modelo híbrido VIPKS, inspirados no modelo Scrum estudado anteriormente e aqueles que foram adquiridos através de experiências profissionais.

Todos na equipe devem se concentrar naquilo que é importante para o projeto. Com base nesta afirmação, foi estabelecido os valores abaixo que conseguimos visualizar na Figura 18 e que será explicado um a um posteriormente.

Figura 18: Valores VIPKS

- Abertura - a equipe e seus *stakeholders* devem estar abertos a todo o trabalho e aos desafios com transparência tornando todas as etapas conhecidas pelos envolvidos no projeto.
- Comprometimento - os membros da equipe devem estar comprometidos com as pessoas que estão em sua volta e através desse compromisso estarem prontos para alcançar qualquer desafio que possa aparecer. Devem estar dispostos a criar e cumprir metas realistas.

- Comunicação - a comunicação entre o cliente e os desenvolvedores deve ser feito de forma clara e objetiva. Quanto maior a capacidade de compreensão, maiores as chances de evitar problemas como ambiguidades e entendimento equivocados.
- Coragem - a equipe precisa ter coragem para atuar em problemas difíceis. Além de assumir possíveis falhas, deve se ter coragem também para fazer o melhor trabalho possível.
- Estilo de gestão - a gestão é baseada na colaboração de todos os envolvidos no projeto. Sejam eles direta ou indiretamente.
- Feedback - quanto mais cedo descobrirmos um problema, menos prejuízos ele pode causar e maiores são as chances de resolvê-lo. Os clientes, por sua vez, procuram se manter próximos da equipe de desenvolvimento e gestor de projetos para prover informações precisas sobre qualquer dúvida que eles tenham ao longo do desenvolvimento.
- Foco - a equipe deve se concentrar na entrega e ter pelo menos um incremento pronto até o final de cada *sprint*. Realizar a melhoria contínua é algo que o time deve praticar diariamente.
- Respeito - como equipes auto-organizadas, não podemos fazer nada sem respeito um pelo outro, a fim de cultivar um ambiente comprometido e produtivo para todos. Toda a equipe de desenvolvimento participa do planejamento, revisão e retrospectiva do *sprint*. Isso promove o respeito por cada função, responsabilidades e perspectivas diversas.
- Usuários - todos terão opinião e direito a sugestões. É priorizado o conhecimento interativo e cooperativo.

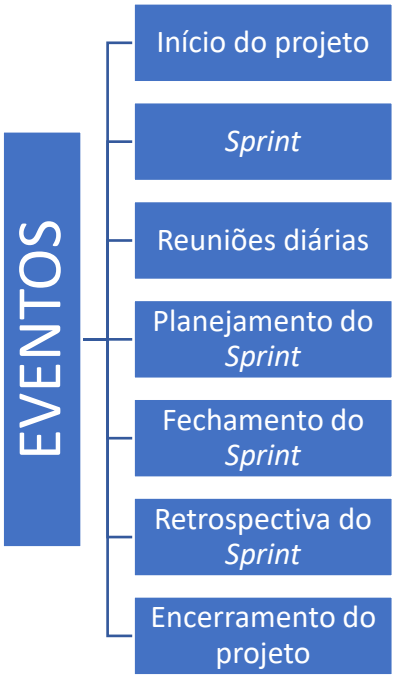
4.2.2 Eventos

Os eventos são criados para criar regularidade e diminuir a necessidade de reuniões não previstas. É uma oportunidade formal para inspecionar e adaptar algo. Eles são projetados especificamente para possibilitar transparência e inspeção.

Todos os eventos possuem tempo determinados e podem ser encerrados assim que cumpridos os seus objetos, evitando desperdícios de tempo dos envolvidos no projeto.

Na Figura 19 conseguimos visualizar quais são os eventos principais do VIPKS.

Figura 19: Eventos VIPKS



- Início do projeto - reuniões uma vez por semana para estabelecer o objetivo principal do projeto, o que se espera do mesmo para que tudo fique de acordo com o esperado pelo cliente.
- *Sprint* - possui de duas a quatro semanas de duração e no fim de cada *sprint* deve ser entregue ao cliente uma nova versão do seu produto. Neste período, a equipe deve desenvolver e testar um incremento de produto potencialmente utilizável. O escopo do *sprint* pode ser esclarecido e renegociado entre o cliente e a equipe de desenvolvimento, conforme o decorrer do projeto.
- Reuniões diárias - reuniões diárias de no máximo vinte minutos com a equipe de desenvolvimento e gestor do projeto em que se é esclarecido três perguntas principais:
    - O que você fez ontem?
    - O que fará hoje?
    - Existem alguns bloqueios ou impedimentos que prejudiquem seu trabalho?
- Planejamento do *sprint* - reuniões de no máximo oito horas com o objetivo de apresentar um plano de trabalho para o próximo *sprint* em que deverá ser definido:
    - Lista de requisitos do produto;
    - Incremento do produto mais recente;
    - Projeção da capacidade da equipe de desenvolvimento durante o *sprint*.

  A reunião de planejamento do *sprint* é dividida em duas subseções:

- O que? - o que podemos entregar de incremento ao produto.
- Como? - como podemos trabalhar para garantir o incremento do produto.

- Fechamento do *sprint* - a equipa apresenta ao cliente o resultado do seu trabalho, obtendo feedback do cliente para possíveis melhorias. Nesta etapa é entregue ao cliente um *status reporting* do *sprint* que constará o que foi feito e o que será feito no próximo *sprint* para juntos poderem alinhar a expectativa do cliente.
- Retrospectiva do *sprint* - deve durar no máximo três horas para um *sprint* de quatro semanas. A equipe de desenvolvimento se autoanalisa e traça um plano de melhoria a ser implementado no próximo *sprint*.
- Encerramento do projeto - deve ser feito uma reunião com toda equipe envolvida no projeto e documentado a finalização do projeto.

4.2.3 Artefatos

Os artefatos são criados para maximizar a transparência das principais informações nos projetos, pois permitem uma visão geral do produto a ser desenvolvido (J. J. Sutherland (Autor), 2020).

Na Figura 20 conseguimos visualizar os artefatos propostos pelo VIPKS.

Figura 20: Artefatos VIPKS

- Lista de requisitos do produto - é uma lista com tudo o que deve ser entregue no produto. Devendo estar organizada e bem detalhada com as funcionalidades principais e por prioridades de entrega uma vez que pode existir mudanças no decorrer do projeto.
- Termo de abertura do projeto (TAP) - é criado para marcar o início do projeto e alinhar o seu âmbito, cronograma, equipe e riscos. É sugerido utilizar uma lista de requisitos do produto e fazer reuniões com os clientes e *stakeholders* para alinhar as expectativas e mitigar os possíveis riscos que possam existir nos projetos. Nesta etapa tomaremos como inspiração o Guia PMBOK® para uma avaliação mais formal, porém, será feito de forma mais simplificada, pois normalmente é nesta etapa que os modelos ágeis pecam devido à falta de documentação. Porém, também utilizaremos a experiencia do autor para que esta etapa não seja muito extensa e tenha uma perspectiva ágil.
- Lista de requisitos do *sprint* - é uma parte da lista de requisitos do produto que a equipe insere no *sprint* para trabalhar. Todos os itens da lista de requisitos do *sprint* devem ser desenvolvidos, testados, documentados e integrados.
- Visão do *sprint* - nesta etapa, utilizaremos a ferramenta de gestão do projeto trello para podermos visualizar o andamento de todas as etapas de cada *sprint* uma vez que a empresa que fará o teste deste modelo que já o utiliza e está ambientado com ele (“https://trello.com,” n.d.). Este quadro deverá ser transparente para equipe e deve conter os prazos de entrega e o andamento das tarefas de forma interativa para que todos consigam saber o andamento do projeto.
- Modelo de arquitetura - deve ser definido no início do projeto, não sendo obrigatório seguir um modelo específico.
- Padrões de desenvolvimento - O código-fonte deve obedecer a padrões de desenvolvimento, previamente estabelecidos, e utilizados por todos os elementos da equipe. Para isso, devem ser instituídas regras e todos as devem seguir, sendo da responsabilidade *team leader* assegurar o bom funcionamento do projeto.
- Codificação - a implementação do código deverá ser feita de forma simples e comentada para que qualquer membro da equipe possa utilizar e atualizar caso seja necessário.
- Incremento - todos os itens do produto que foram entregues durante os *sprints*.
- *Status reporting* - deve ser criado gráficos com a evolução do produto após cada entrega dos *sprints* e ser informado ao cliente em qual etapa estamos, o que já foi desenvolvido e a expectativa de desenvolvimento futuro.
- Testes - para se concluir o desenvolvimento, os desenvolvedores devem realizar os testes e documentar os mesmos. A validação do projeto é feita durante todo o processo de

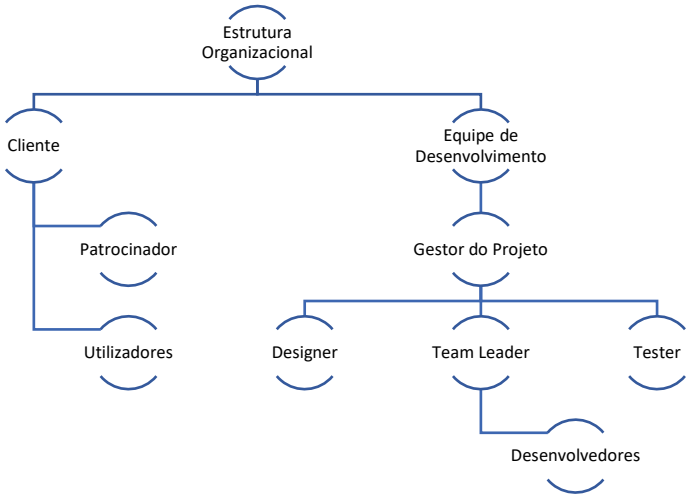
desenvolvimento, que são repetidas iteração a iteração. O cliente será responsável pela execução de testes de aceitação.

- Documentação - é necessário existir uma documentação que registre toda e qualquer mudança existente no decorrer do projeto assim como toda interação feita com o cliente com o propósito de ficar registado para o histórico do projeto, devendo sempre que possível, ligar esta informação com os itens à lista de requisitos do produto do termo de abertura do projeto (TAP).

4.2.4 Estrutura Organizacional e Responsabilidades

Neste capítulo pretende-se abordar a estrutura organizacional e as responsabilidades de todos os envolvidos no projeto. Ele será dividido em cliente e equipe de desenvolvimento. Na Figura 21 conseguimos visualizá-los.

Figura 21: Estrutura Organizacional VIPKS



- Cliente:
    - Patrocinador - pessoa que solicita o projeto. O patrocinador define o que é o projeto, seu âmbito e suas funcionalidades e quem são as pessoas que poderão ajudar nas especificações e no comportamento do produto.

- Utilizadores - são pessoas que o patrocinador escolhe como a principal fonte de informação relativamente aos processos, requisitos e testes de aceitação. Uma das suas principais responsabilidades relaciona-se com a validação de cada entrega, na qual avaliam se o resultado da entrega se encontra de acordo com as especificações.

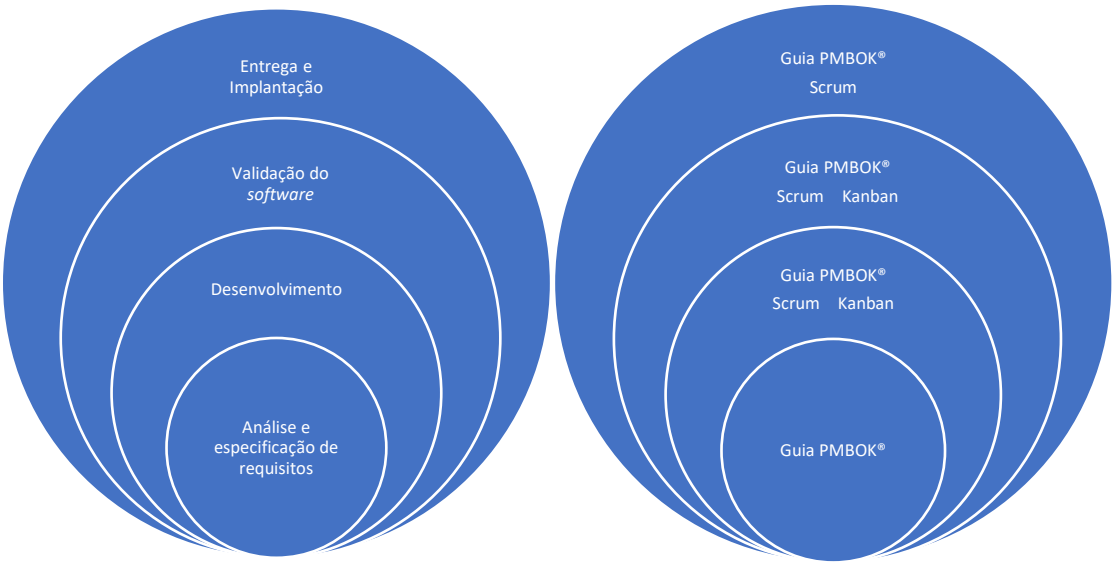
- Equipe de desenvolvimento:
  - Gestor do projeto - atua diretamente junto ao cliente. É responsável por representar o cliente perante a equipe de desenvolvimento. Tem como responsabilidade garantir que os processos e a estrutura do projeto são cumpridos e que o projeto obedece aos objetivos traçados. Deve efetuar a primeira análise dos pedidos do cliente e manter a equipa motivada, de forma a evitar conflitos internos. Deve controlar as horas do desenvolvimento para acompanhamento do projeto.
  - *Designer* - responsável por desenvolver o visual do produto e suas funcionalidades antes do início do desenvolvimento. O desenvolvimento só poderá iniciar após a validação do cliente em cada etapa do projeto. É importante que ele participe das reuniões iniciais para entender tudo que o cliente deseja que o produto possua.
  - *Team Leader* - é um desenvolvedor com mais experiência que será responsável pela arquitetura do sistema. Ele tem autonomia para determinar como as atividades serão executadas, sem depender de gerência.
  - Desenvolvedores - responsáveis por codificar e entregar os incrementos do produto no final de cada etapa de acordo com o que foi desenvolvido pelo designer e aprovado pelo cliente. Deve ter a capacidade de análise para poder verificar impactos e reduzir riscos.
  - *Tester* - responsável pela qualidade do produto. São responsáveis por realizar e documentar testes identificando assim possíveis falhas para acerto antes do trabalho ser entregue ao cliente.

