METODOLOGIAS ÁGEIS[1,2] auxiliando o processo de desenvolvimento de *software* de pequenas e médias empresas

Álvaro Augusto Roberto[3]
Anderson Luiz Barbosa[4]

RESUMO

Este estudo aborda assuntos concernentes aos processos de desenvolvimento de sistemas de organizações desenvolvedoras de *software* de pequeno a médio porte, apresentando as mais recentes metodologias de desenvolvimento de sistemas: as metodologias ágeis. Neste trabalho, objetivou-se demonstrar como a abordagem ágil de desenvolvimento de *software* poderia sanar alguns dos problemas mais encontrados durante o processo de desenvolvimento de *software* em empresas de pequeno a médio porte, assim como apresentar os benefícios desta abordagem para essas empresas. Para tanto, realizou-se uma coleta de dados por meio da divulgação de um formulário constituído por treze questões concernentes ao processo de desenvolvimento de *software* das organizações convidadas a participarem da pesquisa, na qual foi possível analisar os seus ambientes de desenvolvimento de *software*. A análise tinha como intuito apontar os principais problemas enfrentados pelos desenvolvedores de tais organizações e, também, analisar o nível de conhecimento dos desenvolvedores quanto a abordagem de desenvolvimento ágil. Não obstante, a partir dos resultados de um estudo de caso, no qual acompanhou-se a implantação adaptativa da metodologia *Kanban* voltada ao desenvolvimento de *software* de uma pequena empresa, coletou-se alguns dados que apresentam os benefícios adquiridos e problemas sanados após a devida implantação desta nova abordagem além de, também, serem apresentados os problemas que ainda continuam existindo. Ao final do trabalho, conclui-se que a utilização de uma abordagem de desenvolvimento ágil em harmonia com outras ferramentas como, por exemplo, guia de melhores práticas, modelos de melhoria de processos de desenvolvimento de *software*, dentre outras, possibilitam o atendimento das necessidades do mercado vigente assim como a melhoria do processo de desenvolvimento de *software* das organizações.

Palavras-chave: Engenharia de *software* ; Agilidade ; Métodos ágeis

ABSTRACT

This study approaches subjects concerning the systems development processes of small to medium-sized software development organizations and presenting the latest systems development methodologies: the agile methodologies. This monograph aimed to show how the agile software development approach could solve some of the problems most frequently encountered during the software development process on small to medium-sized companies as well as present the benefits of its approach to given companies. To do so, was carried out a data collection through the dissemination of a form composed of thirteen questions pertaining to the software development processes of the companies invited to participate in the study, in which we could analyze their software development environment. The analysis had intention to point out the main problems faced by the developers from such organizations and examine the acknowledge levels of its developers about the agile development approach. Nevertheless, from the results of a case study, which followed up an adaptive implantation to a small company of the Kanban methodology focused on the software development, was collected some data that represented the benefits and solutions for the problems acquired after the proper implantation of this approach and, in addition, the problems that still there are presented too. At the end of this work, it is concluded that the use of an agile development approach in harmony with other tools, eg, a best practice guide, models for processes improvement, among others, enables the meeting of the needs of the current market demands as well as the improvement of the software development processes from the organizations.

Keywords: *Software Engineering ; Agility ; Agile methods*

INTRODUÇÃO

[1] Artigo baseado em Trabalho e Conclusão de Curso (TCC) desenvolvido em cumprimento à exigência curricular do Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas no 2º semestre de 2014

[2] Depositado na Biblioteca da Fatec Americana em 23/12/2014.

[3] Tecnólogo em Tecnologia em Análise e Desenvolvimento de Sistemas – Fatec Americana – Centro Estadual de Educação Tecnológica Paula Souza ; Contato: alvaro.prog@gmail.com

[4] Prof. da.Fatec Americana - Doutorando em Engenharia Elétrica pela Universidade Estadual de Campinas ; Contato: anderson.barbosa@sj.unisal.br

A indústria de desenvolvimento de *software* transmutou-se em uma das mais importantes indústrias da atualidade. Contudo, o *software* foi consagrado uma das propriedades do âmbito intelectual de maior valor no mundo e, de fato, produto indispensável para o estabelecimento do estilo de vida contemporânea.

Isso se deve à constante evolução do *hardware* vivenciada desde a década de 1970, quando esse passava por uma de suas principais transformações. Tais transformações possibilitaram agregar ao *hardware* maior eficiência, declínio sobre seu custo de aquisição e diminuição em seu tamanho. Com estes fatos, esta poderosa tecnologia pôde ser disseminada mundialmente a ponto de ser empregada como ferramenta de trabalho, meio de comunicação e, mais recentemente, como forma de entretenimento.

Concomitantemente à evolução do *hardware* encontrava-se o aumento drástico da demanda por *software,* caracterizados por sua alta complexidade de desenvolvimento devido a sua necessidade de sofisticação quando a comunidade desenvolvedora de soluções em *software* vislumbrou uma série de problemas decorrentes da atividade de desenvolvimento de sistemas, período conhecido como a crise do *software*.

Com o intuito de auxiliar as empresas e suas equipes de desenvolvimento, a engenharia de s*oftware* identificou a necessidade de criação de ferramentas que amparassem o processo de desenvolvimento de sistemas. Para tanto, criou ferramentas como modelos de processos de desenvolvimento de *software*, guia de melhores práticas, metodologias de desenvolvimento ágil, dentre outras ferramentas auxiliadoras.

Amparadas por tais ferramentas de auxílio, as empresas e seus desenvolvedores, puderam ser beneficiados de forma a amenizarem os problemas relatados durante a ascensão da crise do *software* e, também, obtiveram o reconhecimento por parte de seus clientes e de possíveis futuros concorrentes.

Em divergência, mesmo após 40 anos, constata-se os mesmos problemas pautados durante a crise do *software* em cenários de algumas empresas desenvolvedoras de sistemas. Isso se atrela, na maioria dos casos, a empresas de pequeno a médio porte que, por possuírem recursos escassos ou por não adequarem as ferramentas auxiliadoras à realidade da organização, optam por empregar um ciclo de desenvolvimento próprio, imaturo, abstinente de metodologia e padrões e, também, propício a futuros problemas.

Porém, como estes problemas poderiam ser amenizados? Quais estratégias poderiam ser utilizadas? Como adequar o cenário interno de uma empresa às novas metodologias de desenvolvimento atendendo às demandas mutáveis do mercado vigente?

Portanto, o estudo se justificou hipoteticamente pela necessidade de adequação dos desenvolvedores e organizações de pequeno a médio porte do ramo de desenvolvimento de *software* às novas ferramentas de auxílio ao processo de desenvolvimento de sistemas ofertadas pela engenharia de s*oftware* como, por exemplo, modelos de processos de desenvolvimento de *software* e metodologias de desenvolvimento ágil, intuitivamente a atender às demandas do mercado, garantir melhores resultados finais e, principalmente, aperfeiçoar os processos de desenvolvimento de *software* das organizações possibilitando, prioritariamente, o atendimento aos requisitos do sistema e, também, a redução sobre seu custo e o cumprimento do prazo de entrega estipulado.

Objetivou-se, especificamente, estudar as metodologias ágeis *Kanban* e *Scrum* e, a partir de uma coleta de dados, observar seus índices de aceitação e satisfação em organizações de pequeno a médio porte e, também, o nível de conhecimento dos funcionários perante essas e outras metodologias. Além disso, através de um estudo de caso, vislumbrou-se a implantação da metodologia *Kanban* ao Setor de Processamento de Dados do Instituto de Economia (IE) da Universidade de Campinas (Unicamp) identificando os benefícios, vantagens e desvantagens decorrentes desta abordagem.

Metodologicamente, através da pesquisa bibliográfica utilizando-se livros, artigos científicos, artigos eletrônicos e e-book, foi constituído todo o contexto teórico para este trabalho no qual, durante o primeiro capítulo, aborda-se o contexto histórico da atividade de desenvolvimento de *software*. Nesta seção, são trazidos à tona os fatos vislumbrados desde meados da década de 1960, época em que inexistiam métodos de desenvolvimento de *software* bem definidos, período dos computadores de elevado consumo energético e de restrita capacidade computacional.

O segundo capítulo possui uma abordagem mais contemporânea, na qual se vislumbra a criação da engenharia de s*oftware* e sua importância para o âmbito do desenvolvimento de sistemas, as atividades comuns durante o processo de desenvolvimento de qualquer solução em *software* e o surgimento de modelos de processos de desenvolvimento de *software* e suas abordagens genéricas.

Durante o terceiro capítulo serão apresentados dois modelos de melhoria dos processos de desenvolvimento de *software*: o *Capability Maturity Model Integration (*CMMI) e o Melhoria de Processo do Software Brasileiro (MPS.BR). Esta seção abordará sucintamente as propostas dos modelos de melhoria dos processos de desenvolvimento de *software* e suas principais características.

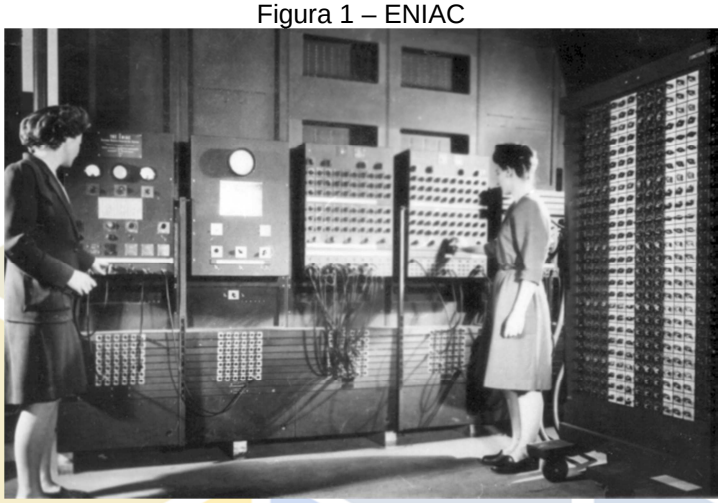
Destinado ao encerramento do levantamento bibliográfico encontra-se o quarto capítulo, constituído por uma breve introdução sobre os princípios e origens das metodologias ágeis e, logo em seguida, são apresentadas as metodologias de desenvolvimento ágeis primordiais a esta monografia, *Kanban* e *Scrum*, abordando os principais rituais e atividades realizadas por dadas metodologias.

A *posteriori*, vislumbrar-se-ão os resultados de uma coleta de dados realizada a partir de um formulário respondido por profissionais desenvolvedores de *software,* no qual se levantou informações concernentes ao processo de desenvolvimento de *software* das organizações dos atuantes e, para finalizar, será apresentado os resultados da implantação da metodologia *Kanban* para o desenvolvimento de *software* no Setor de Processamento de Dados do IE da Unicamp.

1 O DESENVOLVIMENTO DE *SOFTWARE* NA HISTÓRIA

Durante a década de 60 nascia a informática e, conjuntamente, trazia sua grandiosa e inovadora peça de arte: o computador. Como se observa na figura 1, o computador era robusto, consumia muita energia e disponibilizava pouca funcionalidade quando comparado aos computadores contemporâneos que, além de receberem dispositivos de hardware menores e cada vez mais eficientes, foram empregados, continuamente, como ferramenta multiuso para os mais distintos propósitos.

