ANÁLISE COMPARATIVA DA UTILIZAÇÃO DO MODELO TRADICIONAL (*WATERFALL*) DE DESENVOLVIMENTO DE PROJETOS E O MODELO ÁGIL (*AGILE*) EM FÁBRICAS DE *SOFTWARE*

Mario Augusto Rivas
Faculdade de Informática e Administração Paulista – FIAP/Brazil
mario.rivas@citi.com

Enock Godoy de Souza
Faculdade de Informática e Administração Paulista – FIAP/Brazil
enock.godoy@fiap.com.br

***Resumo*: Nos últimos anos a utilização das metodologias ágeis na indústria da tecnologia da informação tem aumentando consideravelmente, devido entre outros fatores à alta produtividade e qualidade dos produtos elaborados. Neste cenário, fábricas de *software* que tradicionalmente utilizavam a metodologia tradicional nas iniciativas de desenvolvimento, estão começando a aplicar frameworks ágeis tais como *Scrum* ou XP. O presente artigo desenvolve uma análise comparativa entre o modelo tradicional (*waterfall*) e o modelo ágil no contexto das fábricas de *software*, destacando os principais tópicos que devem ser considerados na avaliação da metodologia, por meio de uma matriz comparativa.**

***Palavras-chave*: Agile; Scrum; Waterfall; Fábrica de Software.**

***Abstract*: Over the last few years Agile methodologies usage by the information technology industry considerably increased, due to the high productivity and quality of the products created, among other factors. In this scenario, software factories that traditionally applied waterfall methodology for development initiatives are starting to use Agile frameworks as Scrum or XP. This article presents a comparative analysis between the traditional model (waterfall) and the Agile model on the software factories context, describing the main topics that must considered in order appraise a proper methodology, through a comparative matrix.**

***Keywords*: Agile; Scrum; Waterfall; Software Factory.**

I. INTRODUÇÃO

A metodologia tradicional de desenvolvimento de *software* (*waterfall*) baseia-se numa sequência de estágios de produção independentes, que têm por objetivo entregar ao final desta o produto definido na fase inicial. Este processo linear de desenvolvimento é apontado como um dos responsáveis pelo baixo índice de sucesso dos projetos de *software*, em razão de atores tais como a imprevisibilidade dos requisitos, mudança continua do ambiente e a utilização de métodos orientados a processos em vez de pessoas [1].

As metodologias ágeis em contrapartida, estão orientadas a flexibilizar a cadeia produtiva de *software*, no intuito de focalizar no produto em detrimento da burocracia do processo. Perante esta situação, surge nas fábricas de *software* a necessidade de entender a aplicabilidade destas metodologias como fator de vantagem competitiva dentro do contexto operacional e comercial.

A. *Objetivo*

O objetivo principal deste trabalho é comparar a implantação das principais premissas dos modelos tradicional (*Waterfall*) e ágil (*Agile*) nas fábricas de *software*, considerando a taxonomia e o contexto destas.

B. *Problema*

Muitos artigos e trabalhos acadêmicos têm focalizado na problemática da migração do modelo tradicional para o modelo ágil, porém não foi possível achar artigos que tratem o processo de análise que antecede à decisão estratégica da migrar. Alguns autores destacam a resistência que pode acontecer nas organizações que adotem o modelo ágil, por conta da incerteza inerente ao custo e duração do projeto, embora o modelo tradicional também não ofereça nenhuma assertividade nestes quesitos [1]. Outros autores, concluem que "estrutura e culturas organizacionais orientadas à inovação podem se engajar mais facilmente no modelo ágil que as organizações construídas em base a burocracia e formalização [2]. As organizações devem avaliar cuidadosamente se estão prontas para percorrer o caminho ágil".

Dito processo decisório é influenciado não somente pelas diferencias estruturais dos modelos senão também pelo contexto da própria fábrica de *software* e do mercado, variáveis que não podem ser generalizadas.

A relevância deste trabalho reside em apresentar uma versão consolidada da comparação dos modelos tradicional e ágil, desde o ponto de vista funcional das

fábricas de *software*, no intuito de facilitar a tomada de decisão nas organizações.

C. *Hipótese*

A hipótese sugerida é que a escolha de uma metodologia de desenvolvimento depende de fatores intrínsecos do projeto e da fábrica de *software*, e sua aplicabilidade não pode ser generalizada sem considerar o contexto interno e externo da organização.