No próximo capítulo pretende-se especificar como será feito o funcionamento do modelo VIPKS.

4.3 Funcionamento do modelo VIPKS

O funcionamento do modelo híbrido VIPKS foi dividido por etapas de acordo com o ciclo de vida de um projeto de desenvolvimento de *software*. Na Figura 22 conseguimos visualizar quais modelos iremos utilizar em cada etapa do projeto.

Figura 22: Funcionamento VIPKS



4.3.1 Análise e especificação de requisitos

Nesta etapa será feito a definição do produto. Para tal, será realizado reuniões semanais com o o *team leader*, o *designer* e o gestor do projeto juntamente com o cliente com o objetivo de entender e criar o âmbito do projeto, restrições e suas funcionalidades levando em consideração sempre a expectativa do cliente e o melhor funcionamento do produto.

Com base nestas informações o gestor do projeto pode fazer estimativas através do termo de abertura do projeto (TAP).

O termo de abertura de um projeto é um dos artefatos mais importantes nesta etapa, pois reforça o compromisso entre o cliente e a equipe no entendimento dos requisitos do projeto e na definição do âmbito, não perdendo a agilidade, pois durante o desenvolvimento pode ser adaptado.

No termo de abertura do projeto (TAP) deverá conter:

- Definição do âmbito.
- Estimativa dos recursos.
- Definição do time do projeto.
- Custos e orçamento do projeto.

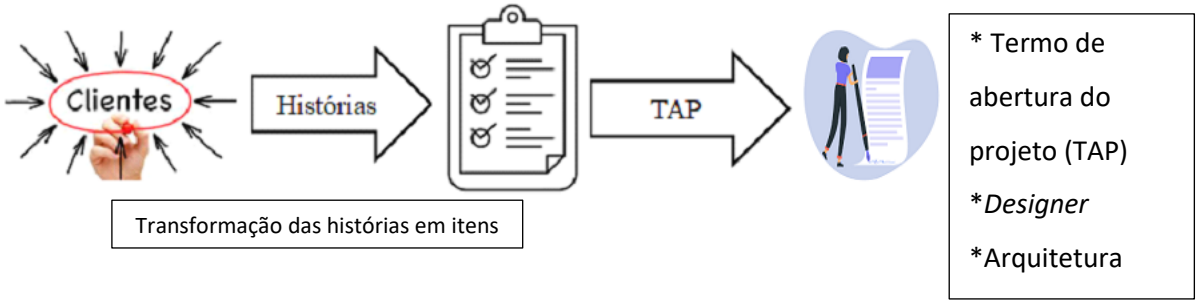
A equipe de desenvolvimento deverá definir já nesta fase a arquitetura e a definição das boas práticas que será utilizado por toda equipe de desenvolvimento.

Após a coleta de dados o *designer* cria o visual do projeto com as funcionalidades principais. As funcionalidades especificas devem ser criadas no decorrer do projeto, antes da elaboração de cada *sprint*.

Nesta etapa também será definido a linguagem do projeto. Os desenvolvedores definirão junto com toda equipe a linguagem que melhor se adequa ao estilo do projeto. Sendo o *team leader* o responsável por esta tomada de decisão.

Na Figura 23 conseguimos visualizar como será o início da gestão do projeto proposto no modelo VIPKS.

Figura 23: Início VIPKS



4.3.2 Desenvolvimento

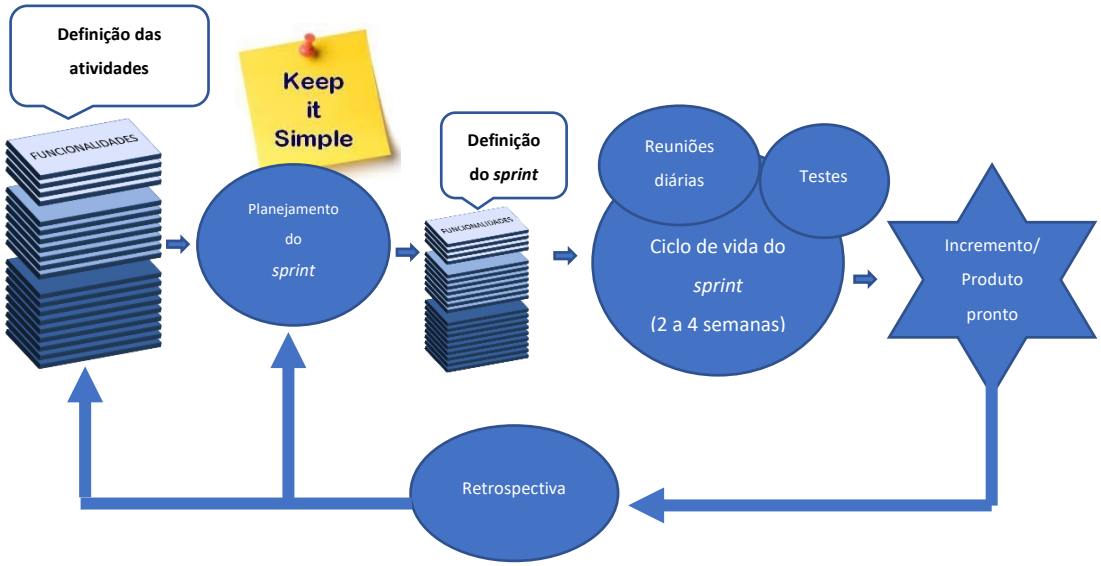
O processo de desenvolvimento do VIPKS será feito de forma semelhante ao Scrum que divide o projeto em fases. Em cada fase, uma parte será totalmente desenvolvida, testada e se tornará pronta para ir para a produção. A equipe não se move para uma nova fase até a fase atual ser concluída.

Inicialmente será definido as atividades, estimado a duração, o cronograma e será identificado os riscos de cada *sprint* com o objetivo de garantir a qualidade e planejar as possíveis mudanças.

O *designer* criará os documentos com e sem as interações para aprovação de cada etapa do projeto junto ao cliente. Estes documentos serão feitos por etapa, respeitando sempre os *sprints* devido as mudanças que poderá ocorrer no decorrer do projeto.

Na Figura 24 podemos observar como funciona um *sprint* no modelo VIPKS que será explicado passo a passo a seguir:

Figura 24: Ciclo de vida de um *sprint* VIPKS



- Primeiro é feito o planejamento do *sprint* com base nas definições das atividades elaborado no início do projeto no TAP e no documento com a visão do produto elaborado pelo *designer*.
- Depois será definido o que será desenvolvido no *sprint*. Com base na definição do *sprint* será planejado:
  - Priorização das atividades.
  - Cronograma do *sprint*.
  - Identificação dos riscos entre as métricas de qualidade.
  - Gestão das mudanças.
- Em seguida será feito o desenvolvimento em si que possui como regra:
  - Reuniões diárias - no primeiro horário do dia de no máximo vinte minutos para que todos possam saber em que ponto os outros estão, se possuem alguma dificuldade e traçar as metas do dia.
  - Ciclo de vida do *sprint* – duas a quatro semanas.
  - Reunião com o cliente semanal para entrega de relatório para informar o andamento do *sprint* (*status reporting*).
- Durante o *sprint* o gestor do projeto fica responsável pelos seguintes processos e controles:
  - Controlar o envolvimento dos *stakeholders*.
  - Monitorar e controlar o trabalho do projeto.
  - Controlar as mudanças.
  - Controlar o escopo.

- Controlar o cronograma.
- Controlar os custos.

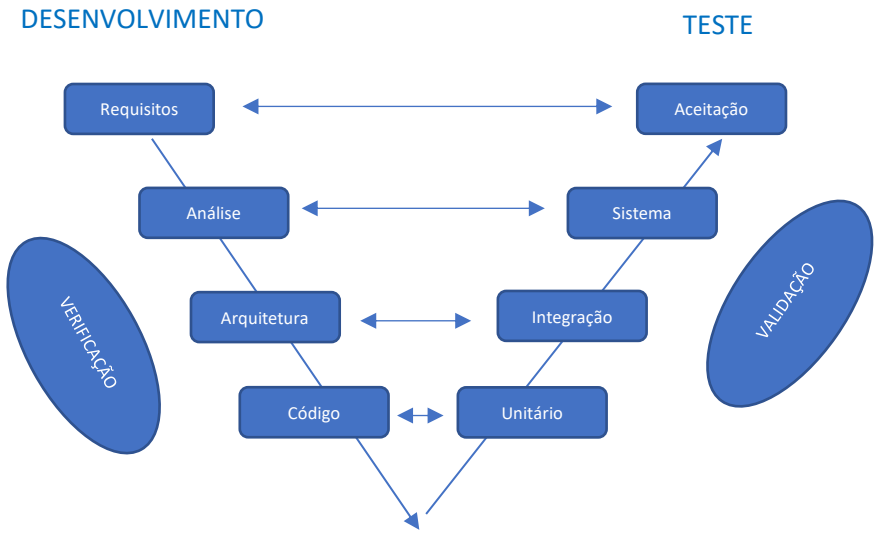
- Após o desenvolvimento começamos a parte de testes que será falado mais detalhadamente no itém 4.3.3 em Validação do s*oftware*.
- Em seguida teremos o itém 4.3.4 Entrega e Implantação no qual será explicado a retrospectiva e o incremento.

4.3.3 Validação do software

A validação do projeto será feita durante todo o processo de desenvolvimento, no final de cada *sprint* através de testes e são repetidos iteração a iteração. O procedimento tem como objetivo antecipar e corrigir falhas e bugs que apareceriam para o usuário final. Toda etapa de testes deve ser documentada.

Na Figura 25 podemos verificar os principais testes que serão utilizados no modelo VIPKS e logo em seguida será explicado cada um deles.

Figura 25: Modelo de testes do VIPKS



- Teste unitário - são executados para eliminar *bugs* no nível do código ou no nível da unidade. O teste de unidade verifica se a menor entidade pode funcionar corretamente quando isolada do resto dos códigos/unidades.
- Teste de Integração - em vez de se atestar funcionalidades do *software*, se analisa a integração entre as diferentes unidades que formam o sistema. Os resultados dos testes são compartilhados com a equipe do cliente.

- Teste de sistema - garante que as expectativas do aplicativo desenvolvido sejam atendidas. Verifica se os requisitos funcionais e não funcionais foram atendidos. Teste de carga e desempenho, teste de estresse, teste de regressão, são subconjuntos do teste de sistema. Deverão ser utilizados os testes mais importantes de acordo com o *software* a ser desenvolvido.
- Teste de aceitação – verifica se o sistema entregue atende aos requisitos do usuário e se o sistema está pronto para uso em tempo real. Dessa forma, o responsável deve checar a organização dos itens disponíveis na tela, observar se o *layout* está correto e se os botões se comunicam corretamente entre as diferentes páginas do sistema. O processo permite, também, verificar o comportamento da plataforma em diferentes dispositivos. Caso esteja utilizando diferentes navegadores ou dispositivos de tamanhos diferentes, saberemos o quanto o *layout* é responsivo ou não.

Após a validação, parte do produto será entregue ao cliente para que ele possa fazer os devidos testes e validações. Pretende-se que se encontre dentro dos parâmetros de qualidade estipulada e acrescente valor para o negócio.

4.3.4 Entrega e Implantação

A entrega e implantação do projeto deverá ser dividida em duas fases: entrega do *sprint* e encerramento. É importante observar que todos os integrantes do projeto, incluindo o time do cliente participam desta etapa do projeto (CRUZ, 2013).