Figura 1 – ENIAC



Fonte: *HISTORIC Computer Images*, 2000.

Inicialmente, o acesso a este tipo de tecnologia se restringia às instituições de ensino, meios militares e grandes empresas multinacionais que, interessadas pelo âmbito da pesquisa, inovação e atendimento de suas necessidades, investiram vastamente em projetos de aperfeiçoamento e melhoria dessa intrigante tecnologia. Para tanto, tornou-se necessário o recrutamento de equipes desenvolvedoras de *software* que, com ou sem experiência, desenvolviam rotinas as quais, após implementadas ao computador, deferiam as necessidades de dada instituição. (KOSCIANSKI ; SOARES, 2007)

Concomitantemente a esses fatos, a partir dos anos 70, o computador passou por relevantes transformações de *hardware*, as quais permitiram a compressão de seu tamanho e a agregação de eficiência a esse equipamento. Isso possibilitou a criação do computador pessoal e alavancou a depreciação desta cobiçada tecnologia. Atualmente, graças a essas e inúmeras outras transformações, torna-se normal a utilização de aparelhos variantes dessa tão poderosa ferramenta.

Por outro lado, com tais benefícios provenientes da evolução do *hardware*, a demanda por *softwares* aumentou drasticamente trazendo certa preocupação aos primeiros programadores, como se observa no relato de Edsger Dijkstra, durante sua apresentação na *Association for the Computer Machinery (*ACM).

> A maior causa da crise do *software* é que as máquinas tornaram-se várias ordens de magnitude mais potentes! Em termos diretos, enquanto não havia máquinas, programar não era um problema; quando tivemos computadores fracos, isso se tornou um problema pequeno e agora que temos computadores gigantescos, programar tornou-se um problema gigantesco. (DIJKSTRA, 1972, p. 5)[5]

Conforme a menção feita por Dijkstra, o renomado termo crise do *software*, surgido durante a primeira geração de programadores, faz menção às complicações afrontadas ao desenvolver soluções em *software* durante o início da revolução digital. Dentre os problemas mais comuns, podem-se destacar a complexidade da correta elucidação de requisitos que, por muitas vezes, acarretava no mau funcionamento

[5] *The major cause is... that the machines have become several orders of magnitude more powerful! To put it quite bluntly: as long as there were no machines, programming was no problem at all; when we had few weak computers, programming has become a mild problem, and now we have gigantic computes, programming has become an equally gigantic problem.* Tradução Própria.

do sistema ou, até mesmo, na abstinência do propósito inicial deste; a dificuldade de desenvolver sistemas *user friendly* devido à falta de recursos das máquinas; o rompimento do orçamento inicial de um determinado projeto devido a problemas durante seu desenvolvimento; e à instabilidade de um *software*, ou seja, sua auto capacidade de parar de funcionar simplesmente sem algum propósito ou indício de mau funcionamento. (KOSCIANSKI; SOARES, 2007)

Tais complicações eram oriundas de um cenário repleto de abordagens empiristas por parte da primeira geração de programadores, uma vez que esses eram abstinentes de ferramentas, métodos, recursos e cursos de apoio ao desenvolvimento de soluções em *software*.

Desde então, o interesse sobre metodologias e ferramentas de auxílio ao desenvolvimento de sistemas tornou-se descomunal. Isto alavancou a necessidade de investimento científico nesta área que, por sua vez, proliferou a ideia da engenharia de s*oftware* que, segundo Koscianski e Soares (2007, p. 21) presume-se que "[...] a primeira vez que se utilizou o termo 'engenharia de s*oftware*' foi em uma conferência com esse nome, realizada em 1968, na Alemanha".

Paralelamente à criação e evolução da engenharia de s*oftware*, cursos e disciplinas foram acrescentados às instituições de nível médio e superior visando a disseminação do aprendizado tecnológico. Além disso, a partir de estudos científicos abordados pela engenharia de software, originaram-se ferramentas de auxílio ao desenvolvimento de *software* podendo-se citar ferramentas como a *Unified Modeling Language* (UML), modelos de processos de desenvolvimento de *software*, modelos de controle de versão de sistemas, ferramentas de comunicação, modelos de melhoria de processos de desenvolvimento de *software*, metodologias de desenvolvimento ágil, dentre outras ferramentas que podem servir de auxílio durante o desenvolvimento de um projeto de sistema.

Porém, ainda hoje, como afirmam Koscianski e Soares (2007, p. 22):

> [...] mais de trinta anos depois, quais são os problemas enfrentados na construção e utilização de software? Ao lermos o relatório da conferência da NATO de 1968 e outros documentos produzidos na década de 1970, fazemos uma descoberta assustadora: os problemas são os mesmos que encontramos atualmente.

2 O PROCESSO DE DESENVOLVIMENTO DE SOFTWARE

Advindos de um cenário caótico no qual a produção de *software* era realizada de maneira desorganizada, desestruturada e abstinente de planejamento, a necessidade de criação de processos planejados, estruturados e padronizados para a orientação aos processos de desenvolvimento de *software* ficou indispensável.

Embora não dispunham de ferramentas auxiliares ao processo de desenvolvimento, os primeiros desenvolvedores embasavam-se em conceitos típicos do âmbito da engenharia convencional utilizando tais conceitos para a padronização do desenvolvimento de *software* em dada época. Esta abordagem colaborou para a criação da engenharia de s*oftware* que, de acordo com Bauer (1969 APUD PRESSMAN, 2011, p. 39), nada mais é que "[...] o estabelecimento e o emprego de sólidos princípios de engenharia de modo a obter um *software* de maneira econômica, que seja confiável e funcione de forma eficiente em máquinas reais".

Com o advento da engenharia de s*oftware*, novas técnicas e padrões passaram a serem desenvolvidas com o intuito de auxiliar a comunidade desenvolvedora de *software*. Contudo, uma das primeiras técnicas elaboradas foi os Processos de Desenvolvimento de *Software* (PDS) que, através da criação de documentos e artefatos capazes de representarem a contextualização do *software* perante as suas necessidades e restrições, vislumbra a obtenção de um produto final de qualidade.

Embora existam vastas abordagens de como desenvolver um determinado *software*, quatro atividades genéricas são imprescindíveis a qualquer projeto de desenvolvimento de *software*. Porém, as técnicas utilizadas variam de acordo com o tipo de *software* a ser desenvolvido, com as pessoas relacionadas ao projeto e com as estruturas organizacionais envolvidas.

De maneira suscita, segundo Sommerville (2011, p. 36), as quatro atividades básicas para o desenvolvimento de um software são a especificação de *software*, o projeto e implementação, a validação de *software* e, por último mais não menos importante, a evolução do *software*.

2.1 Especificação de *software*

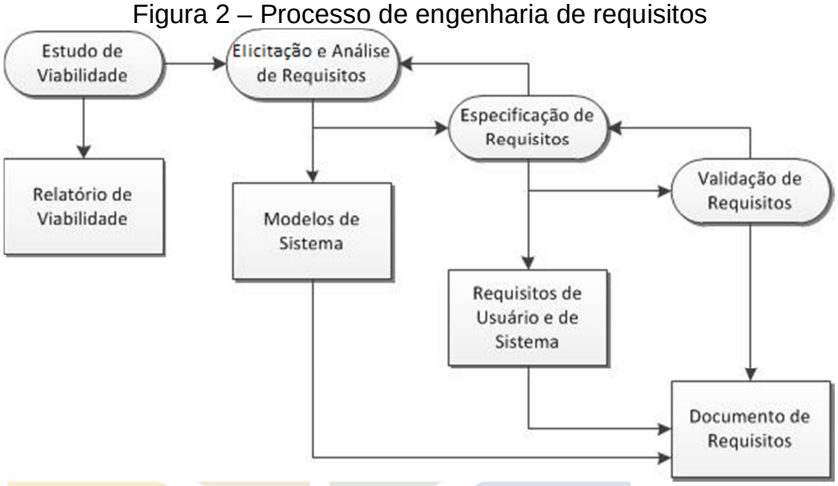
Esta fase do desenvolvimento de *software*, também conhecida como engenharia de requisitos, é uma atividade particularmente crítica e que merece extrema atenção durante sua execução, pois, de certa forma, os erros cometidos durante essa etapa acarretarão em futuros problemas no decorrer do projeto. (SOMMERVILLE, 2011)

Entretanto, é de suma importância elucidar, analisar e validar os requisitos amenizando, por sua vez, a possibilidade de surgimento de problemas oriundos da má especificação de *software* realizada durante os primórdios do projeto.

Para isso, o processo de especificação de *software* ou engenharia de requisitos deve produzir um documento constituído pelas especificações e restrições do sistema em questão. Contudo, de forma a facilitar a compreensão de leigos e, ao mesmo tempo, de especialistas, o diagrama de caso de uso e sua devida documentação são documentos pertinentes à *Unified Modeling Language* (UML) que auxiliam a elucidação, a análise e a validação dos requisitos do sistema.

Geralmente, durante a especificação de *software* são realizados processos de estudo de viabilidade, análise, especificação e validação de requisitos como o ilustrado pela figura 2.

Figura 2 – Processo de engenharia de requisitos



Fonte: Adaptado de SOMMERVILLE, 2011, p. 38

Tais processos visam a criação de um documento de requisitos – por exemplo, a criação do diagrama de casos de uso e de sua respectiva documentação – que servirá, durante todo o projeto, como base para as demais atividades durante o processo de desenvolvimento do *software*.

2.2 Projeto e implementação de *software*

Também conhecida como planejamento e desenvolvimento, a atividade de projeto e implementação de *software* destina-se ao processo de transformação das especificações relatadas no documento de requisitos em um sistema executável, ou seja, é pertinente a essa atividade a responsabilidade de transmutar os requisitos pautados no documento de requisitos em protótipos, linhas de código, módulos do sistema ou, até mesmo, no sistema propriamente dito. (SOMMERVILLE, 2011)

Entretanto, ante a iniciar o processo de desenvolvimento do *software* é imprescindível projetar como a aplicação deverá ser construída estipulando informações como, por exemplo, a maneira como os diversos módulos e funcionalidades do sistema interagirão entre si, quais tecnologias melhor atenderão às especificações do sistema em discussão, quais processos deverão ser executados para que certa funcionalidade seja deferida, dentre outras definições. (BRAUDE, 2005)

Um projeto é constituído por um conjunto de documentos que, geralmente, resume-se em diagramas pertinentes à UML, os quais descrevem o comportamento da aplicação e sua estrutura operacional, ou seja, da mesma maneira que engenheiros civis dispõem da planta de uma casa para a construção da mesma, as equipes de desenvolvimento de *software* dispõem da “planta” do *software* para a construção de uma aplicação. (BRAUDE, 2005)

Devido à grande importância que os documentos pautados durante o projeto de *software* impõem sobre o projeto de uma maneira geral, torna-se viável que as demais atividades tenham início após a minuciosa definição e validação destes documentos.