A metodologia proposta para desenvolver este trabalho baseia-se na pesquisa bibliográfica tradicional do tipo conceitual, contemplando a gestão de fábricas de *software* e os modelos de gestão de desenvolvimento tradicional e ágil. Uma revisão bibliográfica tradicional conceitual busca sintetizar diferentes áreas de conhecimento conceitual, contribuindo para um melhor entendimento das questões [3].

D. *Justificativa Fundamentada*

O desenvolvimento de *software* nos últimos 30 anos evoluiu do conceito artesanal vinculado a indivíduos altamente técnicos trabalhando isoladamente, ao surgimento de fábricas de *software* com times colaborativos distribuídos globalmente.

As metodologias de desenvolvimento não acompanharam o mesmo ritmo de mudanças, gerando uma defasagem entre processo e tecnologia que só veio a ser evidenciada com a aparição dos modelos ágeis. Neste contexto as fábricas de *software* enfrentam a necessidade de adotar novas metodologias para evitar a perda de posicionamento no mercado, incluindo a utilização de modelos ágeis, eventualmente sem a devida avaliação organizacional prévia.

II. FÁBRICAS DE *SOFTWARE*

O desenvolvimento de *software* era originalmente uma tarefa realizada por técnicos individuais especializados na linguagem de programação, no sistema operacional e no próprio computador. A imprevisibilidade do desempenho dos programadores levou a Robert Bemer a criar o termo "fábrica de software", sugerindo a criação destas unidades nas organizações, utilizando suas próprias ferramentas e processos [4].

A. *Origem e Definição*

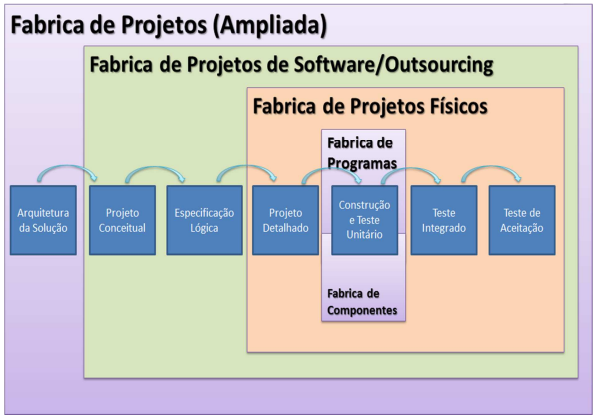
O conceito de "fábrica de software" evoluiu nos anos 1960 e 1970, nos Estados Unidos e no Japão [5]. A primeira fábrica de *software* foi a Hitachi Software em 1969, pondo em prática a ideia dos executivos dessa empresa de separar o grupo de desenvolvimento de *software* do restante da empresa. Nos Estados Unidos a System Development Corp (SDC), que seria integrada à Unisys mais tarde, implantou na sua fábrica de *software* de Santa Monica uma organização baseada em três elementos: um conjunto integrado de ferramentas; procedimentos e políticas de gerenciamento padronizadas e uma organização matricial. [6].

O termo "fábrica de *software*" pode ser definido como a organização previsível e controlada dos elementos que compõem o desenvolvimento de *software*, incluindo processos, recursos e conhecimento [6]. Esta definição enquadra as fábricas de *software* numa dimensão maior do que simplesmente a produção de código fonte, abrangendo potencialmente fábricas de testes ou especificações, por exemplo.

B. *Modelos operacionais*

As fábricas de *software* podem ser classificadas de acordo com seu escopo de fornecimento [7]. Fábricas dedicadas somente à codificação e teste unitário compõem o segmento menor nesta classificação, seguido em ordem crescente pelas fábricas de projetos físicos que incluem o desenho detalhado e os testes integrados e de aceitação, e pelas fábricas de projetos de *software* que abrangem todas as atividades desde o projeto conceitual. Fernandes ainda descreve um nível superior de "fábricas de projetos ampliadas", cujos escopos transcendem a visão tecnológica. O modelo de *outsourcing* de sistemas como especialização da fábrica de projetos e a fábrica de componentes, também fazem parte desta classificação [8].

Fig. 1. Classificação das fábricas de *software* (adaptado de [7]).



Todos os tipos de fábrica de software, independente da classificação, possuem processos definidos que atendem às necessidades da linha de produção. No caso das fábricas de programas, por exemplo, o processo central refere-se à construção e teste unitário dos programas, e deve estar baseado na adoção de alguns dos frameworks de mercado (*waterfall*, RUP, Scrum, etc.). Isto não é necessariamente válido para as fábricas de projetos de software, nas quais o processo central é o gerenciamento de projetos e os *frameworks* adotados são outros (PMBOK, Prince2, etc.).