4.3.4.1 Entrega do sprint

Na entrega do *sprint* é feita uma reunião de revisão do *sprint* em que são realizadas as seguintes atividades:

- Apresentação dos incrementos dos produtos finalizados e prontos - nesta fase será feito a entrega do produto testado e validado pelo *tester* e será feito a validação do âmbito com toda a equipe de desenvolvimento, o gestor do projeto e o cliente.
- Aprovação/Aceitação do cliente - será feito a entrega do produto ao cliente para que ele possa fazer os testes de aceitação e dar a sua aprovação. Nesta etapa participará o *team leader,* o gestor do projeto e o cliente.
- Solicitações de mudanças - é de salientar que se o cliente pretende fazer uma alteração à lista de requisitos, isto terá impacto sobre o trabalho realizado, que implica uma mudança ao que foi feito. Esta alteração deve ficar registrada na visão do produto e ser analisada junto ao

cliente para que ele tenha conhecimento. Estas mudanças poderão ter impacto no custo e/ou tempo do projeto. Por isto, toda mudança deve ser analisada com calma e aprovada junto ao cliente.

- Retrospetiva - após esta entrega, será feito uma retrospetiva, que é um evento no qual a equipe analisa o seu desempenho e faz uma autocrítica e aponta possíveis melhorias para o próximo *sprint*. A reunião de retrospetiva é conduzida pelo *team leader* com a participação de toda equipe de desenvolvimento, incluindo o gestor do projeto. Além de buscar a melhoria continua esta reunião alimenta o gestor do projeto para gerenciar e controlar as comunicações, os riscos e a equipe do projeto. Além de registrar as lições aprendidas para o próximo *sprint*.

4.3.4.2 Encerramento

No encerramento do *sprint*, o gerente do projeto faz a validação do âmbito com o cliente e efetua a finalização formal do *sprint* após o aceite final do cliente. Nesta etapa será encaminhado uma documentação para que o cliente esteja ciente e de acordo com a entega e possíveis mudanças no âmbito do projeto.

Após o termino do último *sprint* e incremento com aprovação do cliente e finalização de todos os treinamentos, o gestor do projeto pode encerrar o projeto, realizando algumas atividades, são elas:

- Arquivamento dos aceites finais do projeto.
- Entrega dos manuais de utilização e documentação do projeto.
- Repasse da tecnologia para todas as áreas afins.
- Registro das lições aprendidas.

Na Tabela 5 podemos observar como foi utilizado os modelos de gestão de projetos existentes e o referencial bibliográfico que serviu como inspiração e objeto de pesquisa para a criação do modelo híbrido VIPKS.

Tabela 5: Funcionamento do modelo híbrido VIPKS e Referencial bibliográfico

Modelo híbrido VIPKS	**Modelos de inspiração para a criação do modelo híbrido VIPKS**	**Atividades principais de acordo com o ciclo de vida do modelo híbrido VIPKS**	**Referencial Bibliográfico**
Análise e especificação de requisitos	Guia PMBOK®	Criação do Termo de abertura do projeto	(*Guia PMBOK®*, 2017)

Desenvolvimento	Guia PMBOK®	Documentos com e sem as interações para aprovação de cada etapa do projeto junto ao cliente; Documento com a duração do projeto, o cronograma e os riscos de cada *sprint.*	(*Guia PMBOK®*, 2017)
	Scrum	Reuniões diárias e semanais; Entrega por *sprint*; Reunião de planejamento do Sprint; Retrospectiva do *sprint*.	(Tomás, 2009); (Sharma & Aggarwal, 2013); (J. J. Sutherland (Autor), 2020)
	Kanban	Ferramenta de gestão de projeto dividido em quadro.	(Bruno, 2019); (https://trello.com)
Validação do *software*	Guia PMBOK®	Documentação dos testes com a aprovação do cliente.	(*Guia PMBOK®*, 2017)
	Scrum	Testes feito na entrega de cada sprint.	(Tomás, 2009); (Sharma & Aggarwal, 2013); (J. J. Sutherland (Autor), 2020)
	Kanban	Ferramenta de gestão de projeto dividido em quadro.	(Bruno, 2019); (https://trello.com)
Entrega e Implantação	Guia PMBOK®	Arquivamento dos aceites finais do projeto; Entrega dos manuais de utilização e documentação do projeto; Registro das lições aprendidas.	(*Guia PMBOK®*, 2017); (Camargo, 2019)
	Scrum	Entrega do *sprint* e encerramento.	(Tomás, 2009); (Sharma & Aggarwal, 2013); (J. J. Sutherland (Autor), 2020)

4.4 Validação do modelo híbrido VIPKS

Neste capítulo pretende-se fazer a validação do modelo híbrido VIPKS em um projeto piloto na empresa Virtual Innovations através de um investigação-ação experimental em um caso real.

Com isto, foi escolhido um projeto de desenvolvimento de *software* iniciado no mês de junho de 2021 com um prazo de entrega de oito semanas no qual foi utilizado o modelo VIPKS desde o pedido do cliente até a entrega do produto.

4.4.1 Implementação do modelo híbrido VIPKS em projeto piloto

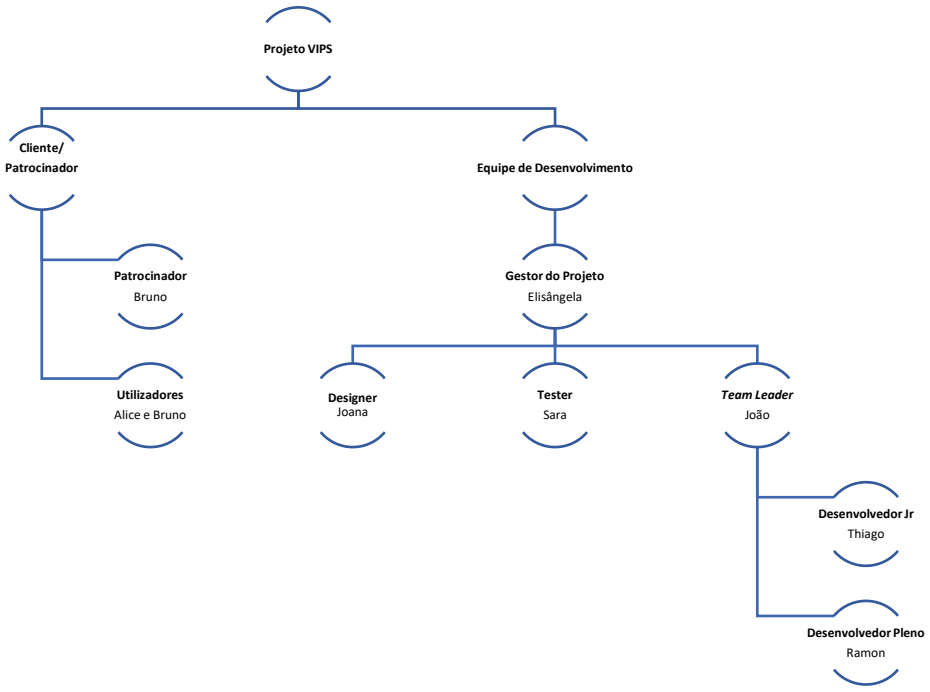
Para a implementação do modelo híbrido VIPKS foi escolhido uma equipe com experiência de pelo menos cinco anos utilizando os modelos ágeis como base para as gestões dos projetos.

O autor atuou como gestor do projeto, visto que já ocupa esta função dentro da empresa em outros projetos e pôde orientar a equipe para seguir as fases do modelo VIPKS.

Inicialmente foi criado o termo de abertura do projeto (TAP) após reuniões entre o cliente, o *designer*, o *team leader* e o gestor do projeto e com isto foi criado a definição do produto e o *design*. Foi também estabelecido a equipe e os custos do projeto. Com base no TAP o desenvolvimento foi iniciado utilizando a visão do produto.

A equipe foi constituida por: três desenvolvedores sendo um junior e dois plenos (sendo um deles o *team leader),* um *designer*, um *tester* e um gestor do projeto. Na Figura 26 conseguimos visualizar a equipe do projeto e suas funções.

Figura 26: Equipe do projeto piloto



Após ser estabelecido a equipe, foi estabelecido o modelo de arquitetura e os *sprints*. O projeto terá uma duração de oito semanas, com isto, foi estabelecido quatro *sprints* (um a cada quinzena).

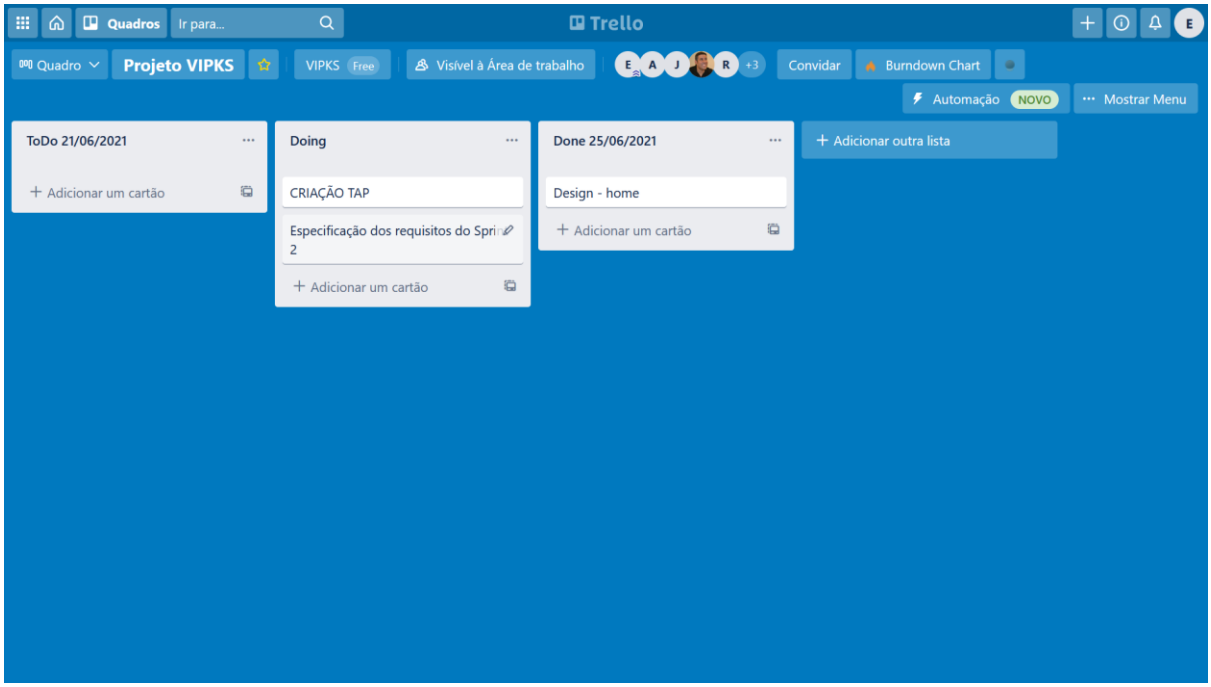
No primeiro *sprint*, a equipe de desenvolvimento definiu a arquitetura do projeto. Todos da equipe participaram e definiram o que deve ser feito em cada *sprint*, indo de acordo com os padrões do modelo VIPKS.

Para que o *sprint* seja bem sucedido, além das reuniões diária é necessário fazer uma boa gestão das horas. O gestor do projeto é responsável pelo controle e acompanhamento destas horas.

A equipe do projeto utilizou uma ferramenta de acompanhamento já utilizada na empresa e conhecida por toda equipe em que todos os envolvidos no projeto podiam verificar, a qualquer momento, o andamento do *sprint* e do projeto como um todo. Na Figura 27 conseguimos visualizar como a ferramenta funciona. Foram utilizados três itens, são eles:

- *ToDo* (data do *sprint*) - tudo que deve ser feito no *sprint*.
- *Doing* - o que está em andamento no *sprint*.
- *Done* - tudo que será entregue no *sprint*.

Figura 27: Quadro de um *sprint* VIPKS (Trello)



Fonte: (“https://trello.com,” n.d.)

Após a execução de cada *sprint* (incluindo os testes) foi feito a entrega ao cliente, seu encerramento e a retrospectiva. A análise retrospetiva ao projeto possibilita à equipe aprender com os erros anteriores para melhoria continua durante o projeto e projetos futuros.

No encerramento do projeto, foi feito a estabilização e formação do cliente sobre o funcionamento do projeto em um único *sprint* e foi realizado os últimos pedidos de melhorias antes da entrega final do projeto.

4.4.2 Análise da implementação do modelo VIPKS

Após a execução e implementação do projeto piloto foi feito um questionário, que podemos visualizar no Anexo 1, em que foi respondido por toda equipe envolvida no desenvolvimento do projeto.