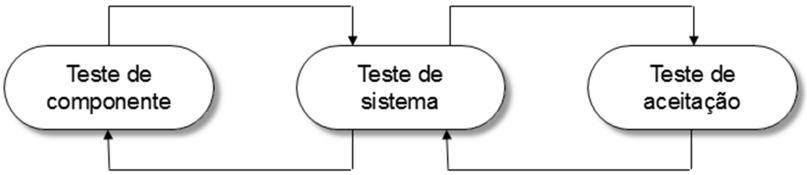
Pertinentes à atividade de projeto e implementação de *software* também se encontram as atividades de teste e depuração operacionalizadas pela equipe desenvolvedora de *software*. Essas atividades possuem a característica de abstrair quaisquer controvérsias aos requisitos pautados nos diagramas do projeto eliminando-se, quando encontrados, defeitos, erros ou falhas do sistema.

2.3 Validação de *software*

Conhecida popularmente por testes de aceitação, a validação de *software* é uma tarefa antecedente à implementação do sistema propriamente dito, na qual objetiva-se "[...] mostrar que um sistema está em conformidade com sua especificação e que atende às expectativas do cliente". (SOMMERVILLE, 2011, p. 41)

Para tanto, torna-se necessário a adoção de um processo de teste de *software* como, por exemplo, o processo de três estágios proposto por Sommerville, ilustrado na figura 3:

Figura 3 – Processo de teste



Fonte: Adaptado de SOMMERVILLE, 2011, p. 41

De acordo com o modelo apresentado, é estipulado um processo de teste de *software* interativo no qual as informações geradas por testes posteriores realimentam as etapas iniciais do processo. Cada etapa do processo envolve um teste específico. Portanto, o teste de componente ou unidade é realizado em componentes ou módulos do sistema individualmente, garantindo a correta operação de tal unidade; o teste de sistema resume-se no teste do software como uma única unidade, ou seja, é o teste realizado a partir da integração de todos os componentes do sistema; e, anteriormente à implementação para uso operacional do *software* desenvolvido, são realizados testes de aceitação que se resumem em testes com dados fornecidos pelo cliente.

Durante os testes podem ser necessários reparos no sistema para melhor atender às especificações do sistema e, concomitantemente às alterações, novos testes deverão ser realizados de maneira a averiguar a correta operação do sistema. Após a aprovação pelo cliente, o sistema poderá ser implementado.

2.4 Evolução do *software*

Posteriormente ou não à entrega do *software* poderão vir à tona requisições de alterações no sistema, sendo os principais tipos dessas, requisições de inclusão de funcionalidades ou adequação às novas tecnologias e restrições advindas do mercado vigente.

Para atender essa demanda, torna-se necessário executar todo o processo de desenvolvimento de *software* como descrito anteriormente. Portanto, conclui-se que estas quatro atividades genéricas formam o ciclo de desenvolvimento do *software*.

2.5 Modelos de processos de desenvolvimento de *software*

Modelos de processos de desenvolvimento de *software* foram desenvolvidos intuitivamente para demonstrarem uma abordagem generalizada do desenvolvimento de um sistema sob determinada perspectiva, de maneira a transformar o desenvolvimento de *software* em uma atividade menos caótica. (PRESSMAN, 2011)

Portanto, é importante salientar que tais abstrações jamais devem ser confundidas como descrições definitivas de processos de desenvolvimento de *software* pois, conforme leciona Sommerville (2011, p. 29), tais abstrações "[...] podem ser consideradas como *frameworks* de processo que podem ser ampliadas e adaptadas para criar processos mais específicos de engenharia de *software*".

Contudo, os processos de desenvolvimento de *software* são vislumbrados em todas as organizações desenvolvedoras de *software* e distinguem-se de acordo com o grau de formalidade, o porte organizacional, os tipos de *softwares* desenvolvidos, dentre outros fatores. Porém, é evidente que não há um modelo de processo de desenvolvimento padrão para todas as organizações e o que existe, sim, é a necessidade de desenvolvimento de um modelo de processos de desenvolvimento de *software* por parte de cada organização, sempre levando em consideração as necessidades e realidades vivenciadas por cada uma delas. (SOMMERVILLE, 2011)

Para a produção de tal modelo, torna-se viável o embasamento em modelos já predefinidos pela engenharia de *software* que, como supramencionado, traz um nível de abstração genérico permitindo-se sua adaptação para o melhor atendimento às necessidades da organização.

Nos capítulos a seguir, serão apresentados dois modelos de processos de desenvolvimento de *software* mais comuns nas organizações desenvolvedoras de sistemas: o modelo cascata e o modelo espiral.

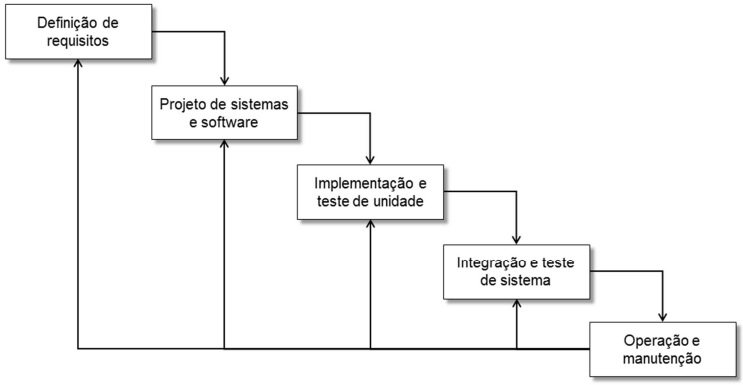
Modelo cascata

Este foi o primeiro modelo de processo de desenvolvimento de *software* publicado, o qual também é conhecido como *waterfall*, *top-down* ou simplesmente por ciclo de vida do *software*. É caracterizado pelo seu encadeamento entre as fases do processo de desenvolvimento e recomendado apenas para casos que os requisitos e restrições do sistema sejam previamente bem definidos e, preferencialmente, estáveis. (MAGELA, 2006; PRESSMAN, 2011; SOMMERVILLE, 2011)

De acordo com este modelo, o congelamento dos requisitos de um sistema prematuramente pode acarretar no não atendimento das necessidades reais do cliente. Da mesma magnitude, alterar os requisitos do sistema proporcionarão atrasos na entrega do *software* e acréscimos ao montante pago pelo desenvolvimento do mesmo. Por outro lado, este modelo também possui benefícios que se resumem nas documentações produzidas por cada fase do processo de desenvolvimento e sua aderência a outros modelos de processo de engenharia. (KOSCIANSKI; SOARES, 2007; SOMMERVILLE, 2011)

Na figura 4, observa-se as fases do modelo em cascata que são diretamente relacionadas às atividades genéricas do processo de desenvolvimento de *software*. Vale ressaltar que, de acordo com o modelo, para dar início às fases posteriores do processo de desenvolvimento, a fase anterior a esta deve ser totalmente concluída, ou seja, para dar início à fase de projeto de sistemas e *software*, a etapa de definição de requisitos deverá ser concluída em sua totalidade.

Figura 4 – Ciclo de vida do *software*



Fonte: Adaptado de SOMMERVILLE, 2011, p. 30.

Durante a fase de definição de requisitos, os objetivos e restrições do sistema são definidos e detalhados, conforme a atividade de especificação de *software* abordada no capítulo 3.1; o projeto de sistemas e *software* envolve a atividade de arquitetura e organização do sistema; a implementação e teste de unidade destina-se à atividade de codificação e teste unitário que, nada mais é que a averiguação de seu correto funcionamento conforme sua especificação; a atividade de integração e teste de sistema é a operação de junção de todas as unidades individuais do sistema intuitivamente a testar a aplicação como um sistema completo, averiguando-se seu correto desenvolvimento e operacionalidade; e, por último, encontra-se a atividade de operação e manutenção, a qual implementa-se o sistema, ou seja, é realizada a entrega ao cliente e manutenções para a correção de erros, falhas ou defeitos identificados durante esta etapa são efetuadas caso sejam necessárias. (SOMMERVILLE, 2011)

Portanto, devido à realidade dinâmica vivenciada atualmente, o modelo cascata não traz vantagens significativas para o desenvolvimento da maioria dos sistemas atuais, uma vez que o *software* está subordinado a constantes adaptações e evoluções, o que inviabiliza a utilização deste modelo.

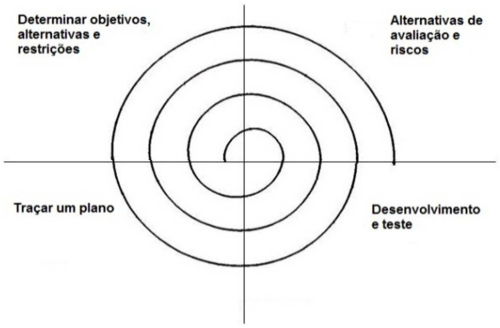
Modelo espiral

Posterior à publicação do modelo cascata, surgiram outros modelos de desenvolvimento de *software* intuitivamente propondo aperfeiçoar processo de desenvolvimento e sanar as deficiências encontradas no modelo supracitado.

Para tanto, a abordagem evolucionária de desenvolvimento de *software*, caracterizada por sua entrega de valor a cada *release*, busca atender às mudanças de requisitos vivenciadas durante o ciclo de vida do *software.* (SOMMERVILLE, 2011)

Um dos modelos surgidos para tal função foi o modelo evolucionário em espiral que, proposto por Boehm em 1980, foi baseado na junção dos aspectos controladores e sistemáticos do modelo cascata com a interatividade do modelo de prototipação. (PRESSMAN, 2011; SOMMERVILLE, 2011)

Figura 5 – Modelo espiral



Fonte: Adaptado de MAGELA, 2006, p. 30.

Conforme sua nomenclatura, as atividades desse modelo são representadas em forma de uma espiral, a qual é dividida em quatro seções: determinar objetivos, alternativas e restrições; alternativas de avaliação e riscos; desenvolvimento e teste; e, traçar um plano. (figura 5)

Um ciclo da espiral, segundo Sommerville (2011, p. 50) funciona da seguinte maneira: começa com a elaboração de objetivos, como desempenho e funcionalidade. Os caminhos alternativos para alcançar esses objetivos e as restrições impostas sobre cada um deles são, então, enumerados. Cada alternativa é avaliada em relação a cada objetivo e as fontes de riscos de projeto são identificadas. O próximo passo é resolver esses riscos por meio de atividades de coleta de informações, tais como análise mais detalhada, prototipação e simulação. Após a avaliação dos riscos, é realizada uma parte do desenvolvimento, seguida pela atividade de planejamento para a próxima fase do processo.

Portanto, a principal diferença entre esse modelo e os demais é sua análise de risco a cada interação da espiral. Tal atividade tenta minimizar os riscos encontrados para que seja possível cumprir o cronograma e o custo preestabelecidos. (MAGELA, 2006; SOMMERVILLE, 2011)

Embora possua um real controle de risco e orçamento de um projeto, esse modelo não suporta de maneira simplista as constantes mudanças do *software* tornando-se um modelo inadequado para o atual cenário de desenvolvimento de *software*. (MAGELA, 2006)

3 MELHORIA DO PROCESSO DE DESENVOLVIMENTO DE SOFTWARE

De acordo com Sommerville (2011), a demanda por *softwares* mais baratos, com maior valor agregado e com prazos de entrega cada vez mais curtos são características vivenciadas pela grande maioria das indústrias desenvolvedoras de soluções em *software*.