III. MODELO TRADICIONAL DE DESENVOLVIMENTO

Até meados dos anos 60 o papel do *software* foi secundário em relação ao hardware, na indústria da computação. Esta situação começou a mudar com a chegada das primeiras linguagens de programação

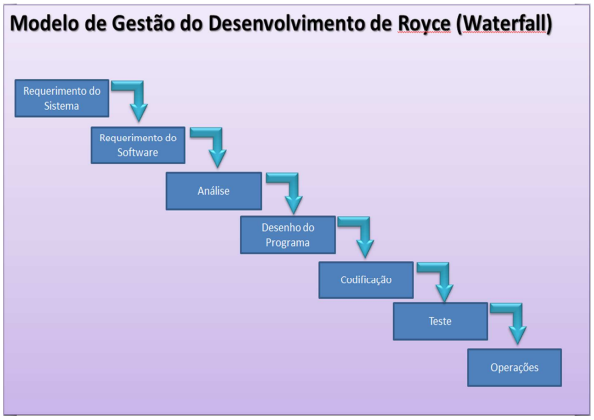
teoricamente independentes do hardware (Fortran, Algol, Cobol, etc.), abrindo possibilidades para a reutilização do *software* e das técnicas de programação.

No ano 1970 o Dr. Winston Royce publicou o artigo original que deu origem ao framework mais popular de desenvolvimento de *software*, chamado *waterfall* (cascata) [9]. Este modelo baseia-se na sequência de atividades padronizadas cujos resultados individuais são utilizados como pré-requisito para a atividade subsequente, graficamente descrito como uma cascata que começa com os requisitos do sistema e termina com a operação (execução) deste.

A. *O Modelo de Gestão de Desenvolvimento do Dr. Winston Royce*

Embora a adoção do modelo do Dr. Royce (*waterfall*) fosse quase total na comunidade do software, os conceitos nele expressados foram simplificados ou reduzidos num modelo linear conhecido como "cascata".

Fig. 2. Modelo Waterfall (adaptado de [9]).



O desenho original do Dr. Royce envolve a iteração entre fases consecutivas do *waterfall*, no entendimento de que o resultado de cada fase pode ter impacto na fase anterior. Por exemplo, na fase de desenho podem ser encontrados elementos novos que impactem a análise feita, logo a fase de análise precisa ser revisitada. Isto pode parecer obvio no relacionamento entre as fases de teste e codificação, porém para o resto do processo não é considerado com fines de planejamento. Eventualmente o Dr. Royce destaca o risco inerente deste modelo: "Eu acredito neste conceito, mas a implementação descrita acima [*waterfall*] possui riscos e convida às falhas" [9].

Na opinião do autor do modelo, o maior risco é o fato de que as iterações transcendam as fases consecutivas e gerem uma revisão total do desenho (*design*), pois, caso exista uma incompatibilidade entre os novos requisitos e o desenho atual, as mudanças necessárias podem ser tão disruptivas, que impliquem em um completo redesenho do sistema [9]. A proposta do Royce para mitigar os riscos decorrentes do modelo é composta pelos seguintes passos: 1) criar uma fase de desenho preliminar do *software* antes da fase de análise; 2) gerar documentação abrangente do desenho; 3) a partir do desenho preliminar fazer uma simulação completa do processo (flexibilizando os requisitos de documentação), para chegar num protótipo ou versão "1" que possa ser utilizada para o processo completo; 4) planejar, controlar e monitorar o teste e 5) envolver ao cliente [9].

B. *Críticas ao Modelo Tradicional*

Alguns autores argumentam que o método *waterfall* sempre foi efetivo em projetos com requisitos bem definidos, tecnologia bem conhecida e baixo risco geral [10]. Neste contexto, o escopo, os requisitos, a tecnologia, as tarefas requeridas e os níveis de recursos são suficientemente previsíveis para que o projeto possa ser totalmente planejado antecipadamente. O modelo *waterfall* fundamentalmente assume que os "sistemas são totalmente especificáveis, previsíveis e podem ser feitos por meio de um meticuloso e extensivo planejamento" [2].