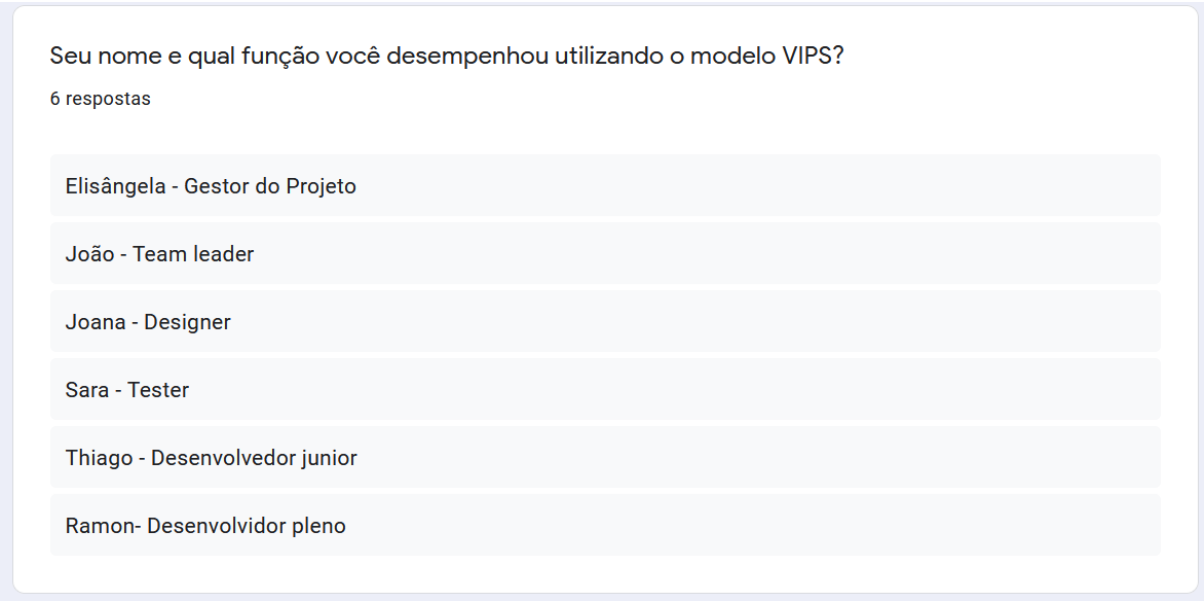
O objetivo deste questionário foi validar se o modelo híbrido VIPKS é útil para os projetos de desenvolvimento de *software*, se ajudou a solucionar o problema de gestão do âmbito existente em muitas empresas assim como a relação e comunicação entre o cliente e a equipe de desenvolvimento.

O questionário foi dividido de acordo com o ciclo de vida do projeto e questões genéricas para uma melhor análise dos dados. Foi utilizado uma escala de 0 a 5 para quantificar as respostas de cada questão, na qual 0 é considerado fraco e 5 é forte.

A seguir conseguimos validar como o modelo híbrido VIPKS contribuiu para a melhoria dos processos de desenvolvimento de *software* após a execução de todo projeto.

Conforme podemos ver na Figura 28 todos os envolvidos no projeto responderam as questões propostas para análise do modelo VIPKS.

Figura 28: Pessoas que responderam ao questionário VIPKS



Fonte: Google formulário VIPKS

Na primeira etapa do ciclo de vida do projeto (Análise e especificação de requisitos), é muito importante que exista uma boa comunicação entre o cliente e os envolvidos no projeto para que seja definido e documentado o âmbito do projeto com bastante detalhe evitando assim problemas no decorrer do projeto.

Podemos constatar na Tabela 6, que a questão “Considera importante a criação do TAP (Termo de Abertura do Projeto)?” obteve uma pontuação máxima com média 5, nos quais todos os

envolvidos no projeto consideram este documento importante para ajudar toda a equipa a entender o objetivo do projeto. O sucesso deve-se ao fato deste documento ter sido feito de forma simplificada do que é sugerido pelo Guia PMBOK® não desperdiçando tanto tempo do projeto e por ter dado informações mais detalhadas ao projeto com definições funcionais e não funcionais.

Já as demais questões obtiveram todas médias superiores a 4, indicando que as etapas de definições e estimativas do projeto feitas no seu início, contribuem para o seu sucesso.

A questão "Considera importante a definição do time do projeto?" reuniu a nota máxima pela equipe de codificação, enquanto o encarregado pelo *designer* não lhe atribuiu tanta importância, com uma classificação de 3. Uma das possíveis razões é o *design* não possuir grande interação ao longo do projeto com toda a equipe, tendo um contato maior com o cliente e o gestor do projeto.

"Considera importante a definição do âmbito?" teve a segunda maior média com 4,66, como podemos constatar na Tabela 6, contribuindo para a média geral de 4,56. Não sendo considerado de grande importância pelo *tester* uma vez que ele não precisa da sua definição para execução da sua atividade.

Tabela 6: Questionário – Análise e especificação de requisitos

Questões - Análise e especificação de requisitos	**GP**	***Design***	***Tester***	**TL**	**Desenvol-vedor Junior**	**Desenvol-vedor Pleno**	**Média 4,56**
Considera importante a criação do TAP (Termo de Abertura do Projeto)?	5	5	5	5	5	5	5
Considera importante a definição do âmbito?	5	5	4	5	5	4	4,66
Considera importante a estimativa dos recursos?	5	5	4	5	4	4	4,5
Considera importante a definição do time do projeto?	5	3	4	5	4	5	4,33
Considera importante a definição da arquitetura e das boas práticas?	4	4	4	5	5	4	4,33

Na segunda etapa do ciclo de vida do projeto (Desenvolvimento), é importante que exista uma boa compreensão e comunicação entre todos os envolvidos para evitar erros, retrabalho e custos desnecessários.

Na Tabela 7 podemos contatar que a importância das reuniões diárias reuniu a melhor avaliação, com uma média de 4,83, garantindo assim que os erros e dúvidas fossem identificados antes do desenvolvimento e entrega do *sprint* alinhando a expectativa do cliente com o desenvolvimento do projeto, evitando desperdício de tempo e retrabalho.

A pior média, com 3,83, foi "Considera importante a documentação (entregas e reuniões) dos *sprints*?". Uma das justificativas possíveis é a documentação representar um esforço adicional em comparação a outros modelos ágeis e isto ainda causar alguma resistência, principalmente para o *tester* e *designer* que foram os mais resistentes entre a criação de documentação e o uso de ferramenta de acompanhamento do projeto com uma pontuação de 3 em cada uma delas.

Tabela 7: Questionário - Desenvolvimento

Questões Desenvolvimento	**GP**	***Design***	***Tester***	**TL**	**Desenvol-vedor Junior**	**Desenvol-vedor Pleno**	**Média 4,29**
Considera importante a reunião diária?	5	5	4	5	5	5	4,83
Considera importante a visão do *sprint*?	5	4	4	5	5	4	4,5
Considera importante o ciclo de vida do *sprint*?	4	3	3	5	5	5	4,16
Considera importante a documentação (entregas e reuniões) dos *sprints*?	4	3	3	5	4	4	3,83
A ferramenta de acompanhamento do projeto escolhido (Trello) ajudou a organizar os *sprint*s?	4	3	3	5	5	5	4,16

Na terceira etapa do ciclo de vida do projeto (Validação do *software*), como podemos visualizar na Tabela 8, foi a parte que teve a menor média, 4,16. Uma das possíveis razões é a equipe de desenvolvimento não considerar esta etapa mais detalhada nos modelos que utiliza. Porém, obteve a pontuação máxima do *Tester* que é o profissional que atua diretamente nesta fase do projeto levando a considerar que a maneira que os testes foram divididos teve grande impacto no desenvolvimento do seu trabalho.

Tabela 8: Questionário – Validação de software

Questões - Validação do *software*	**GP**	***Design***	***Tester***	**TL**	**Desenvol-vedor Junior**	**Desenvol-vedor Pleno**	**Média 4,16**
Considera importante como foi dividido os tipos de testes?	4	3	5	3	4	3	4,16
A validação com o modelo VIPKS permite antecipar e corrigir falhas e *bugs* que apareceriam para o usuário final?	4	4	4	4	4	3	4,16

Na quarta etapa do ciclo de vida do projeto (Entrega e Implantação), o mais importante de um projeto é que ele seja entregue e aprovado pelo cliente para que ele se sinta satisfeito com o resultado apresentado.

Como podemos constatar na Tabela 9, obteve-se uma média de 4,25. A questão “Participação de toda equipe na apresentação dos incrementos dos produtos finalizados e prontos foi uma prática importante?” teve uma média de 4,5. Uma das justificativas é que a equipe se sente valorizada por fazer parte da entrega do produto.

Porém ainda existe alguma resistência na elaboração das documentações. Este fato provavelmente deve-se a falta de costume por parte da equipe e por considerarem que esta etapa gera mais trabalho.

Tabela 9: Questionário – Entrega e Implantação

Questões - Entrega e Implantação	GP	*Design*	*Tester*	TL	Desenvol-vedor Junior	Desenvol-vedor Pleno	Média 4,25
Participação de toda equipe na apresentação dos incrementos dos produtos finalizados e prontos foi uma prática importante?	5	5	3	5	5	4	4,5
Considera importante a documentação de entrega e implantação do *sprint*?	5	4	3	4	4	4	4

A etapa final do questionário teve como objetivo perceber se o modelo híbrido VIPKS obteve um melhor resultado nos projetos e se os envolvidos no projeto voltariam a usar este modelo nos seus projetos.

Foi possível verificar na Tabela 10 que teve uma boa aceitação com uma média de 4,26 no qual constatou-se que o projeto ficou mais organizado, com melhor eficácia e qualidade para maior satisfação do cliente. Podendo ser detectado de forma precoce os erros/defeitos.

A questão “O desenvolvimento com VIPKS torna o trabalho mais organizado/planejado?” teve o melhor resultado, com 4,5. O que mostra que a equipe de desenvolvimento gostou de trabalhar com este modelo.

Com menos pontuação foi a questão “Se pudesse escolher uma modelo de gestão de projetos, escolheria o VIPKS?” com uma média de 4. Esta pontuação possivelmente se deve ao fato de ser um modelo novo e ainda não ter sido utilizado em muitos projetos não possuindo ainda grande confiança por parte da equipe.

Tabela 10: Questionário – Questões genéricas

Questões - Questões genéricas	GP	*Design*	*Tester*	TL	Desenvol-vedor Junior	Desenvol-vedor Pleno	Média 4,26
O desenvolvimento com VIPKS aumentou a eficácia do desenvolvimento?	4	4	4	4	5	5	4,33
O desenvolvimento com VIPKS aumentou a qualidade do produto?	4	4	4	5	5	4	4,33
O desenvolvimento com VIPKS torna o trabalho mais organizado/planejado?	5	4	4	4	5	5	4,5
O desenvolvimento com VIPKS permite a detecção precoce de erros/defeitos?	4	3	5	4	5	4	4,16
Se pudesse escolher uma modelo de gestão de projetos, escolheria o VIPKS?	4	4	4	4	4	4	4

Fazendo uma análise de todos os resultados obtidos no questionário apresentado anteriormente, pode-se verificar que a média é superior a 4 em uma escala de 0 a 5, comprovando que a modelo híbrido VIPKS foi bem recebido por todos os envolvidos no desenvolvimento do projeto tendo uma maior pontuação na questão "Considera importante a criação do TAP (Termo de Abertura do Projeto)?" com nota máxima de todos os envolvidos, o que comprova a importância da documentação e da boa definição do âmbito no início dos projetos de desenvolvimento de *software*.

Porém, ainda existe algum receio em utilizar o modelo híbrido VIPKS uma vez que ele não foi usado em muitos projetos.

4.5 Comparação VIPKS vs Scrum vs Guia PMBOK®

Neste capítulo pretende-se fazer uma comparação entre os modelos de gestão de projetos utilizados no modelo VIPKS e o modelo. Para esta comparação foi utilizado os critérios utilizados no capítulo 3.2 (Tabela 3) no qual foi feito a comparação entre os modelos Scrum e Guia PMBOK®. No Anexo 2 podemos consultar a planilha de comparação entre os três modelos de gestão.