Consequentemente, muitas companhias têm aderido a projetos de melhoria do processo de desenvolvimento de *software* com o intuito de amenizar o custo sob o produto final, agregar o maior valor possível ao produto final e realizar as entregas devidamente em seus prazos estipulados.

Porém, os modelos de melhoria do processo de desenvolvimento de *software* possuem duas abordagens distintas, sendo uma voltada à maturidade do processo, a qual o foco se estabelece na melhoria dos processos adotados pela organização e no gerenciamento do projeto introduzindo boas práticas de engenharia de s*oftware* e, a outra abordagem destina-se ao desenvolvimento ágil de *software,* focado no emprego do desenvolvimento interativo postergando-se as atividades consideradas menos importantes durante o processo de desenvolvimento de *software*. (SOMMERVILLE, 2011)

Sendo assim, a abordagem voltada à maturidade do processo caracteriza-se por adotar atividades que não são diretamente relevantes à programação. Já a abordagem ágil foca especificamente no código sendo desenvolvido, minimizando as formalidades e documentações encontradas na primeira abordagem.

Nas seções a seguir serão apresentados, sucintamente, as estruturas e características de dois modelos de melhoria dos processos de desenvolvimento de *software:* um modelo internacionalmente conhecido, o *Capability Maturity Model Integration (*CMMI) e, um modelo nacionalmente conhecido, o Melhoria de Processo do *Software* Brasileiro (MPS.BR).

3.1 Capability maturity model integration

Patrocinado pelo Departamento de Defesa (DOD) dos Estados Unidos da América (EUA) e criado pelo *Software Engineering Institute (SEI*) no final da década de 1980, o modelo para melhoria de processos, apenas para *softwares*, o *Software Capability Maturity Model (*SW-CMM), gerou novos padrões para a engenharia de sistemas e intuitivamente a passou a averiguar a capacidade e qualidade dos fornecedores de *software* ao DOD dos EUA. (KOSCIANSKI; SOARES, 2007)

Devido tal abordagem por parte de um importante instituto governamental americano, este modelo foi amplamente aceito pela comunidade desenvolvedora de *software*. Porém, pela sua especificidade à área

de *software,* outras áreas importantes da empresa não participam do escopo do SW-CMM. Entretanto, com o objetivo de integrar outros modelos de maturidade o SEI, em 2002, segundo Fernandes e Abreu (2012, p. 314), foi criado o CMMI:

> [...] um modelo evolutivo em relação aos vários CMMs, com o objetivo de combinar as suas várias disciplinas em uma estrutura única, flexível e componentizada, que pudesse ser utilizada de forma integrada por organizações que demandavam processos de melhoria em âmbito corporativo.

O objetivo primordial do modelo CMMI é "servir de guia para a melhoria de processos na organização e também da habilidade dos profissionais em gerenciar o desenvolvimento, aquisição e manutenção de produtos ou serviços". (KOSCIANSKI; SOARES, 2007, p. 102)

Para atingir este objetivo, o modelo é composto por duas abordagens distintas para sua implementação: a abordagem por estágios e a abordagem contínua. Entretanto, dentre elas a mais utilizada pelas organizações é a abordagem por estágios, a qual será brevemente apresentada durante a próxima seção.

Abordagens por estágios

Segundo Koscianski e Soares (2007, p. 105), "a representação por estágios organiza as áreas de processo em cinco níveis de maturidade (como no SW-CMM) para suportar e guiar a melhoria de processos".

Tal abordagem sugere que a melhoria dos processos da empresa seja executada de acordo com a ordem dos cinco níveis de maturidade: inicial (1), gerenciado (2), definido (3), gerenciado quantitativamente (4) e otimizado (5). Porém, cada nível de maturidade, assim como ilustrado na figura 6, é composto por diversas áreas de processos que, por sua vez, são constituídos por objetivos e práticas genéricas e obrigatórias.

Figura 6 – Áreas de Processo – CMMI por estágios

Nível de Maturidade 5 - Otimizado	Inovação e implantação na organização
	Análise e resolução de causas
Nível de Maturidade 4 - Gerenciado Quantitativamente	Desempenho do processo organizacional
	Gerência quantitativa do projeto
Nível de Maturidade 3 - Definido	Desenvolvimento de requisitos
	Solução técnica
	Integração do produto
	Verificação
	Validação
	Foco no processo organizacional
	Treinamento organizacional
	Gerência de projeto integrada
	Gerência de riscos
	Análise de decisão e resolução
	Desempenho do processo organizacional
	Definição do processo organizacional
Nível de Maturidade 2 - Gerenciado	Gerência de requisitos
	Planejamento do projeto
	Gerência e controle do projeto
	Gerência de acordos com fornecedores
	Medição e Análise
	Garantia da qualidade do processo e do produto
	Gerência de configuração
Nível de Maturidade 1 - Inicial	

Fonte: Adaptado de KOSCIANSKI; SOARES, 2007, p. 108.

Segundo Fernandes e Abreu (2012, p. 321), “o cumprimento das metas específicas e genéricas correspondentes a estas áreas de processo é um pré-requisito para o atingimento do nível de maturidade correspondente”. Em outras palavras, para que um nível de maturidade seja atingido todas as atividades referentes às áreas de processos pertinentes ao nível em questão deverão ser cumpridas.

Não obstante, Koscianski e Soares (2007, p. 106) salientam que “[...] quando uma organização atinge as práticas necessárias para estar em um nível, isto significa que pratica todos os requisitos necessários dos níveis imediatamente anteriores”.

Para melhor entendimento, os níveis serão destrinchados de modo a apresentar as características genéricas de cada um deles, dando início pelo primeiro nível de maturidade, o qual é caracterizado pelos processos caóticos, pela falta de padrões e pelos problemas referentes ao não cumprimento dos prazos de entrega e do custo pré-estabelecido.

Porém, “o fato de uma organização estar no nível 1 e ter processos de desenvolvimento caóticos não significa necessariamente que seus produtos finais são ruins. É possível, até mesmo, que bons produtos sejam entregues”. (KOSCIANSKI; SOARES, 2007, p. 106)

Já o segundo nível de maturidade, o gerenciado, é caracterizado pela preocupação em tornar os requisitos gerenciados e, também, planejar, medir e controlar os processos a serem executados. Portanto, este nível tem grande preocupação em sempre analisar o andamento das tarefas eliminando, quando identificadas, anomalias e não conformidades através de ações corretivas. (KOSCIANSKI; SOARES, 2007, p. 107)

Diferentemente do nível gerenciado, o definido foca no processo de engenharia de produtos no qual os processos são bem caracterizados e entendidos devido à utilização de padrões, procedimentos, ferramentas e métodos bem definidos. Outra característica que distingue o nível definido do anterior é o maior detalhamento e rigorosidade na descrição dos processos. (FERNANDES; ABREU, 2012. KOSCIANSKI; SOARES, 2007)

O quarto nível de maturidade, denominado gerenciado quantitativamente, realiza o controle dos processos através da abordagem de indicadores e métodos estatísticos além de outras técnicas quantitativas. Portanto, todos os dados sobre os processos são coletados e analisados estatisticamente. Graças a esta abordagem, segundo Koscianski e Soares (2007, p. 108), “uma distinção marcante do nível 4 em relação ao anterior é o aumento da previsibilidade do desempenho de processos”.

Destinado ao quinto e último nível de maturidade do modelo encontra o nível otimizado, no qual existe a melhoria contínua dos processos a qual é obtida a partir de inovações e o melhor proveito das tecnologias.

Contudo, vale ressaltar que a atividade de melhoria de processos não é pertinente especificamente aos grupos hierárquicos mais altos da organização e, sim, de todos os seus constituintes. (FERNANDES; ABREU, 2012. KOSCIANSKI; SOARES, 2007)

3.2 Melhoria de processos do *software* brasileiro

Devido às dificuldades de implantação do modelo CMMI em empresas brasileiras de desenvolvimento de *software* como, por exemplo, pela falta de recursos financeiros e a falta de pessoal especializado, pesquisadores brasileiros deram início à criação do modelo de Melhoria de Processos do *Software* Brasileiro (MPS.BR) em 2003. Dado modelo, segundo Koscianski e Soares (2007, p. 142) focado nas “[...] micro, pequenas e médias empresas de *software* brasileiras que possuem poucos recursos para melhoria de processos, mas que estão diante da necessidade de fazê-lo”.

Seu desenvolvimento foi embasado nas normas NBR ISO/IEC 12207, ISO/IEC 15504 e no CMMI. Este modelo, diferentemente do CMMI, possui sete níveis de maturidade, os quais foram divididos de A (melhor nível) a G (pior nível). Assim como no modelo CMMI, segundo Fernandes e Abreu (2012, p. 337): para atingir um nível de maturidade, é esperado que todos os processos relativos a este nível atendam aos resultados esperados dos próprios processos, assim como os resultados esperados dos atributos dos processos correspondentes àquele nível (e também aos processos de todos os níveis anteriores).

Na figura 7, vislumbra-se os sete níveis de maturidade do MPS.BR, seus atributos e sua similaridade com o modelo CMMI.

Figura 7 – Níveis de maturidade do MPS.BR

Nível de Maturidade	Processos do Nível de Maturidade	Equivalência com o CMMI
G – Parcialmente Gerenciado	Gerência de requisitos	Nível 2
	Gerência de projetos	
F – Gerenciado	Aquisição	
	Gerência de configuração	
	Gerência de portfólio de projetos	
	Garantia da qualidade	
E – Parcialmente Definido	Treinamento	Nível 3
	Avaliação e melhoria do processo organizacional	
	Definição do processo organizacional	
	Adaptação do processo para gerência de projeto	
D – Largamente Definido	Desenvolvimento de requisitos	
	Solução técnica	
	Integração do produto	
	Instalação do produto	
	Liberação do produto	
	Validação	
	Verificação	
C – Definido	Análise de decisão e resolução	
	Gerência de riscos	
B – Gerenciado Quantitativamente	Desempenho do processo organizacional	Nível 4
	Gerência quantitativa do produto	
A – Em Otimização	Inovação e implantação na organização	Nível 5
	Análise de causas e resolução	

Fonte: Adaptado de KOSCIANSKI; SOARES, 2007, p. 155

Portanto, a divisão dos níveis de maturidade em sete partes permite sua implementação, avaliação e melhoria em um curto período de tempo e de maneira gradual sendo adequado a micro, pequenas e médias empresas devido aos seus menores custos de implementação. Não obstante, de acordo com resultados de desempenho relatados à Associação para Promoção da Excelência do Software Brasileiro (SOFTEX) em 2010, constata-se que empresas que utilizam este modelo de melhoria de processos obtiveram aumento no faturamento e na quantidade de clientes internos, além do decréscimo no prazo de entrega e do custo médio dos projetos. (FERNANDES; ABREU, 2012)

4 METODOLOGIAS DE DESENVOLVIMENTO ÁGIL

Como já visto previamente, o ambiente de desenvolvimento de *software* atual caracteriza-se por sua mutabilidade, ou seja, sua alta capacidade de mudança, adaptação e aperfeiçoamento durante o processo de desenvolvimento de soluções em *software*.