Estas premissas referidas pelos autores acima são claramente conflitantes com a realidade da grande maioria dos projetos de *software* nos quais o fator humano (originalmente menosprezado pelos metodologistas do *software* dos anos 60 e 70) revela-se como pouco previsível: "…esta técnica [*waterfall*] requer que todas as boas ideias apareçam ao começo do ciclo de desenvolvimento, estágio no qual podem ser incorporadas ao plano. Porém, como todos nós sabemos, as boas ideias aparecem espontaneamente ao logo do processo – no começo, na metade e mesmo um dia antes da implantação...Com a técnica *waterfall*, uma boa ideia que apareça tarde no ciclo de desenvolvimento não é um presente, é uma ameaça" [11].

IV. MODELO ÁGIL

Em Fevereiro de 2001, dezessete metodologistas do *software* "leve" e pensadores se reuniram para discutir e chegar a um consenso que refletisse suas ideias e conceitos em relação à situação atual do desenvolvimento de *software*. Entre eles estavam os cofundadores do *Scrum* Jeff Sutherland e Ken Schwaber, junto com Mike Beedle que trabalhou nos padrões iniciais do *Scrum* e foi coautor do primeiro livro de *Scrum*. O grupo denominou-se 'The Agile Alliance' e estampou o acordo no documento 'Manifesto for Agile Software Development', influenciado pelas melhores práticas da indústria japonesa, particularmente pelos princípios de desenvolvimento enxuto (*lean*) de empresas como Toyota e Honda [12].

A. *O Manifesto Ágil*

Longe de ser uma guia metodológica ou um framework, o Manifesto Ágil é considerado uma intenção manifesta e coletiva dos seus autores, de

impulsionar uma visão inovadora na engenharia de *software* [12]:

"Estamos descobrindo maneiras melhores de desenvolver *software*, fazendo-o nós mesmos e ajudando outros a fazerem o mesmo. Por meio desse trabalho, passamos a valorizar:

Indivíduos e interações mais que **processos e ferramentas.**

Software em funcionamento mais que **documentação abrangente.**

Colaboração com o cliente mais que **negociação de contratos.**

Responder a mudanças mais que **seguir um plano.**

Ou seja, mesmo havendo valor nos itens à direita (do termo 'mais que'), valorizamos mais os itens à esquerda." [12].

Este enfoque da disciplina do desenvolvimento de *software*, radicalmente oposto à crença tradicional que enxerga a eficiência na produção de sistemas como fruto da evolução dos processos de engenharia, teve de enfrentar vários paradigmas até então considerados axiomáticos entre a comunidade.

Claro exemplo desta situação é o reconhecimento tácito da dinâmica dos requisitos enunciada anos antes no princípio de incerteza dos requisitos de Humphrey: "para um sistema de *software*, novo os requisitos não serão completamente conhecidos até depois que os usuários o tenham utilizado" [13]. O Manifesto *Agile* não só reconhece esta situação, mas também assume a necessidade de integrar as mudanças de requisitos em todo o processo de desenvolvimento, visando colaborar com o cliente na obtenção da melhor vantagem competitiva. Outro paradigma questionado por este movimento refere ao papel do cliente no processo de desenvolvimento, que tradicionalmente estava delimitado a definir os requisitos de negocio e receber o produto já terminado no outro extremo da cadeia de produção. O Manifesto *Agile* sugere a participação diária do cliente durante todo o processo, não só definindo os requisitos como também participando das decisões que afetam o produto a ser criado [12].

Nos anos subsequentes à criação da *Agile Alliance*, surgiram em Europa e nos Estados Unidos diversos *frameworks* de desenvolvimento, orientados a difundir o Manifesto *Agile* nas organizações, dos quais o mais popular até hoje é o *Scrum*.