- Gerenciamento de projetos – o modelo VIPKS utiliza da junção do Guia PMBOK® e Scrum com foco no planejamento e no produto.
- Documentação – O VIPKS utiliza a documentação como uma parte do controle e em todas as etapas assim como o Guia PMBOK®, porém de forma simplificada e com uma linguagem informal como no Scrum.
- Controle – O modelo VIPKS é um meio termo entre o Guia PMBOK® e o Scrum. Une a comunicação constante do Scrum com uma parte da documentação do Guia PMBOK®. Porém feito de forma mais simplificada e leve.
- Conceito – Ao contrário do Guia PMBOK® e Scrum, o modelo VIPKS define o âmbito no início, servindo de base para a divisão dos *sprints*. Podemos existir mudanças no decorrer do projeto desde que seja negociado com o cliente o impacto no tempo e/ou custo.
- Mudanças – No VIPKS são consideradas, porém deve ser analisado o impacto de tempo e/ou custo e negociado com o cliente. Isto a difere do Guia PMBOK® que considera as mudanças como um risco para o sucesso do projeto e o Scrum que organiza os *sprints* e pagamentos de acordo com os *sprints*.
- Custos – O VIPKS, assim como o Guia PMBOK® estima os custos com base no âmbito e análise dos requisitos do projeto e não nos *sprints* como acontece no Scrum.
- Orientação – Assim como o Scrum, o VIPKS é orientado para as pessoas com indivíduos e não recursos como acontece no Guia PMBOK®:
- Burocracia – Neste ponto o modelo VIPKS utiliza a simplicidade utilizada no Scrum em que parte do princípio que realizar algo simples e útil é melhor do que gastar tempo em algo complexo que poderá não ser utilizado.
- Equipes – o modelo VIPKS foi criado com o intuito de atuar em pequenas e médias equipes multifuncionais com liderança do *team leader* para a parte de desenvolvimento e o gestor do projeto para o projeto como um todo unindo assim o que considera o melhor dos dois mundos do modelo Scrum que são equipes auto-organizáveis e dinâmicas com um pouco do controle e liderança utilizados no Guia PMBOK®.
- Ciclo de vida – o VIPKS utiliza um ciclo de vida interativo/incremental e adaptativo, baseado em *sprints* semelhantes ao modelo Scrum. É feito reuniões diárias de planejamento de

planejamento. Porém trata as mudanças como algo que acontece, mas que devem ser aceitas com aprovação e negociação do tempo e/ou custo junto ao cliente.

- Processo de gerenciamento de qualidade – a qualidade é tratada de forma diferente no VIPKS, unindo os *feedbacks* constante com o cliente ao longo do projeto e a qualquer momento do modelo Scrum com alguma documentação dos testes técnicos e funcionais do Guia PMBOK®. Porém, não está associado com as normas ISO como no Guia PMBOK® e sim em atender as necessidades do cliente e do projeto comprovando com documentos simples que o produto funciona como esperado.
- Comunicação – Assim como acontece no Scrum, a comunicação no modelo VIPKS é interpessoal, realizada em reuniões objetivas, rápidas e com pauta preestabelecidas.
- Cliente – Assim como no Scrum, no modelo VIPKS o cliente participa ativamente de todas as etapas do projeto.
- Fechamento – o modelo VIPKS reúne o processo de lições aprendidas do Scrum e uma parte da documentação utilizada no Guia PMBOK®.

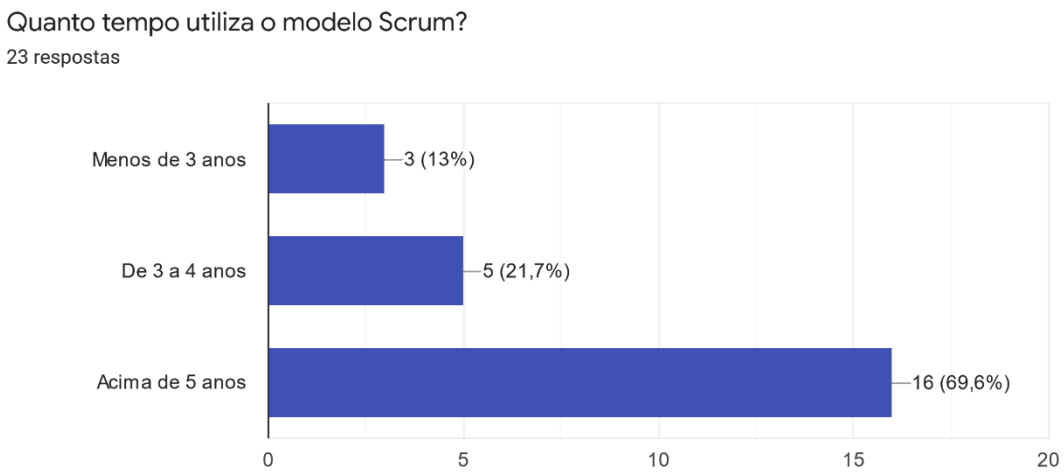
4.6 Comparação VIPKS vs Scrum

Para uma melhor análise do modelo VIPKS pretende-se fazer uma comparação entre o modelo ágil Scrum por ser o modelo mais utilizado atualmente pelas empresas segundo pesquisa feita pela *14th annual State of Agile* e por ser o modelo utilizado atualmente na empresa Virtual Innovations (objeto de estudo desta dissertação) e o modelo VIPKS (StateOfAgile, 2020).

Esta comparação foi feita através de questionário que foi respondido no mês de julho e agosto de 2021 de forma anônima por vinte e três pessoas que utilizam o modelo Scrum como principal ferramenta de gestão de projeto nas empresas que atuam em Portugal, ocupando eles cargos diversos nas empresas como: gestores de projetos, desenvolvedores de todos os níveis, *designer* e *tester*.

Na Figura 29 podemos observar que a maior parte das pessoas que responderam ao questionário possuem acima de 5 anos de experiência utilizando o modelo Scrum, sendo estes um percentual de 69,6% dos entrevistados.

Figura 29: Utilizadores do Scrum que responderam o "QUESTIONÁRIO Scrum"

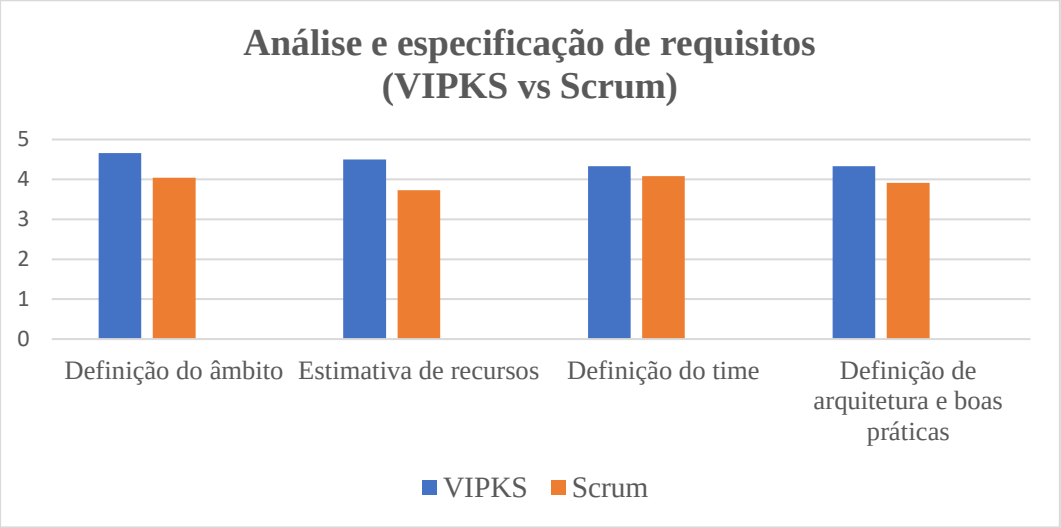


Fonte: Google formulário – QUESTIONÁRIO Scrum

Para esta comparação foi utilizado como inspiração o questionário utilizado no modelo VIPKS respondido pela equipe do projeto que podemos consultar no anexo 1 com suas respostas analisadas no capítulo 4.4.2 e foi feito uma comparação com o modelo Scrum. Será utilizado os mesmos parâmetros do anexo 1 com uma pontuação de zero a cinco, no qual zero é considerado fraco e cinco é considerado forte. Podemos visualizar esta comparação no anexo 4.

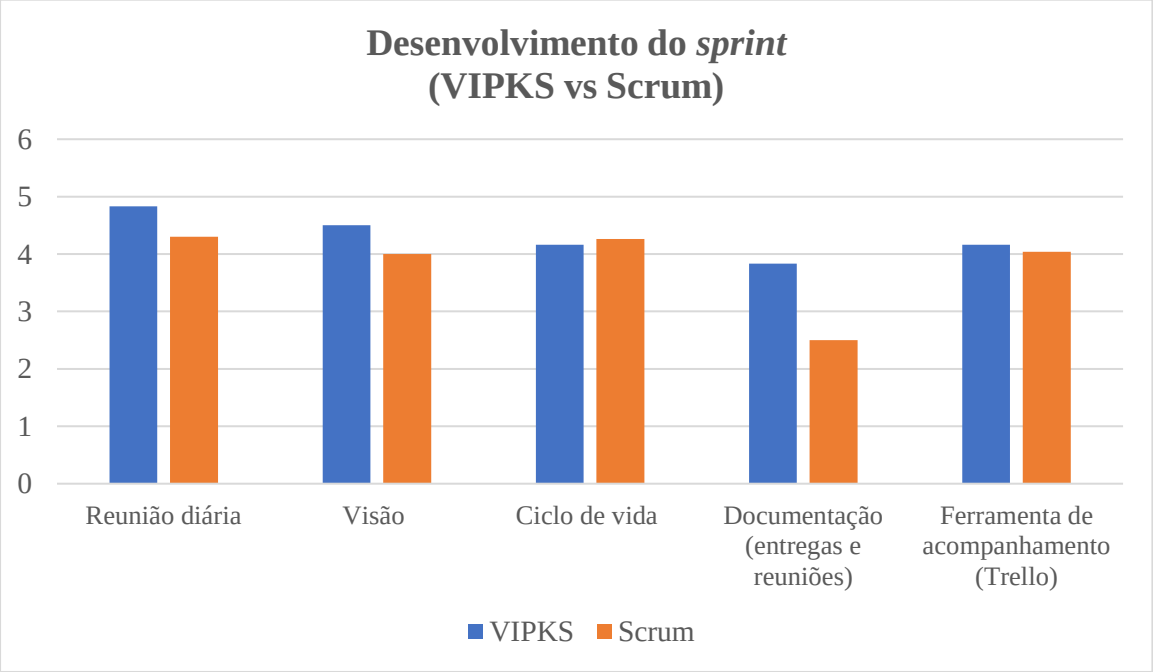
Na Figura 30, conseguimos constatar que não existe uma grande diferença entre as práticas de VIPKS e do Scrum na "Análise e especificação de requisitos" uma vez que a importância das definições do âmbito, recursos, time, arquitetura e boas práticas foram consideradas favoráveis em todas as respostas com uma pontuação média acima de 3,5. Isto deve-se ao fato de boa parte desta etapa no modelo híbrido VIPKS ter sido feito inspirado no Scrum.

Figura 30: Questões - Análise e especificação de requisitos (VIPKS vs Scrum)



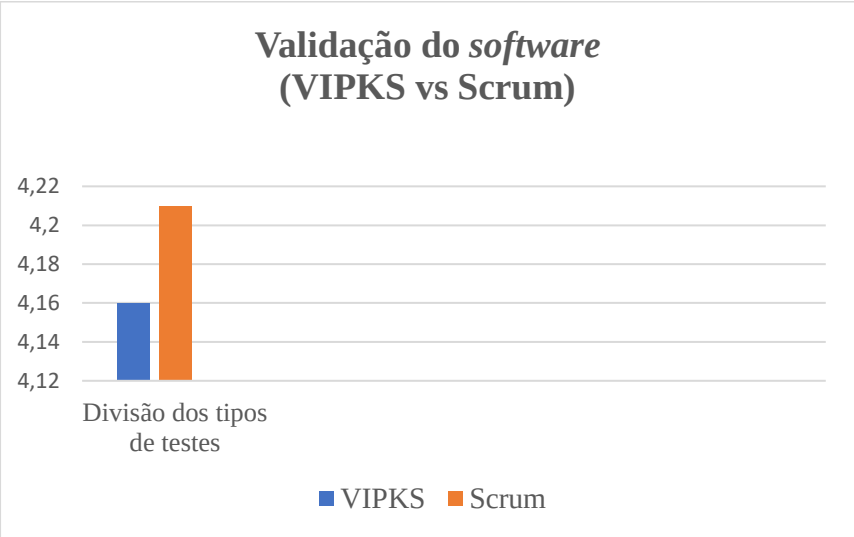
Na Figura 31, nas questões de "Desenvolvimento do *sprint*", um ponto a se observar é que na etapa em que foi questionado sobre se "Considera importante a documentação nas entregas e reuniões do *sprint*?" os usuários do Scrum não acreditam ser importante esta etapa. Provavelmente deve-se ao fato da documentação não ser considerada muito importante uma vez que o objetivo do Scrum é ser dinâmico e acredita-se que a elaboração de documentos mais detalhados sejam uma perca de tempo e mais trabalho.