Portanto, como os modelos de desenvolvimento tradicionais são pesados, aplicam-se apenas aos processos e não se adaptam facilmente a ambientes mutáveis, tornou-se necessário a criação de metodologias leves, que levassem em consideração a entrega incremental de *software* e a adaptação às mudanças de requisitos durante o processo de desenvolvimento. (SOMMERVILLE, 2011)

Daí a origem do desenvolvimento ágil que obteve popularização em 2001, quando 17 especialistas em processos de desenvolvimento de *software* reuniram-se, estudaram e analisaram medidas que possibilitariam o aumento dos índices de sucesso dos projetos. Contudo, o grupo de especialistas chegou à conclusão de que independentemente das práticas e ênfases de cada metodologia ágil, quatro princípios são primordiais a qualquer uma delas.

Para a divulgação destes quatro valores obtidos através desta pesquisa, os especialistas publicaram o que conhecemos como Manifesto Ágil, o qual enfatiza: indivíduos e interações em vez de processos e ferramentas; *software* executável em vez de documentação; colaboração do cliente ao invés de negociação de contratos; e, respostas rápidas a mudanças em vez de seguir planos. (BECK et al., 2001. KOSCIANSKI; SOARES, 2007)

Porém, como concluem Koscianski e Soares (2007, p. 194):

> O Manifesto Ágil não rejeita processos e ferramentas, documentação, negociação de contratos nem planejamento, mas simplesmente mostra que estes têm importância secundária quando comparados com

os indivíduos, com o *software* executável, com a colaboração dos clientes e as respostas rápidas as mudanças.

No mesmo contexto, Koscianski e Soares (2007, p.194) afirmam que os conceitos privilegiados pelo Manifesto Ágil são mais adequados a forma como as pequenas e médias empresas trabalham e respondem às mudanças.

Entretanto, intuitivamente a apresentar duas abordagens de metodologias de desenvolvimento ágil em destaque nos ambientes das mais diversas empresas, serão expostas as características e cerimônias genéricas das metodologias *Kanban* e *Scrum*.

4.1 Kanbam

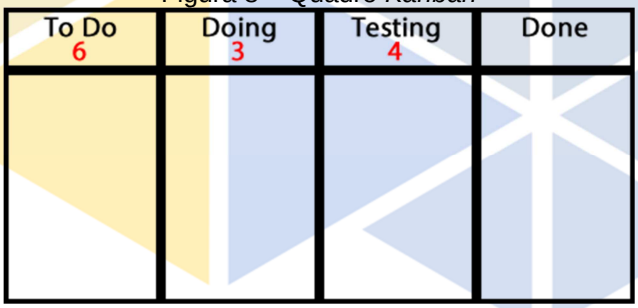
O sistema *Kanban* para o desenvolvimento de *software* tem sua abordagem consistente no fluxo e no contexto de desenvolvimento sendo caracterizada por ser uma metodologia diretamente relacionada ao Desenvolvimento *Lean* de Produtos que, por sua vez, oferece uma abordagem menos prescritiva e torna possível transformar-se em uma extensão às outras metodologias ágeis de desenvolvimento de *software* como, por exemplo, a *Scrum*. (BOEG, 2012)

Embora possua inúmeras abordagens para essa metodologia, seus princípios são os mesmos para cada uma delas. Em outras palavras, para qualquer abordagem *Kanban* deve-se considerar que essa metodologia é um método de gestão de mudanças que visa visualizar todo o trabalho em andamento, visualizar cada etapa do processo de desenvolvimento, restringir uma capacidade máxima de trabalho por etapa, medir e gerenciar o fluxo de trabalho e identificar oportunidades de melhoria de forma contínua. (BOEG, 2012)

Originada do japonês, a palavra *kanban* significa cartão visual. Entretanto, como supracitado, todo o fluxo do trabalho torna-se visível com a utilização da metodologia *Kanban*. Além disso, essa metodologia é conhecida por Sistema Puxado *Kanban*, a qual restringe a capacidade de trabalho por etapa, impossibilitando a sobrecarga de tarefas para os indivíduos, tornando-se necessário concluir o trabalho atual para que as tarefas posteriores possam ser executadas. Isso resulta em tarefas sendo puxadas pelo sistema de acordo com sua capacidade de trabalho e não, como em outras metodologias, empurradas para serem executadas devido a previsões ou demandas previamente estipuladas.

Na figura 8, demonstra-se um exemplo de um quadro *Kanban* de desenvolvimento de *software*.

Figura 8 – Quadro *Kanban*



Fonte: Imagem própria.

No quadro da figura 8, observa-se um exemplo de fluxo do desenvolvimento de *software,* no qual as etapas do desenvolvimento estão descritas no cabeçalho das colunas e as restrições de tarefas por cada etapa estão em forma numérica e em coloração avermelhada.

Devido à sua abordagem menos prescritiva esta metodologia é de fácil implementação e adaptabilidade. Contudo, basta definir as etapas de desenvolvimento do produto, segmentar as atividades e apropriar o modelo genérico apresentado anteriormente à realidade da organização juntamente a outras ferramentas que garantam a qualidade do processo de desenvolvimento como, por exemplo, modelos de melhoria de processos de desenvolvimento de *software*.

4.2 Scrum

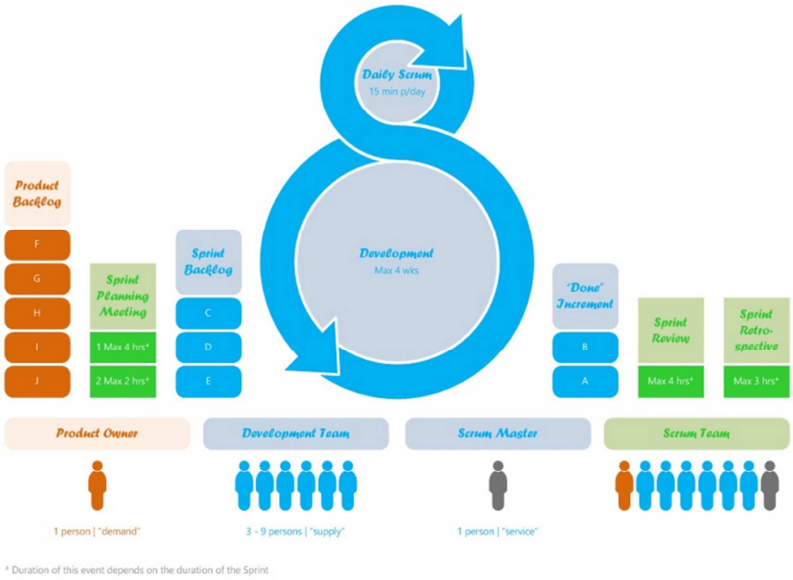
O *Scrum* é um *framework* de desenvolvimento de *software* ágil de abordagem interativa e incremental fundamentada em teorias empíricas que afirmam que o conhecimento é adquirido da tomada de decisões e da experiência baseada no que é conhecido. (SCHWABER e SUTHERLAND, 2011, SOMMERVILLE, 2011)

De acordo com Koscianski e Soares (2007, p. 200), “o foco da metodologia é encontrar uma forma de trabalho dos membros da equipe para produzir o *software* de forma flexível e em um ambiente em constante

mudança". Portanto, esse *framework* busca uma forma de trabalho flexível e adaptável a ambientes dinâmicos assegurando tratar mudanças frequentes de requisitos dentre outros problemas que poderão surgir conforme o desenvolvimento do sistema.

Para suprir estas necessidades, o *Scrum* é composto por papéis, eventos, artefatos e regras sendo cada um deles de suma importância para seu sucesso e, segundo seus criadores, Schwaber e Sutherland (2011, p. 3), "as regras do *Scrum* integram os papéis, eventos e artefatos, administrando as relações e interações entre eles".

Figura 9 – Visão geral do *Scrum*



Fonte: http://calvinx.com/wp-content/uploads/2014/05/scrum-overview-mark-hoogveld.jpg

Na figura 9, vislumbra-se uma visão geral do *framework Scrum*, na qual são apresentados todos os papéis, eventos e artefatos do *framework* que serão detalhados nos capítulos seguintes.

Papéis

O *Scrum*, mais comumente mencionado como time *Scrum*, possuí a capacidade de auto-organização. Portanto, esses grupos auto organizáveis são competentes a ponto de completar o trabalho solicitado sem a necessidade de intervenção de membros exteriores à equipe. (SCHWABER e SUTHERLAND, 2011, p. 3)

Esse time é constituído por três papéis. Estes são o *Product Owner*, o time de desenvolvimento e o *Scrum Master*. Segundo Sabbagh (2013, p. 59), "[...] as pessoas que desempenham esses papéis são igualmente responsabilizadas pelos resultados do trabalho e, assim, se comprometem com o projeto. [...] Dessa forma, eles são membros de um mesmo time, e trabalham juntos, de forma colaborativa, para alcançarem seus resultados."

A seguir, observar-se-ão as definições e características do *Product Owner*, do time de desenvolvimento e do *Scrum Master*.

- **Product owner:** também conhecido como PO ou dono do projeto, este papel representa o responsável pelo produto. Desta forma, este integrante do time *Scrum* é responsável por gerenciar o *Product Backlog* e maximizar o valor do trabalho do time de desenvolvimento. Sendo o gerenciamento do *Product Backlog* de responsabilidade exclusiva do *Product Owner*, este o deve fazer de forma a garantir que o mesmo seja visível, transparente e claro para todos alcançando, da forma mais benéfica possível, as metas e objetivos requeridos pelo cliente. (SCHWABER e SUTHERLAND, 2011) Para seu sucesso, é inevitável que toda a organização respeite as decisões tomadas por ele e que o time de desenvolvimento siga o especificado no *Product Backlog*.
- **Time de desenvolvimento: o** time de desenvolvimento constitui-se de profissionais que transformam parte do *Product Backlog* em um incremento do sistema ao final de cada *Sprint*, ou seja, são responsáveis por criarem rotinas que, após serem testadas e averiguadas conforme sua especificação são liberadas para serem entregues como parte incremental do produto. Caracterizado por ser auto organizável e multifuncional, é exclusividade deste grupo abstrair como transformar o *Product Backlog* em incrementos de funcionalidades utilizáveis. Além disso, o time de desenvolvimento não possui diferenciação de especialidades entre seus integrantes, ou seja, todos

são classificados como desenvolvedores, independentemente do trabalho que está sendo realizado pelo indivíduo. Segundo Schwaber e Sutherland (2011, p. 6), "o tamanho do time de desenvolvimento é pequeno o bastante para se manter ágil e grande o suficiente para completar uma parcela significativa do trabalho dentro dos limites da *Sprint*". Portanto, dependendo da grandiosidade e complexidade do projeto, o número de integrantes do time de desenvolvimento pode sofrer alterações.