B. Scrum

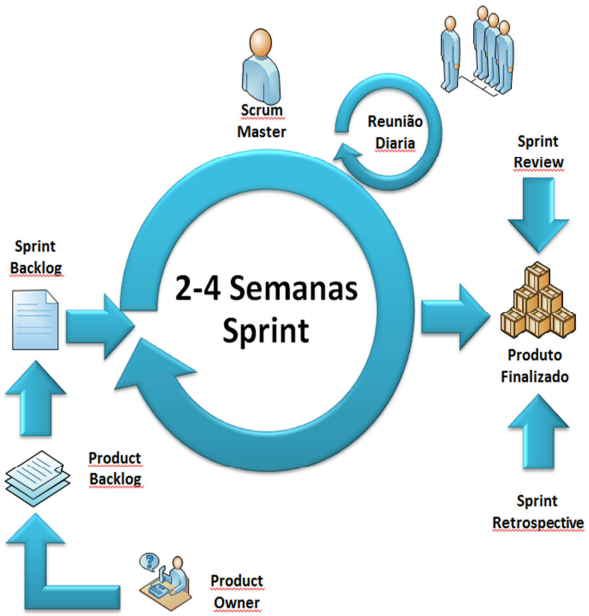
De acordo com um dos criadores, o *Scrum* é um *framework* simples utilizado para organizar as equipes e fazer o trabalho de forma mais produtiva e com alta qualidade [14]. A origem do termo refere a uma formação clássica do *rugby*, na qual um grupo pequeno, coeso e organizado de jogadores, consegue elencar os esforços individuais num objetivo comum. O *Scrum* focaliza na geração do valor agregado para o cliente, eliminando o desperdiço e priorizando a entrega periódica de *software* funcional. Essa entrega periódica decorre da estrutura iterativa do processo, conformada por períodos de duas (2) a quatro (4) semanas chamados *Sprints*, cada um dos quais se encerra com a entrega de um incremento no produto final [15].

O Scrum possui três (3) papéis, três (3) cerimônias e três (3) artefatos. O *Product Owner* (dono do produto) tem como responsabilidade definir os requisitos do produto; decidir a data de implantação e o conteúdo; priorizar os requisitos de acordo ao valor agregado e aceitar ou rejeitar os resultados do trabalho. Embora o *framework* não inclua a participação de um gerente de projeto, o papel de líder da equipe é adjudicado ao *Scrum Master*, quem deve atuar como um facilitador assegurando que a equipe esteja produtiva e funcional; removendo barreiras e blindando a equipe de interferências externas, além de assegurar que o framework seja seguido. O terceiro papel é a própria equipe de desenvolvimento, que tem por responsabilidades a produção do trabalho, sendo auto-organizada em todos os aspectos [15].

O *Scrum* prevê uma cerimônia ou reunião ao começo de cada iteração denominada *Sprint Planning*, na qual são definidos os requisitos que a equipe irá entregar ao final do *sprint*. Vale ressaltar que as reuniões do *Scrum* têm sido chamadas de cerimônias pelos praticantes deste *framework* no Brasil, pois os treinamentos oficiais têm usado majoritariamente essa terminologia. Todos os dias durante o *sprint* acontece a reunião diária de alinhamento conhecida como *Daily Scrum Meeting*, geralmente feita numa sala sem cadeiras (*standup meeting* ou reunião em pé) para estimular a objetividade da reunião. O objetivo desta é que cada membro da equipe responda as perguntas: a) Que foi feito ontem? b) Que será feito hoje? e c) Quais os problemas que estão sendo enfrentados? A terceira cerimônia requerida pelo *Scrum* é o *Sprint Review Meeting*, que tem por objetivos revisar o que está sendo entregue ao cliente e analisar o funcionamento da equipe durante o *sprint*, também chamado de *Sprint Retrospective* [15].

Alinhado com as premissas do Manifesto *Agile* [12] que priorizam o *software* em funcionamento mais que a documentação abrangente, *Scrum* exige somente três (3) artefatos documentais: o *Product Backlog* que contém os requisitos ainda não criados do produto, o *Sprint Backlog* descrevendo os requisitos a serem desenvolvidos no *sprint* e o *Burndown Chart*, gráfico que expõe o avanço na execução dos requisitos ao longo do *sprint* e serve como status do projeto [15].

Fig. 3. O ciclo do Scrum (adaptado de [14]).



C. *Aplicabilidade do Scrum em projetos de grande escala e missão crítica*

Embora existam numerosos casos de sucesso em implantações de *Scrum* [16], ainda persistem na comunidade acadêmica e na indústria de tecnologia de informação dúvidas respeito à aplicabilidade deste *framework* em projetos de grande escala e/ou missão crítica [16]. Alguns autores argumentam que a maioria dos projetos desenvolvidos com *Scrum* é de pequena escala e relativamente simples, com grande sucesso [17]. Entretanto, existem poucos e muito limitados estudos referentes à aplicabilidade do *Scrum* em grandes projetos [17].