Figura 31: Questões - Desenvolvimento do *sprint* (VIPKS vs Scrum)



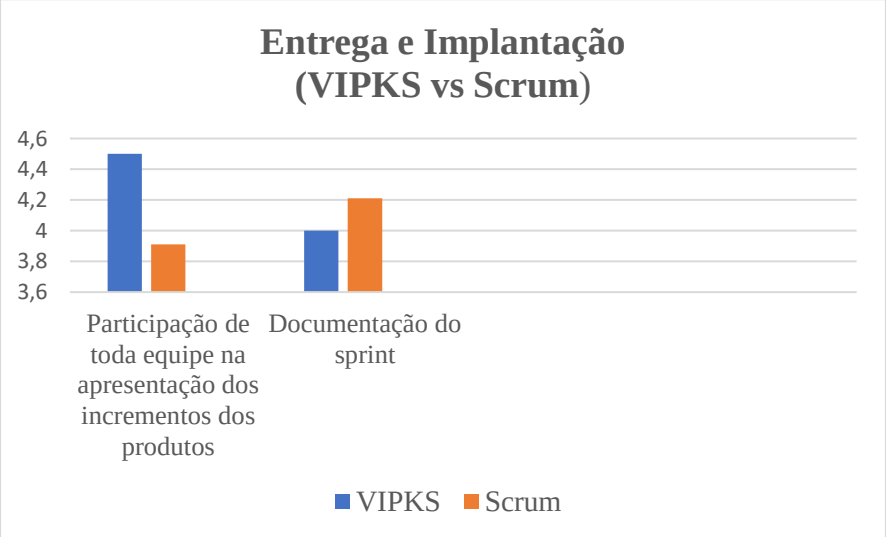
Na Figura 32, nas questões que se referem a “Validação do *Software*”, podemos constatar que os entrevistados estão mais confortáveis com os tipos de testes utilizados no Scrum uma vez que consideram que mais teste são trabalhos desnecessários uma vez que mudanças e alterações podem acontecer a todo momento no decorrer do projeto.

Figura 32: Questões - Validação do *software* (VIPKS vs Scrum)



Na Figura 33, em “Entrega e implantação”, existe uma diferença para a “Participação de toda equipe na apresentação dos incrementos do produto” no qual os usuários do modelo VIPKS acreditam ser muito importante com uma média de 4,5 e o Scrum não as considera muito importante com uma média inferior a 4. Apesar da diferença não ser tão grande, acredita-se que é pelo fato de não existir esta etapa no Scrum.

Figura 33: Questões - Entrega e implantação (VIPKS vs Scrum)



4.7 Considerações do modelo VIPKS

O modelo híbrido VIPKS possui uma essência ágil, complementado por algumas das melhores práticas do Guia PMBOK®. Com isto, podemos concluir que o modelo híbrido VIPKS passou a valorizar a documentação e a comunicação entre o cliente e os envolvidos no projeto dando voz a todos de forma igualitária.

Podemos citar como pontos principais do modelo híbrido VIPKS:

- Fase inicial – Esta fase é a principal fase para o sucesso do projeto. Nesta fase é importante a aproximação entre a equipe de desenvolvimento e o cliente para a especificação do projeto da melhor forma possível, produzindo o termo de abertura do projeto com as suas especificações funcionais e não funcionais e o design inicial do produto.
- Documentação – Todas as etapas do modelo híbrido VIPKS passa a ter um documento, mesmo que de forma simplificada. Isto garante que todos os envolvidos possuam a mesma visão do projeto. A criação do termo de abertura do projeto permite que toda equipe e o cliente estejam em sintonia e concordância com o que esperam do *software*. Assim como na visão do *sprint*, ajudando a equipe de desenvolvimento a justificar as opções tomadas na implementação.
- Comunicação – As reuniões diárias, de final do *sprint* e entrega envolvendo todos da equipe é essencial para que todos estejam de acordo com os acontecimentos e mudanças e minimizando as falhas nas comunicações.
- Ferramenta de acompanhamento do projeto – o uso da ferramenta em quadro para acompanhamento do projeto facilita a visualização e acompanhamento do projeto por toda equipe e o cliente estando todos cientes do seu andamento em tempo real.
- Fase final - A fase final do projeto, assim como o seu início, é fundamental para o seu sucesso. Por isto dedicou-se a eles um *sprint* e teve a participação de toda a equipe valorizando todos os envolvidos no projeto.

Para validação do modelo híbrido VIPKS, foi feito a implementação em um projeto piloto na empresa Virtual Innovations, envolvendo uma equipe de seis pessoas que já utilizavam os modelos ágeis nos projetos.

Para avaliar o modelo híbrido VIPKS a equipe preencheu um questionário com o qual se pretende perceber se o modelo é adequado para o planejamento de projetos de desenvolvimento de *software*.

Uma vez que o modelo híbrido VIPKS usou como inspiração os modelos ágeis Scrum/Kanban e os modelos clássicos orientado pelo Guia PMBOK®, foi feito uma comparação entre eles e o VIPKS.

Porém, os resultados podem ser questionáveis, pois apesar de ter sido feito uma pesquisa através de questionário sobre o modelo Scrum, os resultados da sua comparação dependem muito da subjetividade do autor que a efetua.

Podemos concluir que o modelo híbrido VIPKS se revelou benéfico para os projetos de desenvolvimento de *software* unindo os benefícios do modelo ágil com as práticas de documentação dos modelos clássicos, sendo importante a criação do TAP que foi a parte que reuniu unanimidade na utilização do modelo e mais documentação e controle no decorrer do projeto. Porém, requer-se que o modelo seja utilizado em mais projetos de diversos tamanhos para verificar se os resultados são favoráveis.

4.8 Sumário

Neste capítulo, foi criado e validado o modelo híbrido VIPKS para gestão de projetos de desenvolvimento de *software*. Este modelo foi constituído utilizando os modelos ágeis Kanban e Scrum e o modelo clássico orientado pelo Guia PMBOK®.

Foi estabelecido os valores, eventos, artefatos, estrutura organizacional e responsabilidades do modelo VIPKS. Assim como o seu funcionamento, que foi divido por etapas de acordo com o ciclo de vida de um projeto de desenvolvimento de *software*.

Também foi feito a validação do modelo híbrido VIPKS em um projeto piloto na empresa Virtual Innovations através de um investigação-ação experimental em um caso real e posteriormente foi feito uma análise da implementação do modelo VIPKS através de questionário com toda equipe de desenvolvimento e uma avaliação do questionário.

Após a validação do modelo VIPKS foi feito uma comparação entre os modelos utilizados no projeto piloto (Kanban, Scrum e Guia PMBOK®) e uma comparação entre o modelo VIPKS e o Scrum através de questionário sobre o Scrum respondido de forma anônima por usuários do Scrum.

Após estas análises, foi feito as considerações sobre o VIPKS e foi citado os pontos principais utilizados no modelo.

No próximo capítulo pretende-se fazer uma reflexão sobre toda a dissertação e será realizado uma avaliação relativamente ao cumprimento dos objetivos propostos e sugestões para melhoria e trabalhos futuros.

5 Conclusões e Trabalhos Futuros

A utilização dos modelos híbridos nos projetos de desenvolvimento de *software* que utilizam as boas práticas dos modelos clássicos com âmbitos bem-definidos e as boas práticas dos modelos ágeis com entrega constante do produto e interações entre as pessoas envolvidas, vem se tornando cada vez mais eficaz e usual na gestão de projetos de desenvolvimento de *softwares*.

Observou-se que os modelos ágeis são amplamente utilizados nas empresas de desenvolvimento de *software.* O modelo ágil mais utilizado nas empresas é o Scrum com 58%, segundo a 14ª pesquisa anual ágil realizada em 2020. A mesma pesquisa, também constatou que o Scrum é utilizado em conjunto com outros modelos, criando assim modelos híbridos como por exemplo o ScrumBan, Scrum/XP, etc.

Com o intuito de solucionar alguns dos problemas existentes na empresa na qual o autor exerce a sua atividade profissional, o autor se desafiou a estudar diversos modelos de gestão de projetos para criar seu próprio modelo híbrido, oriundo de alguns modelos ágeis e clássicos para solucionar os problemas que a empresa possui na gestão do âmbito dos projetos, déficit de documentação e falta de processos no trabalho das equipes envolvidas nos projetos.

Após este estudo sobre quais são os modelos de gestão de projetos mais utilizados no desenvolvimento de *software*, permitiu-se a comparação entre estes modelos.

Esta comparação permite que outros profissionais possam utilizar esta pesquisa para adotarem seus próprios modelos de gestão de projetos que vão de acordo com as suas necessidades uma vez que o modelo híbrido VIPKS foi criado com a combinação dos modelos ágeis Scrum e Kanban e o modelo clássico orientado pelo Guia PMBOK® para atender necessidades encontrados na empresa que o autor atua.

O modelo híbrido VIPKS tem uma dinâmica ágil e foi criado de acordo com o ciclo de vida de um projeto de desenvolvimento de *software.*

Podemos destacar como pontos principais do modelo VIPKS a fase inicial com a produção do termo de abertura do projeto, as especificações funcionais e não funcionais do projeto e a criação do *design.* Estes pontos, são executados sempre valorizando a documentação e a utilização da ferramenta de gestão de projetos em quadro para acompanhamento de todos os envolvidos no projeto. Também foi valorizado a participação de toda a equipe do projeto em todas as etapas, evitando falhas nas comunicações e dando voz a todos de forma igualitária.

A participação de toda equipe na apresentação do produto final ao cliente é uma prática importante defendidas pelo VIPKS e é algo que motivou os envolvidos no projeto, pois permitiu que tanto os que efetuam a codificação, como aqueles que executam os testes, se sentissem valorizados.

Este aspecto incentivou o trabalho em conjunto, para encontrarem a melhor solução, existindo assim um compromisso mútuo.

O modelo VIPKS vem sendo utilizado com frequência nos projetos da empresa em que o autor atua, uma vez que se observou um ganho de valor na entrega dos projetos, dando-se mais valor e esclarecimento ao seu âmbito, melhor organização e orientação aos envolvidos nos projetos.

Para o desenvolvimento desta dissertação, foram utilizados os relatórios produzidos pela empresa VersionOne para perceber a realidade de utilização dos modelos ágeis. Neste sentido, sugere-se a realização de um estudo em Portugal sobre a adoção dos modelos ágeis na gestão de projetos de *software*.

Com base nos resultados deste trabalho, o modelo VIPKS pode ser utilizado como base para novos estudos sobre os fatores de sucesso em instituições de sistema da informação, considerando diferentes tipos de projetos e profissionais com perfis distintos, aumentando assim a abrangência de investigação e a melhoria do modelo podendo perceber se outras organizações optam por criar os seus modelos híbridos ou aprimorar e adaptar os valores, eventos e artefatos do modelo VIPKS, para que o modelo proposto se torne mais robusto e aplicável a um maior número de cenários.

Como estudo futuro, propõe-se a utilização do modelo VIPKS na filial da empresa no Brasil e em projetos maiores de longa duração. Recomenda-se também que sejam feitos estudos em outros projetos da empresa em diferentes áreas que utilizam os modelos ágeis para o gerenciamento de projetos, a fim de averiguar se os resultados se repetem em diferentes contextos.

REFERÊNCIA

Abrahamsson, P., Warsta, J., Siponen, M. T., & Ronkainen, J. (2003). New directions on agile methods: A comparative analysis. *Proceedings - International Conference on Software Engineering*, *3–5*(June), 244–254. https://doi.org/10.1109/icse.2003.1201204

Baldissera, A. (2012). Pesquisa-Ação: Uma Metodologia Do "Conhecer" E Do "Agir" Coletivo. *Sociedade Em Debate*, *7*(2), 5-25–25.

Barry Boehm, R. T. (2003). *Balancing Agility and Discipline: A Guide for the Perplexed*. (A.-W. Professional, Ed.) (1ª). Canada.

Begel, A., & Nagappan, N. (2007). Usage and Perceptions of Agile Software Development in an Industrial Context: An Exploratory Study. *Proceedings - 1st International Symposium on Empirical Software Engineering and Measurement, ESEM 2007*, (October 2007), 11. https://doi.org/10.1109/ESEM.2007.12

Bomfin, D. F., Nunes, P. C. de Á., & Hastenreiter, F. (2012). Gerenciamento de Projetos Segundo o Guia PMBOK: Desafios para os Gestores. *Revista de Gestão e Projetos*, *3*(3), 58–87. https://doi.org/10.5585/gep.v3i3.78

Bruce Maxim, R. S. P. (2014). *Software Engineering: A Practitioner's Approach* (8ª).

Bruno, L. (2019). FATORES DE DISSEMINAÇÃO DE MÉTODOS ÁGEIS EM EMPRESAS DE TI. *Journal of Chemical Information and Modeling*, *53*(9), 1689–1699. https://doi.org/10.1017/CBO9781107415324.004

Camargo, R. (2019). Conheça as 10 áreas de conhecimento do PMBOK® - 16.06.2019. São Paulo. Retrieved from https://robsoncamargo.com.br/blog/Areas-de-conhecimento-do-PMBOK

Cockburn, A. (2003). People and methodologies in software development. *Faculty of Mathematics and Natural Sciences*, 1–89.

Coimbra. (2020). Agile Methods: DSDM. Retrieved January 5, 2021, from https://projetoseti.com.br/agile-methods-dsdm/

Couto, C., Moreira, T., Pedrosa, G., Augusto, B., Roberto, D., & Balbino, P. (2017). Belo Horizonte - 2017.