- ***Scrum Master: e***ste papel está destinado ao servo-líder do time *Scrum*, ou seja, o responsável por garantir que o *framework* seja entendido e utilizado de maneira coerente averiguando se o time adere às teorias, práticas e regras do *Scrum*. Além de auxiliar os membros do time *Scrum* a qualquer momento, ele também possui a responsabilidade de maximizar o valor criado pelo time adaptando as interações do *framework* para melhor atender às necessidades do projeto.

Eventos

Os eventos *Scrum* resumem-se às rotinas que proporcionam o desenvolvimento do *software*, ou seja, o ciclo de desenvolvimento que, no *framework*, denomina-se *Sprint.*

Durante a *Sprint* existem cerimônias a serem realizadas: *Sprint Planning, Daily Scrum, Sprint Review e Sprint Retrospective*. Todas estas, inclusive a *Sprint*, são caracterizadas por possuírem um período máximo de execução objetivando limitar o tempo em que um objetivo deve ser alcançado e restringindo o desperdício de tempo com tarefas que não agreguem valor.

- ***Sprint:*** *s*egundo Schwaber e Sutherland (2011, p. 8), "o coração do *Scrum* é a *Sprint*". É um evento que leva de uma a quatro semanas para ser concluído sendo gerado, ao final deste, uma versão incremental do produto. Para o atendimento da criação do incremento desejado, cada *Sprint* possui uma definição que guiará todo o processo de produção, garantindo a conformidade do produto final. Entretanto, para a realização de tal evento, reuniões de planejamento, reuniões de análise de progresso, reuniões de revisão e de retrospectiva são realizadas de acordo com a especificação e características das mesmas, as quais poderão ser observadas nas próximas seções.
- ***Sprint planning:*** *d*urante esta reunião de planejamento, na qual todo o time *Scrum* deve estar presente, o trabalho a ser realizado durante a *Sprint* é planejado. Esta reunião deve ter, no máximo, oito horas de duração para uma *Sprint* de quatro semanas. Os resultados desta reunião resumem-se na definição do produto a ser entregue pela *Sprint* em questão e como o trabalho a ser realizado durante a *Sprint* será realizado.
- ***Daily scrum:*** a reunião diária do *Scrum* é um evento de curta duração chegando, no máximo, a quinze minutos diários. É realizada durante o início da jornada diária de trabalho dos membros do time de desenvolvimento, na qual todos expõem o que foi realizado e as barreiras encontradas desde a última reunião diária e o que será realizado até a próxima reunião diária. Segundo Schwaber e Sutherland (2011, p. 11): reuniões diárias melhoram as comunicações, eliminam outras reuniões, identificam e removem impedimentos para o desenvolvimento, destacam e promovem rápidas tomadas de decisão, e melhoram o nível de conhecimento do time de desenvolvimento. Esta é uma reunião chave para a inspeção e adaptação.
- ***Sprint review:*** *d*enominada revisão da *Sprint*, é realizada apenas ao final de cada *Sprint* intuitivamente a apresentar e inspecionar o produto gerado e adaptar o *Product Backlog* caso seja necessário. Esta reunião é caracterizada por sua informalidade, sua rápida duração – usualmente até quatro horas – e pelo possível refinamento do *Product Backlog*. Para tanto, participam desta reunião o time *Scrum* e as partes interessadas convidadas pelo *Product Owner*.
- ***Sprint retrospectiive:*** o último evento de uma *Sprint* destina-se à reunião de retrospectiva da mesma, na qual o time *Scrum* tem a oportunidade de avaliar a si próprio e identificar diretrizes de melhorias a serem adotadas durante as próximas *Sprints*. Esta reunião também é caracterizada por sua curta duração que, varia de projetos para projetos e que pode levar de trinta minutos a, no máximo, três horas. Todo o time *Scrum* participa de dada reunião colaborando com proposições de melhorias a serem implementadas a qualquer momento na *Sprint* seguinte.

Artefatos

Os artefatos do *Scrum* representam a matéria prima do *framework*, a qual deve ser o mais transparente possível para que todos os envolvidos com o projeto compreendam seus significados e características.

O *Scrum*, genericamente, é dotado de três artefatos indispensáveis: *Product Backlog*, *Sprint Backlog* e *Increment*.

- **Product backlog:** também é conhecido como *backlog* do produto, é o artefato mestre para o desenvolvimento do produto. Caracteriza-se por ser uma lista ordenada que lista todas as funcionalidades do produto são definidas conforme sua importância e necessidade para o cliente. O *Product Backlog* é inicialmente criado com as abstrações dos requisitos mais bem definidos e críticos

para o produto. Este artefato é dinâmico e sempre recebe incrementos que garantirão a apropriação, competitividade e utilidade do produto final. Seu gerenciamento e refinamento destina-se ao *Product Owner* e ao time *Scrum* que, em conjunto, adicionam detalhes, estimativas e ordenam os itens do *Product Backlog* conforme sua importância para o produto final.

- ***Sprint backlog:*** *a Sprint Backlog* é constituída por itens do *Product Backlog* que são selecionados para a *Sprint* definindo-se quais funcionalidades estarão presentes no próximo *release* e quanto trabalho demandará para incrementar tais funcionalidades à nova versão do sistema. Portanto, tal artefato torna visível o trabalho a ser realizado pelo time de desenvolvimento para que os objetivos da *Sprint* sejam atingidos. Para tal transparência, é realizado um plano pelo qual todos os membros do time de desenvolvimento terão acesso e poderão modificar sempre que um trabalho for necessário.
- **Increment:** este é o último estágio de uma *Sprint* no qual todo o trabalho realizado pelo time *Scrum* é terminado e somado aos outros itens já terminados do *Product Backlog*. Torna-se importante salientar que para que todo o trabalho realizado durante a *Sprint* seja considerado um incremento, este deve estar na condição de utilizável, independentemente da decisão do *Product Owner* de liberá-lo para uso ou não.

5 METODOLOGIAS ÁGEIS EM EMPRESAS DE PEQUENO A MÉDIO PORTE

Com o intuito de analisar o cenário de empresas desenvolvedoras de *software* brasileiras de pequeno a médio porte, foi realizada uma pesquisa exploratória levantando-se dados concernentes à atividade de desenvolvimento de *software* de cada organização, assim como a identificação do nível de conhecimento dos profissionais atuantes destas empresas em relação às metodologias de desenvolvimento ágeis e aos modelos de processos de desenvolvimento de *software* pautados pela engenharia de s*oftware*.

Não obstante, também foi realizado um estudo de caso, o qual se justificou pela implantação da metodologia *Kanban* para o desenvolvimento de *software* no Setor de Processamento de Dados do IE da Unicamp observando-se os benefícios obtidos e as dificuldades enfrentadas durante o processo de adaptação da empresa quanto à nova metodologia de trabalho.

5.1 Resultados da pesquisa realizada

Durante três meses, foi divulgado um formulário constituído por treze questões genéricas do âmbito da informática para a identificação de algumas informações concernentes ao ambiente de desenvolvimento de *software* das empresas como supracitado.

Nesta pesquisa objetivou-se analisar o ambiente de desenvolvimento de *software* das organizações pesquisadas assim como o nível de conhecimento de seus funcionários quanto à abordagem ágil de desenvolvimento de *software*. Quanto aos resultados, esperava-se que as organizações, de pequeno a médio porte, participantes da pesquisa, possuíssem o seu processo de desenvolvimento de *software,* embasados em técnicas inviáveis, para a demanda do mercado vigente e, também, que a bagagem intelectual de seus funcionários não fosse utilizada totalmente.

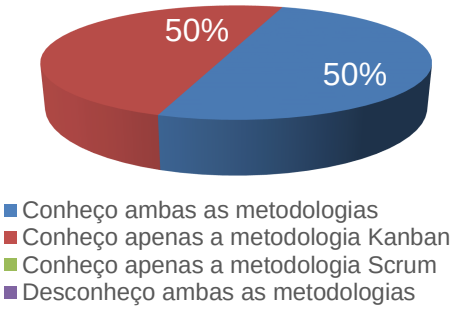
Contudo, foram convidados a participarem desta pesquisa, desenvolvedores de *software* de pequenas e médias empresas e, também, alguns funcionários de grandes empresas do ramo de desenvolvimento de *software* independentemente de sua origem – pública, privada, dentre outras.

Ao término do período de aplicação dos formulários houve 16 respostas, De acordo com as respostas do formulário foram criados alguns gráficos para melhor analisar os resultados obtidos.

Em uma primeira a análise da pesquisa realizada, constatou-se que todos os participantes desta responderam que reconhecem o que são metodologias ágeis de desenvolvimento de *software* e, consequentemente, quais seus princípios, valores e objetivos.

Porém, quando perguntou-se sobre o conhecimento de duas metodologias ágeis tradicionais mais encontradas em empresas de pequeno a médio porte – as metodologias *Kanban* e *Scrum* – os resultados foram surpreendentes. Como a figura 10 ilustra, todos os entrevistados afirmam conhecer pelo menos uma das duas metodologias.

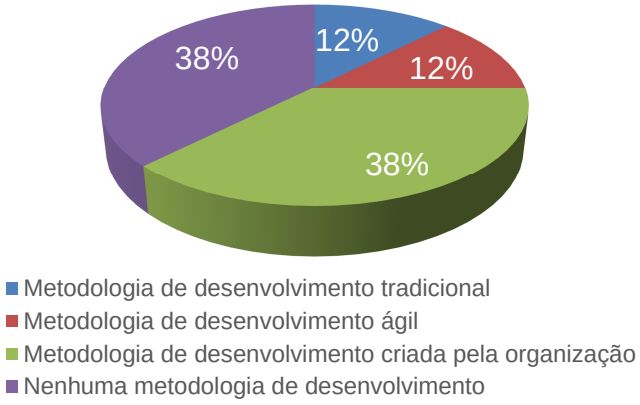
Figura 10 – Índice de conhecimento das metodologias ágeis *Kanban* e *Scrum*



Fonte: Pesquisa própria

Contudo, como os funcionários destas instituições desenvolvedoras de *software* afirmaram possuir um breve conhecimento sobre as duas metodologias ágeis de desenvolvimento mais vislumbradas em pequenas e médias empresas, esperava-se que as organizações em que atuavam usassem, em algum momento, alguma abordagem ágil de desenvolvimento de *software.*Entretanto, não foi o resultado supramencionado que se obteve. Conforme a figura 11 observa-se que apenas 12%, ou seja, dois dos entrevistados, realmente utilizam à risca, metodologias ágeis em suas organizações. Tais metodologias, em ambos os casos, resumem-se na metodologia de desenvolvimento *Scrum*. Vale ressaltar que estes dois casos pertencem a empresas de diferentes portes, sendo uma de pequeno e a outra de grande porte.

Figura 11 – Tipos de metodologia mais utilizadas pelas organizações

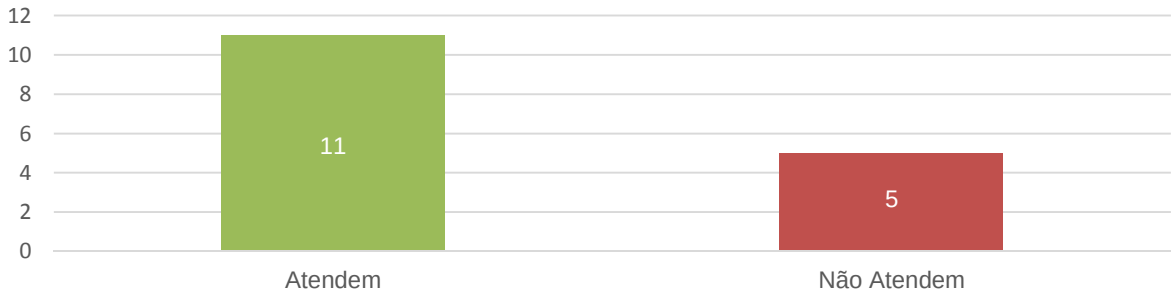


Fonte: Pesquisa própria

Além disso, observa-se que grande maioria das empresas, de acordo com os dados da pesquisa realizada, utiliza como base para o processo de desenvolvimento de *software* atividades que atendam às necessidades temporárias da organização. Em outras palavras, tais organizações não utilizam nenhuma metodologia de desenvolvimento ou, quando a utilizam, abstraem as atividades e processos que mais se adaptam à realidade organizacional dando origem, por sua vez, a um modelo de desenvolvimento de *software* único e repleto de abordagens empiristas.

Não obstante, quando se perguntou se as metodologias utilizadas pelas organizações atendiam suas necessidades, um novo dado surpreendente foi constatado: onze dos dezesseis participantes responderam que as metodologias utilizadas pela organização atendiam às suas necessidades (figura 12).

Figura 12 – Índice de atendimento às necessidades das organizações



Fonte: Pesquisa própria

Porém, após a análise minuciosa dos dados obtidos pela pesquisa, constata-se que os mesmos entrevistados que afirmaram que as metodologias atendiam às necessidades da organização relataram, em questões posteriores, que durante o processo de desenvolvimento de *software* são encontradas algumas dificuldades. Portanto, com o suporte de tais relatos, pode-se concluir que as metodologias utilizadas pelas organizações atendem parcialmente às necessidades das instituições desenvolvedoras de *software*.

Contudo, dentre os principais problemas relatados pelos entrevistados destacam-se: a falta do planejamento do projeto; a falta do gerenciamento do tempo; problemas de comunicação entre a equipe de desenvolvimento e os *stakeholders* do projeto; dificuldade de elucidar os reais requisitos demandados pelos clientes; o sobrecarregamento de tarefas aos funcionários; e, principalmente, a falta de um padrão de desenvolvimento.

Caracterizada por ser a última pergunta do formulário, levantou-se uma questão concernente a propostas de melhorias dos processos de desenvolvimento de *software* abordados por cada organização. Como resposta, obtivemos argumentos como: aumentar o nível de comprometimento da equipe; trabalhar melhor o *feedback* em todo o processo de desenvolvimento; reavaliar os requisitos após cada ciclo de desenvolvimento; dentre outras propostas de melhoria.

Nesse sentido, a partir de seus rituais, práticas e atividades, as metodologias de desenvolvimento de *software* ágeis buscam amenizar ou, até mesmo, solucionar os problemas supracitados e, também, almejam a melhoria do processo de desenvolvimento de *software* tornando-se uma das alternativas disponíveis para auxiliar ou implementar o processo de desenvolvimento de *software* das empresas.

5.2 Estudo de caso

Em um departamento público desenvolvedor de *software*, durante um período de seis meses, foi feito um experimento de implantação da metodologia de desenvolvimento ágil *Kanban* intuitivamente a sanar alguns dos problemas encontrados em seu processo de desenvolvimento de soluções em *software*. Entretanto, antes de tudo, torna-se necessário apresentar as características do processo que era utilizado pela organização assim como seus principais problemas e dificuldades.

Este departamento, mais especificamente o Setor de Processamento de Dados (SPD) do IE da Unicamp, pode ser considerado uma pequena empresa do ramo de informática constituída por uma equipe de analistas de sistemas e técnicos em informática, além de seus estagiários de nível médio e superior.

Dado departamento é responsável por dar suporte à informática para todo o instituto, seja da parte física (hardware) ou da parte lógica (software). Além disso, existem sistemas de uso interno, criados e mantidos pelo SPD, os quais são utilizados por funcionários, docentes e alunos do IE da UNICAMP.

Para tanto, uma equipe constituída por três analistas e, no máximo, quatro estagiários são responsáveis por dar suporte às aplicações criadas e mantidas por este departamento, além do site do IE.

O modelo de requisição de um novo sistema ou de alterações de sistemas já existentes é e continuará sendo feito por intermédio de chamados, realizados especificamente pelos usuários dos sistemas. Conforme os chamados são aprovados, um encontro com o requerente é agendado para que os requisitos sejam abstraídos e, logo em seguida, dá-se início ao processo de desenvolvimento da solução desejada.

De acordo com o modelo supracitado, alguns problemas eram originados como, por exemplo, a sobrecarga de atividades para os funcionários, a falta de organização das tarefas em andamento, a falta de identificação dos chamados já atendidos, a ausência do planejamento de testes do sistema, a demora na resolução do chamado, dentre outros problemas.

Portanto, a necessidade de implantação de um modelo que amenizasse a presença destes problemas foi trivial. Após a análise dos principais requisitos críticos para o sucesso deste estabelecimento, chegou-se à conclusão de que a metodologia de desenvolvimento ágil *Kanban* ofertaria importantes melhorias ao processo de desenvolvimento de *software* do departamento.

A preferência pela metodologia de desenvolvimento *Kanban* se deu pelo fato desta ser caracterizada por sua abordagem menos prescritiva quando comparada a outras metodologias como, por exemplo, a *Scrum* e, também, devido à sua flexibilidade e transparência.

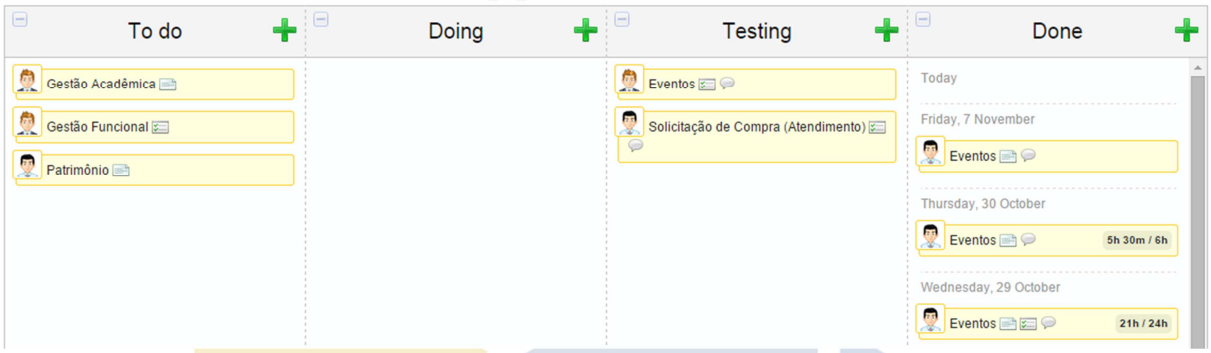
Contudo, após um breve estudo sobre tal metodologia, deu-se início a sua implantação. Durante um acordo entre os três analistas ficou decidido que dois deles ficariam responsáveis por dar suporte aos chamados referentes ao *site* do instituto enquanto um analista, conjuntamente com os estagiários, daria o suporte necessário aos chamados referentes aos sistemas.

De acordo com a metodologia *Kanban*, todo o processo de desenvolvimento deveria ser visível, ou seja, deveria ser transparente a todos os responsáveis pelos projetos. Portanto, um quadro *Kanban* foi utilizado seguindo os seguintes estágios: *to do*, *doing*, *testing* e *done*. Devido à falta de espaço físico, utilizou-se a ferramenta disponível gratuitamente na I*nternet* denominada *KanbanFlow*.

Além da divisão por estágios, foi acordado entre os membros do time de desenvolvimento que cada indivíduo poderia estar efetuando apenas uma atividade por vez, estagnando definitivamente, o problema de sobrecarga apresentado no antigo modelo de desenvolvimento de software.

A figura 13 apresenta o quadro de tarefas *Kanban* utilizado pelos membros da equipe de desenvolvimento do SPD.

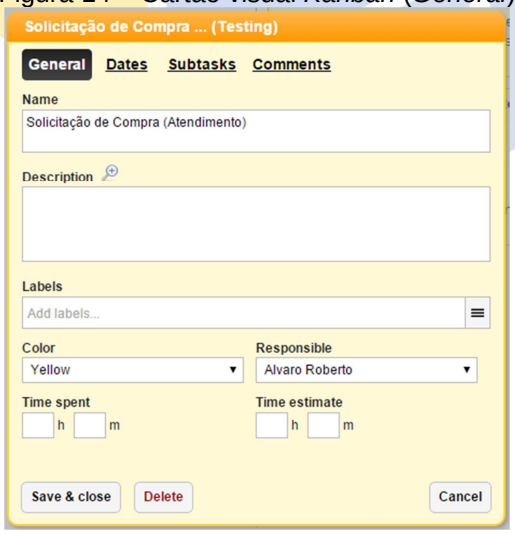
Figura 13 – Exemplo do quadro *Kanban*



Fonte: Imagem própria

Cada chamado, após sua aprovação, era transformado em cartões visuais seguindo-se algumas regras de nomenclatura. Na aba *General,* duas informações eram preenchidas: o *Name* e o *Responsible* do cartão visual. No campo *Name* era informado o sistema a que pertencia o chamado e no campo *Responsible* era informado o funcionário responsável por atender ao chamado, como exemplifica a figura 14.

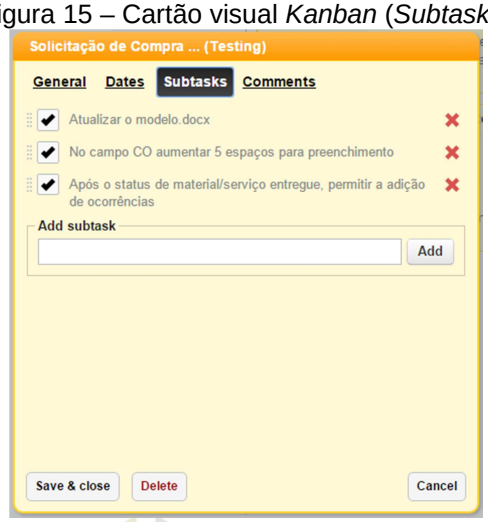
Figura 14 – Cartão visual *Kanban* (*General*)



Fonte: Imagem própria

Já na aba *Dates*, nenhuma das informações eram utilizadas. Na aba seguinte, intitulada *Subtasks*, era feito uma lista de checagem composta por todas as alterações/inclusões necessárias para que o chamado fosse atendido, assim como ilustra a figura 15.