CRUZ, F. (2013). Scrum e PMBOK unidos no Gerenciamento de Projetos, *4*, 416. Retrieved from https://books.google.com/books?id=SJA37S2QGR0C&pgis=1

Eder, S., Conforto, E. C., Amaral, D. C., & Silva, S. L. da. (2014). Diferenciando as abordagens tradicional e ágil de gerenciamento de projetos. *Production*, *25*(3), 482–497. https://doi.org/10.1590/s0103-65132014005000021

Espinha, R. G. (2020). Ciclo de Vida de um Projeto.

Fonseca, K. H. (2012). Investigação – Ação : Uma Metodologia Para Prática E Reflexão Docente. *Revista Onis Ciência*, 16–31.

Fowler, M. (2001). The new methodology. *Wuhan University Journal of Natural Sciences*, 12–24.

Francischini, R., & Crist, F. (2016). MODELOS HÍBRIDOS DE GESTÃO DE PROJETOS COMO ESTRATÉGIA NA CONDUÇÃO DE SOLUÇÕES EM CENÁRIOS DINÂMICOS E COMPETITIVOS. *Revista Brasileira de Gestão e Desenvolvimento Regional*, *v. 12*, n.*3*, 443–457.

Franzen, E., & Lutz, D. (2018). Implantação De Novouma Fábrica De Software Baseado Nos Modelos, 214–234.

Guia PMBOK®. (2017) (6ª Edição). Project Management Institute, Inc. 14 Campus Boulevard Newtown Square, Pensilvânia 19073-3299 EUA Fone: +1 610-356-4600 Fax: +1 610-356-4647 E-mail: customercare@pmi.org Website: www.PMI.org ©2017. Retrieved from www.PMI.org

Hislop, G. W., Lutz, M. J., Naveda, J. F., McCracken, W. M., Mead, N. R., & Williams, L. A. (2002). Integrating agile practices into software engineering courses. *International Journal of Phytoremediation*, *21*(1), 169–185. https://doi.org/10.1076/csed.12.3.169.8619

https://trello.com. (n.d.). Retrieved from https://trello.com/

https://www.zoho.com. (n.d.). Retrieved from https://www.zoho.com

J. J. Sutherland (Autor), N. L. (Tradutor). (2020). *SCRUM: guia prático*. (E. Sextante, Ed.) (1ª).

Juliani, D. P., Juliani, J. P., Alves Bello, J. D. S., & De Souza, J. A. (2012). Modelo para Construção de Base de Conhecimentos sobre Projetos Suportado por Ferramentas Colaborativas. *Revista de Gestão e Projetos*, *3*(3), 277–290. https://doi.org/10.5585/gep.v3i3.128

Kumar, G., & Bhatia, P. (2012). Impact of Agile methodology on software development process. *International Journal of Computer Technology and Electronics Engineering (IJCTEE)*, *2*(4), 46–50.

Lutfiani, N., Harahap, E. P., Aini, Q., Ahmad, A. D. A. R., & Rahardja, U. (2020). Inovasi Manajemen Proyek I-Learning Menggunakan Metode Agile Scrumban. *InfoTekJar: Jurnal Nasional Informatika Dan Teknologi Jaringan*, *5*(1), 96–101.

Mahdi JAVANMARD, M. A. (2015). Comparison between Agile and Traditional Software Development Methodologies. *The University of Western Australia*, *36*(3), 9. https://doi.org/10.1145/130840.130843

Márques-Vargas, L. (2016). Project agile management for software development: A comparative study on the applicability of Scrum together with PMBOK and / or Prince 2. *Revista de Gestão e Projetos*, *7*(3).

Meirinhos, M. & O. A. (2010). Educação O estudo de caso como estratégia de investigação em educação The case study as research strategy in education. *Revista de Educação*, *2*(2), 49–65.

Moniruzzaman, A. B. M., Hossain, D. S. A., Gautam, S., & Kumar, D. M. (2013). Comparative Study on Agile software development methodologies, *3*(02), 158–164. Retrieved from http://arxiv.org/abs/1307.3356

Neto, E. M. da C. (2018). Gerenciamento Ágil de Projetos de Desenvolvimento de Software : Uma Alternativa ao Gerenciamento Tradicional. Salvador, Brasil.

Norma NBR ISO/IEC 12207. (1998).

Ohno, T. (1997). *O Sistema Toyota de Produção por Taiichi Ohno*.

Portillo, C. A. (2010). Gerenciamento eficaz do escopo do projeto. *Livraria Virtual PMI*, 4.

Portugal, A. (2020). Waterfall e Agile: Vantagens e Desvantagens dessas Metodologias. Retrieved October 6, 2021, from https://gobacklog.com/blog/waterfall-ou-agile/

Qasaimeh, M., Mehrfard, H., & Hamou-Lhadj, A. (2008). Comparing agile software processes based on the software development project requirements. *2008 International Conference on Computational Intelligence for Modelling Control and Automation, CIMCA 2008*, 49–54. https://doi.org/10.1109/CIMCA.2008.54

Rasnacis, A., & Berzisa, S. (2016). Method for Adaptation and Implementation of Agile Project Management Methodology. *Procedia Computer Science*, *104*(December 2016), 43–50. https://doi.org/10.1016/j.procs.2017.01.055

Rebaiaia, M.-L., & Rodrigues Vieira, D. (2014). Integrating PMBOX Standards, Lean and Agile Methods in Project Management Activities. *International Journal of Computer Applications*, *88*(4), 40–46. https://doi.org/10.5120/15343-3680

Santos, F. A. D. O., Santos, M. F. O. M., Reis, E. M., & Costa, A. D. O. (n.d.). Metodologias Híbridas de Desenvolvimento de Software : Uma Opção Viável para Gestão de Projetos.

Sharma, K., & Aggarwal, D. H. (2013). Review of Agile Software Development Methodologies. *International Journal of Advanced Research in Computer Science and Software Engineering*, *3*(2), 2277. Retrieved from https://pdfs.semanticscholar.org/09a8/693d0a6a7af4f397906755f6ae778b07f9d9.pdf

Silva, D. E. dos S., Souza, I. T. de, & Camargo, T. (2013). METODOLOGIAS ÁGEIS PARA O DESENVOLVIMENTO DE SOFTWARE: APLICAÇÃO E O USO DA METODOLOGIA SCRUM EM CONTRASTE AO MODELO TRADICIONAL DE GERENCIAMENTO DE PROJETOS. *Computação Aplicada*, *v.2*, n.*1*.

SOARES, B. D. S. (2009). Comparação entre Metodologias Ágeis e Tradicionais para o Desenvolvimento de Software. *Idesia*, *27*(2), 6. https://doi.org/10.4067/s0718-34292009000200002

Soares, M. (2004). Comparação entre Metodologias Ágeis e Tradicionais para o Desenvolvimento de

Software. *INFOCOMP Journal of Computer Science*, *3*(2), 8–13.

Soares, M. D. S. (2004). Metodologias Ágeis Extreme Programming e Scrum para o Desenvolvimento de Software. *Revista Eletrônica de Sistemas de Informação*, *3*(1). https://doi.org/10.21529/resi.2004.0301006

Sommerville, I. (2003). Engenharia de Software. In Pearson (Ed.), *Engenharia de Software* (6ª edição, p. 592p).

Souza, K. P. de, & Gasparotto, A. M. S. (2013). A IMPORTÂNCIA DA ATIVIDADE DE TESTE NO DESENVOLVIMENTO DE SOFTWARE. *XXXIII ENCONTRO NACIONAL DE ENGENHARIA DE PRODUCAO*. Salvador, BA, Brasil.

Souza, J. (2018). Estudo comparativo das Metodologias àgeis e PMBOK. Portugal: 2018.

StateOfAgile. (2020). 14th annual STATE OF AGILE REPORT. *Annual Report for the STATE OF AGILE*, *14*(14), 2–19. Retrieved from https://explore.digital.ai/state-of-agile/14th-annual-state-of-agile-report%0Ahttps://stateofagile.com/#

STOICA, M., MIRCEA, M., & GHILIC-MICU, B. (2013). Software Development: Agile vs. Traditional. *Informatica Economica*, *17*(4/2013), 64–76. https://doi.org/10.12948/issn14531305/17.4.2013.06

Strode, D., & Hunt, P. (2006). Métodos ágeis : uma análise comparativa, 257–264.

Strode, D., & Strode, D. (2016). Agile methods : a comparative analysis Agile methods : a comparative analysis, (January 2006), 257–264. Retrieved from www.naccq.ac.nz

Taroco, B. A., & Werner[1], C. (2015). Análise Comparativa Entre As Metodologias De Desenvolvimento Tradicionais E Ágeis.

The Standish Group International, I. (2015). CHAOS Report 2015. *CHAOS Report 2015*, 13. Retrieved from https://www.standishgroup.com/sample_research_files/CHAOSReport2015-Final.pdf

Thielmann, R., & Rodrigues, Gustavo Alves, Raphael Lima, R. P. (2015). ANÁLISE E COMPARAÇÃO DO KANBAN TRADICIONAL E VARIAÇÕES : UM ESTUDO DE CASO SOBRE Introdução e Objetivos. *XI CONGRESSO NACIONAL DE EXCELÊNCIA EM GESTÃO*, (1984–9354), 19.

Tomás, M. R. Métodos ágeis: características, pontos fortes e fracos e possibilidades de aplicação, No. WPS09/ IET Working Papers Series (2009).

Trigo, R. A., & Barreto, L. C. (2019). EVOLUÇÃO DOS MÉTODOS DE DESENVOLVIMENTO DE SOFTWARE EM MICROEMPRESAS. *Revista Gestão Em Foco*, *Edição nº*, 114–122.

Vargas, L. M. (2016). Gerenciamento Ágil de Projetos em Desenvolvimento de Software: Um Estudo Comparativo sobre a Aplicabilidade do Scrum em Conjunto com PMBOK e/ou PRINCE2. *Revista de Gestão e Projetos*, *07*(03), 48–60. https://doi.org/10.5585/gep.v7i3.398

VersionOne. (2016). 10th Anual State of Agil Report. *UN Millennium Development Library: Investing in Strategies to Reverse the Global Incidence of TB*, 16. https://doi.org/10.4324/9781849773546-9

ANEXO 1 - QUESTIONÁRIO VIPKS

O objetivo deste questionário é validar o modelo híbrido VIPKS. As perguntas foram divididas de acordo com o ciclo de vida do projeto (Análise e especificação de requisitos; Desenvolvimento; Validação do software, Entrega e Implementação) e Questões genéricas.

*Obrigatório

1. Seu nome e qual função você desempenhou utilizando o modelo VIPKS? *

2. Análise e especificação de requisitos - Considera importante a criação do TAP (Termo de Abertura do Projeto)? *

Marcar apenas uma oval.

	1	2	3	4	5	
Fraco	◯	◯	◯	◯	◯	Forte

3. Análise e especificação de requisitos - Considera importante a definição do âmbito? *

Marcar apenas uma oval.

	1	2	3	4	5	
Fraco	◯	◯	◯	◯	◯	Forte

4. Análise e especificação de requisitos - Considera importante a estimativa dos recursos? *

Marcar apenas uma oval.

	1	2	3	4	5	
Fraco	◯	◯	◯	◯	◯	Forte

5. Análise e especificação de requisitos - Considera importante a definição do time do projeto? *

Marcar apenas uma oval.

	1	2	3	4	5	
Fraco	◯	◯	◯	◯	◯	Forte

6. Análise e especificação de requisitos - Considera importante a definição da arquitetura e das boas práticas? *

Marcar apenas uma oval.

	1	2	3	4	5	
Fraco	◯	◯	◯	◯	◯	Forte

7. Desenvolvimento - Considera importante as reuniões diárias? *

Marcar apenas uma oval.

	1	2	3	4	5	
Fraco	◯	◯	◯	◯	◯	Forte

8. Desenvolvimento - Considera importante a visão do sprint? *

Marcar apenas uma oval.

	1	2	3	4	5	
Fraco	◯	◯	◯	◯	◯	Forte

9. Desenvolvimento - Considera importante o ciclo de vida do sprint? *

Marcar apenas uma oval.

	1	2	3	4	5	
Fraco	◯	◯	◯	◯	◯	Forte

10. Desenvolvimento - Considera importante a documentação (entregas e reuniões) dos sprints? *

Marcar apenas uma oval.

	1	2	3	4	5	
Fraco	◯	◯	◯	◯	◯	Forte

11. Desenvolvimento - A ferramenta de acompanhamento do projeto escolhido (Trello) ajudou a organizar os sprints? *

Marcar apenas uma oval.

	1	2	3	4	5	
Fraco	◯	◯	◯	◯	◯	Forte

12. Validação do software - Considera importante como foi dividido os tipos de testes? *

Marcar apenas uma oval.

	1	2	3	4	5	
Fraco	◯	◯	◯	◯	◯	Forte

13. Validação do software - A validação com o modelo VIPKS permite antecipar e corrigir falhas e bugs que apareceriam para o usuário final? *

Marcar apenas uma oval.

	1	2	3	4	5	
Fraco	◯	◯	◯	◯	◯	Forte

14. Entrega e Implantação - Participação de toda equipe na apresentação dos incrementos dos produtos finalizados e prontos foi uma prática importante? *

Marcar apenas uma oval.

	1	2	3	4	5	
Fraco	◯	◯	◯	◯	◯	Forte

15. Entrega e Implantação- Considera importante a documentação de entrega e implementação do sprint? *

Marcar apenas uma oval.

	1	2	3	4	5	
Fraco	◯	◯	◯	◯	◯	Forte

16. Questões genéricas - O desenvolvimento com VIPKS aumentou a eficácia do desenvolvimento? *

Marcar apenas uma oval.

	1	2	3	4	5	
Fraco	◯	◯	◯	◯	◯	Forte

17. Questões genéricas - O desenvolvimento com VIPKS aumentou a qualidade do produto? *

Marcar apenas uma oval.

	1	2	3	4	5	
Fraco	◯	◯	◯	◯	◯	Forte

18. Questões genéricas - O desenvolvimento com VIPKS torna o trabalho mais organizado/ planejado? *

Marcar apenas uma oval.

	1	2	3	4	5	
Fraco	◯	◯	◯	◯	◯	Forte

19. Questões genéricas - O desenvolvimento com VIPKS permite a deteção precoce de erros / defeitos? *

Marcar apenas uma oval.

	1	2	3	4	5	
Fraco	◯	◯	◯	◯	◯	Forte

20. Questões genéricas - Se pudesse escolher uma modelo de gestão de projetos, escolheria o VIPKS? *

Marcar apenas uma oval.

	1	2	3	4	5	
Fraco	◯	◯	◯	◯	◯	Forte

ANEXO 2 – COMPARAÇÃO VIPKS VS SCRUM VS GUIA PMBOK®

Abordagens	VIPKS	Guia PMBOK®	Scrum
Gerenciamento de projetos	- Foco no planejamento e produto.	- Foco no planejamento.	- Foco no produto.
Documentação	- Produzida em todas as etapas; - Forma simplificada; - Instrumento de controle; - Linguagem amigável.	- Principal instrumento de controle; - Toda etapa do projeto deve ser documentada. Sendo este um fator impeditivo para a mudança de etapa; - Extensa e desenvolvida pelo gerente de projetos.	- Produzida conforme os ciclos são realizados; - O foco do trabalho está na implementação do produto com qualidade, a documentação é secundária; - Existe uma flexibilidade na produção; - Simples, resumida e com linguagem amigável.
Controle	- No desenvolvimento do produto e através de documentação e comunicação constante.	- Necessita de um número maior de certezas para estabelecer as métricas de trabalho (custo, dimensionamento de equipe e tempo).	- Trabalha com incertezas e com priorizações durante o desenvolvimento do produto.
Conceito	- Âmbito definido no início que servirá de base para a divisão dos *sprints*.	- Âmbito claro e detalhado do que deverá ser realizado no projeto; - Devem ser descritos os elementos principais que servem de base para as estimativas de custo e tempo.	- Abstração de funcionalidades, estabelecendo um âmbito inicial que servirá de base para o detalhamento no decorrer dos ciclos de vida.
Mudanças	- São consideradas, porém deve ser analisado o	- São consideradas riscos ao sucesso do projeto,	- São consideradas inerentes ao processo de

	impacto de tempo e/ou custo e negociado com o cliente.	não são impedidas, mas são permitidas apenas após análise de impacto e custos, os benefícios forem superiores aos custos de alteração de âmbito.	amadurecimento da equipe e devem ser implementadas no próximo ciclo possível.
Custos	- São estimados com base do âmbito e análise dos requisitos do projeto; - A inexistência do âmbito inviabiliza a estimativa de custo do projeto.	- São estimados com base do âmbito e análise dos requisitos do projeto; - Existem processos específicos que gerenciam os custos do projeto; - A inexistência do âmbito inviabiliza a estimativa de custo do projeto.	- Os custos são estimados através do *product backlog*, agilizando o início da execução pois os detalhamentos são realizados a cada iteração; - Não há um processo específico que trate do assunto.
Orientação	- A pessoas: as pessoas são tratadas como indivíduos e não recursos. - O ritmo de trabalho é respeitado, cada membro da equipe escolhe quais e estimam o tempo a ser gasto das tarefas que irá desenvolver no próximo *sprint*.	- A processos: processos bem definidos devem ser executados obrigatoriamente para garantir o desenvolvimento de qualidade; - O gerente deve distribuir as atividades e montar o cronograma conforme os talentos das pessoas envolvidas no projeto.	- A pessoas: as pessoas são tratadas como indivíduos e não recursos. - O ritmo de trabalho é respeitado, cada membro da equipe escolhe quais e estimam o tempo a ser gasto das tarefas que irá desenvolver no próximo *sprint*.
Burocracia	- Simplicidade: parte do princípio que realizar algo simples e útil é melhor do	- Burocrática: exigem o desenvolvimento da solução completa, gera	- Simplicidade: parte do princípio que realizar algo simples e útil é

	que gastar tempo em algo complexo que poderá não ser utilizado.	sobrecarga da equipe e compromete a velocidade de desenvolvimento.	melhor do que gastar tempo em algo complexo que poderá não ser utilizado.
Equipes	- Pequenas e médias, multifuncionais; - Liderança designada pelo *team leader* e gestor do projeto; - Auto-organizáveis.	- Grandes, selecionados conforme a estrutura da empresa; - Há obrigatoriamente um gerente de projetos como líder; - Os membros são alocados no projeto conforme a necessidade organizacional, podendo ter dedicação parcial ou total dos envolvidos, além de ser modificada durante a execução do projeto; - Não há restrição quanto a formação de equipes geograficamente separadas.	- Pequenas, multifuncionais, sem distinção de títulos profissionais, e, enquanto for economicamente viável, devem permanecer iguais durante todo o projeto; - Não há liderança designada, são auto-organizáveis; - O *Scrummaster* e o *product owner* não são considerados no número de membros da equipe pois podem participar de mais de um projeto.
Ciclo de vida	- Interativo/incremental e adaptativo, baseado em *sprints*; - As reuniões de planejamento e revisão da *sprint* definem o detalhamento do âmbito do período; - Mudanças são aceitas com aprovação e	- São definidos três modelos de ciclo de vida: preditivo, interativo/incremental e adaptativo; - Os dois últimos são mais utilizados e permitem o planejamento detalhado em ondas, o âmbito é planejado em uma visão macro no início do	- Ciclo de vida baseado em *sprints*, o projeto tem seu âmbito detalhado ao longo do desenvolvimento; - As reuniões de planejamento e revisão da *sprint* definem o detalhamento do âmbito do período e as possíveis

	negociação do tempo e/ou custo.	projeto e pormenorizado conforme as etapas de desenvolvimento se repetem.	mudanças necessárias.
Processo de gerenciamento de qualidade	- Está em atender o cliente através do cumprimento dos requisitos e do *feedback* constante do cliente ao longo do projeto; - São feitos testes técnicos e funcionais no final de cada *sprint* e documentados para garantir a qualidade do produto.	- Deve ser aplicado a todo o projeto e garantir que os requisitos sejam cumpridos e validados com a melhor qualidade possível; - Os padrões de qualidade devem ser compatibilizados com as normas ISO.	- Está em atender o cliente através do cumprimento dos requisitos e do *feedback* constante do cliente ao longo do projeto; - A cada ciclo são retornados os pontos de melhoria que devem ser trabalhados o mais breve possível; - O foco está no produto com a melhor qualidade possível.
Comunicação	- Interpessoal, realizada nas reuniões de planejamento e revisão do *sprint*, e reuniões diárias; - São reuniões objetivas, rápidas e com pauta preestabelecida.	- Planejada através da identificação das necessidades das partes e determinação dos meios adequados; - Devem ser gerados documentos formais de comunicação para registrar e divulgar as conclusões as partes interessadas.	- Interpessoal, realizada nas reuniões de planejamento e revisão do *sprint*, e reuniões diárias; - São reuniões objetivas, rápidas e com pauta preestabelecida.
Cliente	- Participação ativa do cliente como parte do projeto.	- O cliente está presente em poucos momentos de planejamento do âmbito ou de gerenciamento de	- Participação ativa do cliente como parte do projeto.

		riscos.	
Fechamento	- O processo de retomada das lições aprendidas é realizado ao final de cada *sprint* durante a reunião de retrospectiva. - Ao final do projeto é realizada reunião para arquivamento de documentos e retomada das lições aprendidas durante todo o projeto.	- Ao final do projeto é realizada reunião para arquivamento de documentos e retomada das lições aprendidas durante o projeto; - Gerenciamento de aquisições.	- O processo de retomada das lições aprendidas é realizado ao final de cada *sprint* durante a reunião de retrospectiva.

ANEXO 3 - QUESTIONÁRIO Scrum

O objetivo deste questionário é conhecer a opinião de pessoas que utilizam este modelo na empresa Virtual Innovations. O questionário será respondido de forma anônima.

*Obrigatório

1. Quanto tempo utiliza o modelo Scrum? *

Marque todas que se aplicam.

- [ ] Menos de 3 anos
- [ ] De 3 a 4 anos
- [ ] Acima de 5 anos

2. Análise e especificação de requisitos - Considera importante a definição do âmbito? *

Marcar apenas uma oval.

	1	2	3	4	5	
Fraco	◯	◯	◯	◯	◯	Forte

3. Análise e especificação de requisitos - Considera importante a estimativa dos recursos? *

Marcar apenas uma oval.

	1	2	3	4	5	
Fraco	◯	◯	◯	◯	◯	Forte

4. Análise e especificação de requisitos - Considera importante a definição do time do projeto? *

Marcar apenas uma oval.

	1	2	3	4	5	
Fraco	◯	◯	◯	◯	◯	Forte

5. Análise e especificação de requisitos - Considera importante a definição da arquitetura e das boas práticas? *

Marcar apenas uma oval.

	1	2	3	4	5	
Fraco	◯	◯	◯	◯	◯	Forte

6. Desenvolvimento - Considera importante as reuniões diárias? *

Marcar apenas uma oval.

	1	2	3	4	5	
Fraco	◯	◯	◯	◯	◯	Forte

7. Desenvolvimento - Considera importante a visão do sprint? *

Marcar apenas uma oval.

	1	2	3	4	5	
Fraco	◯	◯	◯	◯	◯	Forte

8. Desenvolvimento - Considera importante o ciclo de vida do sprint? *

Marcar apenas uma oval.

	1	2	3	4	5	
Fraco	◯	◯	◯	◯	◯	Forte

9. Desenvolvimento - Considera importante a documentação (entregas e reuniões) dos sprints? *

Marcar apenas uma oval.

	1	2	3	4	5	
Fraco	◯	◯	◯	◯	◯	Forte

10. Desenvolvimento - A ferramenta de acompanhamento do projeto escolhido (Trello) ajudou a organizar os sprints? *

Marcar apenas uma oval.

	1	2	3	4	5	
Fraco	◯	◯	◯	◯	◯	Forte

11. Validação do software - Considera importante como é dividido os tipos de testes? *

Marcar apenas uma oval.

	1	2	3	4	5	
Fraco	◯	◯	◯	◯	◯	Forte

12. Entrega e Implantação - Participação de toda equipe na apresentação dos incrementos dos produtos finalizados e prontos foi uma prática importante? *

Marcar apenas uma oval.

	1	2	3	4	5	
Fraco	◯	◯	◯	◯	◯	Forte

13. Entrega e Implantação- Considera importante a documentação de entrega e implementação do sprint? *

Marcar apenas uma oval.

	1	2	3	4	5	
Fraco	◯	◯	◯	◯	◯	Forte

ANEXO 4 - COMPARAÇÃO VIPKS vs SCRUM

Questões - Análise e especificação de requisitos	**VIPKS**	**Scrum**
Considera importante a definição do âmbito?	4,66	4,04
Considera importante a estimativa dos recursos?	4,5	3,73
Considera importante a definição do time do projeto?	4,33	4,08
Considera importante a definição da arquitetura e das boas práticas?	4,33	3,91
Questões - Desenvolvimento	**VIPKS**	**Scrum**
Considera importante a reunião diária?	4,83	4,30
Considera importante a visão do *sprint*?	4,5	4
Considera importante o ciclo de vida do *sprint*?	4,16	4,26
Considera importante a documentação (entregas e reuniões) dos *sprints*?	3,83	2,5
A ferramenta de acompanhamento do projeto escolhido (Trello) ajudou a organizar os *sprints*?	4,16	4,04
Questões - Validação do *software*	**VIPKS**	**Scrum**
Considera importante como foi dividido os tipos de testes?	4,16	4,21
Questões - Entrega e Implantação	**VIPKS**	**Scrum**
Participação de toda equipe na apresentação dos incrementos dos produtos finalizados e prontos foi uma prática importante?	4,5	3,91
Considera importante a documentação de entrega e implantação do *sprint*?	4	4,21