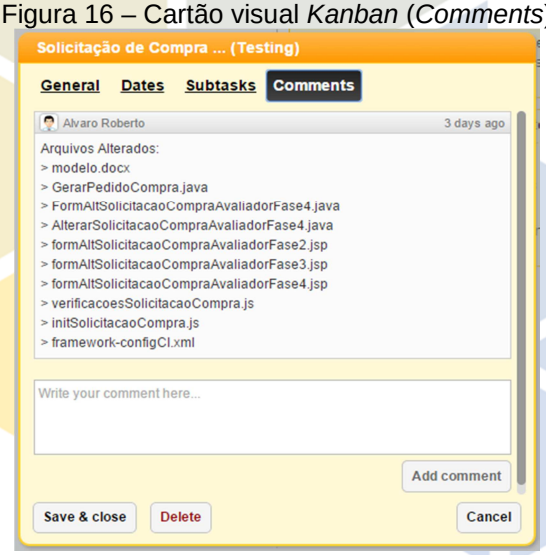
Figura 15 – Cartão visual *Kanban* (*Subtasks*)



Fonte: Imagem própria

Na última aba – *Comments* – foi acordado entre a equipe de desenvolvimento que fossem listados todos os arquivos alterados/incluídos ao decorrer das alterações/inclusões do sistema, conforme exemplifica a figura 16.

Figura 16 – Cartão visual *Kanban* (*Comments*)



Fonte: Imagem própria

Durante o período de implantação deste novo processo de desenvolvimento de soluções em *software*, pôde constatar-se um aumento no comprometimento dos usuários quanto às tarefas que, consequentemente, fez com que entregas fossem realizadas mais rapidamente.

Além deste benefício, o problema de sobrecarga de tarefas para um determinado funcionário extinguiu-se e trouxe a transparência de todo o processo de desenvolvimento de *software* da organização. Isto possibilitou que todos os chamados em espera, em desenvolvimento, em teste e finalizados pudessem ser vislumbrados por todos do time. Não obstante, constatou-se também um declínio sobre a quantidade de demandas por ajustes às alterações efetuadas devido ao emprego de testes de unidade o que poderá acarretar, em um futuro próximo, na utilização da técnica de Desenvolvimento Orientado a Testes.

Porém, algumas complicações foram encontradas durante a implantação deste novo método de desenvolvimento de *software*. O maior deles está relacionado à falta de um controlador de versões, pois, como existem alguns usuários trabalhando no mesmo sistema simultaneamente, os arquivos alterados por estes devem ser agrupados em uma versão final. Tal atividade é feito de forma manual fazendo uso da ferramenta comparadora do *IDE NetBeans*.

Mas, caracterizado por ser um dos problemas mais críticos desta organização é a falta de documentação dos sistemas, atividade menosprezada pelo SPD durante o processo de desenvolvimento de *software.*

6 CONSIDERAÇÕES FINAIS

De acordo com as exigências do mercado de *software* vigente, constatam-se demandas por *softwares* cada vez mais eficientes e com prazos de entrega cada vez mais curtos. Devido a esta realidade, a indústria desenvolvedora de *software* vem lidando com cenários desafiadores, os quais demandam um bom processo de desenvolvimento para atender à maioria destes aspectos da mutabilidade dos requisitos dos sistemas e das exigências cada vez mais árduas de seus clientes.

Para tanto, a abordagem ágil de desenvolvimento de *software* tem se empenhado muito bem, possibilitando que organizações sejam beneficiadas num período de médio e longo prazo. Porém, de acordo com os resultados da pesquisa realizada, devido à falta de recursos, medo da implantação de uma nova abordagem para o desenvolvimento de *software* ou, até mesmo, a preocupação com investimentos em outros setores da empresa, as organizações de pequeno a médio porte entrevistadas encontram-se utilizando métodos de desenvolvimento de *software* obsoletos, os quais não atendem, em sua totalidade, às demandas do mercado atual o que acarreta, na grande maioria dos casos, na entrega de *softwares* obsoletos e incompatíveis com as necessidades do cliente.

As metodologias de desenvolvimento ágeis foram criadas para divulgar uma nova abordagem de desenvolvimento de *software*, auxiliando a adaptação das empresas às demandas do mercado vigente, as quais priorizam a criação de *software* funcional, a colaboração do cliente durante o processo de desenvolvimento e a rápida resposta às mudanças do escopo dos projetos.

Porém, para que benefícios e soluções de problemas sejam concretizados, torna-se necessário seguir os conceitos, processos, e regras da metodologia de desenvolvimento ágil utilizada além de realizar um bom gerenciamento do projeto como um todo assim como o gerenciamento das complicações que forem advindas durante a execução do mesmo. Em outras palavras, é necessário conhecer bem a metodologia a ser implementada e possuir uma boa prática gerencial.

Além disso, é importante fazer uso de outras ferramentas que auxiliem e proporcionem a melhoria contínua do processo de desenvolvimento de *software* além de aproveitar a qualidade intelectual de seus funcionários, dando-lhes mais liberdade e tornando-os mais próximos da atividade gerencial da organização. Ferramentas citadas neste devido trabalho e que poderiam ser utilizadas para a melhoria dos processos de desenvolvimento de *software* de uma empresa são os modelos de melhoria CMMI e o MPS.BR.

Durante a implantação da metodologia *Kanban* para o desenvolvimento de software no SPD do IE da Unicamp, alguns benefícios puderam ser identificados mesmo com a simples implantação das principais atividades da metodologia *Kanban*.

Conforme alguns relatos das referências utilizadas para a composição deste trabalho, constatam-se grandes casos de sucesso quando uma abordagem ágil de desenvolvimento de *software* é atrelada a um modelo de melhoria dos processos de desenvolvimento de *software* como, por exemplo, os casos da *Google,* do *Yahoo,* da Globo, dentre outras organizações que fazem uso da metodologia ágil *Scrum* visando sempre a melhoria contínua dos processos de desenvolvimento de suas organizações.

Contudo, fica evidente que as organizações de pequeno a médio porte estão prestes a investir em abordagens ágeis de desenvolvimento de *software* intuitivamente a adequarem-se às necessidades e exigências do mercado o que proporcionará, direta ou indiretamente, o atendimento das necessidades do cliente, a resolução de problemas encontrados no processo de desenvolvimento de *software* e, também, a melhoria gradual e contínua do processo de desenvolvimento de *software* de dadas organizações.

6.1 Propostas para trabalhos futuros

Objetivando estender os conceitos abordados neste trabalho, são listadas a seguir algumas propostas para trabalhos futuros:

- Comparação entre abordagens ágeis e tradicionais de desenvolvimento de *software;*
- *Test Driven Development* (TDD);
- Modelos de melhoria de processos de desenvolvimento de software e seus benefícios;
- Os benefícios da utilização de um controlador de versões; e,
- Processos de melhoria da engenharia de requisitos.

REFERÊNCIAS

BECK, Kent. et al. ***Manifesto for agile software development***. 2001. Disponível em: <http://agilemanifesto.org>. Acesso em: 15 out. 2014.

BOEG, Jesper. ***Kanban* em 10 passos**: otimizando o fluxo de trabalho em sistemas de entrega de *software*. Trad. Leonardo Campos et al. São Paulo: InfoQ Brasil, 2012.

BRAUDE, Eric. **Projeto de s*oftware***: da programação à arquitetura: uma abordagem baseada em Java. Trad. Edson Furmankiewicz. Porto Alegre: Bookman, 2005.

BRODE, Cesar. ***Scrum***: guia prático para projetos ágeis. São Paulo: Novatec, 2013.

DIJKSTRA, Edsger Wybe. ***The humble programmer***. 1972, 17p. Disponível em: <http://www.cs.utexas.edu/users/EWD/ewd03xx/EWD340.PDF>. Acesso em: 22 maio 2014.

FERNANDES, Aguinaldo Aragon; ABREU, Vladimir Ferraz de. **Implantando a governança de TI**: da estratégia à gestão dos processos e serviços. 3.ed. Rio de Janeiro: Brasport, 2012.

HISTORIC Computer Images, 2000. Disponível em: <http://ftp.arl.army.mil/ftp/historic-computers/gif/eniac7.gif>. Acesso em: 22 maio 2014.

KNIBERG, Henrik; SKARIN, Mattias. ***Kanban* e *Scrum***: obtendo o melhor de ambos. Trad. Juliana Berossa Steffen. et al. São Paulo: InfoQ Brasil, 2010.

KOSCIANSKI, André; SOARES, Michel dos Santos. **Qualidade de s*oftware***: aprenda as metodologias e técnicas mais modernas para o desenvolvimento de *software*. 2.ed. São Paulo: Novatec, 2007.

MAGELA, Rogério. **Engenharia de s*oftware* aplicada**: princípios. Rio de Janeiro: Alta Books, 2006.

PRESSMAN, Roger S. **Engenharia de s*oftware***: uma abordagem profissional. Trad. Ariovaldo Griesi. 7.ed. Porto Alegre: AMGH, 2011.

SABBAGH, Rafael. ***Scrum***: gestão ágil para projetos de sucesso. São Paulo: Casa do Código, 2013.

SCHWABER, Ken; SUTHERLAND, Jeff. **Guia do *Scrum***: Um guia definitivo para o *Scrum*: as regras do jogo. 2011, 18p. Disponível em: <http://www.scrumguides.org/docs/scrumguide/v1/Scrum-Guide-Portuguese-BR.pdf>. Acesso em: 16 out. 2014.

SOMMERVILLE, Ian. **Engenharia de s*oftware***. 9.ed. São Paulo: Pearson, 2011.

Anderson Luiz Barbosa

Anderson Luiz Barbosa é doutorando em Engenharia Elétrica pela Universidade Estadual de Campinas; é mestre em Engenharia Elétrica - Telemática e Telecomunicações - pela Universidade Estadual de Campinas, mestre em Informática - Gerenciamento de Sistemas Informação - pela Pontifícia Universidade Católica de Campinas, especialista em Metodologia do Ensino Superior pelo Centro Universitário Salesiano de São Paulo, bacharel em Análise de Sistemas pela Pontifícia Universidade Católica de Campinas e técnico em Processamento de Dados pelo Colégio Técnico da Universidade Estadual de Campinas. É professor titular do Centro Universitário Salesiano de São Paulo - Unidade de Ensino de Americana e Unidade de Ensino de Campinas - Campus São José, onde ocupa o cargo de Diretor de Operações, professor titular (Pleno II) da Faculdade de Tecnologia (FATEC) de Americana, professor visitante da Universidade Católica de Brasília e professor titular da Faculdade Salesiana Dom Bosco de Piracicaba. Dentre os principais temas de atuação e pesquisa estão: Avaliação Institucional, Gestão Educacional, Tecnologia Educacional, Educação a Distância, Engenharia de Software, Gerenciamento de Projetos e Sistemas de Informação. É Avaliador Institucional e de Cursos de Graduação do Ministério da Educação.

Contato: anderson.barbosa@sj.unisal.br

Fonte: CNPQ – Curriculo Lates