Na opinião dos autores, ainda não existem dados conclusivos para determinar a aplicabilidade do *framework* neste tipo de projetos, porém sugerem os seguintes pontos a serem considerados [17]:

- Investir tempo na composição idônea das equipes, considerando os projetos nos quais trabalharão;
- Escolher um *Scrum Master* que não só remova os impedimentos para avançar no projeto, senão que seja capaz de gerenciar a consistência do produto horizontalmente entre os times do projeto;
- Dedicar suficiente tempo para o planejamento e desenho;
- Adaptar o *framework* de acordo á natureza do projeto.

Outra dimensão importante que deve ser considerada em projetos de grande escala advém do fato comum na indústria de *software* de terceirizar segmentos do desenvolvimento, muitas vezes para locações distantes geograficamente como Índia e China [18]. Os autores sugerem a adaptação eventual do *framework* ágil escolhido para resolver problemas decorrentes da separação dos times. “As práticas ágeis modificadas resolvem este problema [a separação geográfica] dissolvendo algumas das práticas do *framework* enquanto se mantém a essência dos princípios e valores ágeis, e fornecendo uma nova estrutura de colaboração. Esta nova estrutura não é burocrática, porem pode ser ajustada em base às necessidades e níveis de confiança da organização” [18].

V. ANALISE COMPARTIVA DOS MODELOS

De acordo com o exposto anteriormente, o modelo ágil para gestão do desenvolvimento possui diferenças significativas em relação ao modelo tradicional ou *waterfall*, às quais serão explicitadas em relação aos tipos de fábricas de *software* propostos em [7].

A. *Gestão dos Requisitos e Escopo*

O modelo tradicional tem como premissa a preexistência dos requisitos do cliente detalhados e completos, assumindo que todo o contexto pode ser definido a priori pelo demandante, i.e. o cliente. Os requisitos no modelo ágil são dinâmicos e podem ser modificados durante todo o processo de desenvolvimento. “Mudanças nos requisitos são bem-vindas, mesmo tardiamente no desenvolvimento. Processos ágeis tiram vantagem das mudanças com o propósito de alcançar vantagem competitiva para o cliente” [12]. Embora o impacto da gestão dos requisitos seja maior para fábricas de programas ou componentes, em geral é o modelo comercial destas que define o impacto real. Modelos de contratação “fixed price/fixed time” são claramente os mais afetados.

B. *Processo e Controle*

O Manifesto Ágil prioriza as interações e os indivíduos perante as ferramentas e os processos. Neste ponto a diferença em relação ao modelo tradicional se manifesta claramente, devido ao fato deste último ser totalmente atrelado a processos estritos, similares a uma linha de produção industrial. A necessidade de reconhecer o individuo como elemento fundamental da produção no modelo ágil, se contrapõe à visão mecanicista do modelo tradicional, na qual a aderência aos processos de engenharia é a chave do sucesso. Isto se reflete também nos perfis de liderança sugeridos pelas metodologias baseadas nos modelos tradicional e ágil. Enquanto no modelo tradicional a liderança é manifesta a traves do comando e controle, normalmente personificado no papel do Gestor do Projeto (*Project Manager*), o *Scrum* define o *Scrum Master* como um

papel de facilitador que ajuda ao time alcançar as metas e que tem escassa autoridade formal.

C. *Envolvimento do Cliente*

No modelo tradicional o cliente cumpre um papel importante no começo e no fim do projeto, quando são feitas as definições dos requisitos e a homologação dos resultados, respectivamente. Muito pelo contrário, nos projetos gerenciados no modelo ágil, o cliente é um papel fundamental, participando ativamente ao longo de todo o processo.

Por definição, as fábricas de software são estruturas lógicas isoladas das áreas de negócios ou incluso empresas externas à organização cliente, motivo pelo qual o envolvimento direto do cliente acontece (geralmente) nos eventos administrativos padrão (contratação, pagamentos, etc.). A vinculação permanente dentro do processo da fábrica, como sugere o modelo ágil, envolve certo grau de dificuldade para as fábricas externas à organização cliente, e se faz pouco produtivo para fábricas de menor porte (programas, componentes e projeto físico).

D. *Gestão da Documentação*

Segundo o Manifesto *Agile* [12], a prioridade sempre é o *software* em funcionamento, mais que a documentação extensiva, logo a gestão da documentação ocupa um lugar menos importante na gestão do projeto, até mesmo porque o próprio código gerado deve servir como documentação tácita. Contrariamente, o modelo tradicional adota uma política de documentação explícita. “Ocasionalmente sou solicitado para revisar o progresso de outros desenhos de *software*. Meu primeiro passo é investigar o estado da documentação. Se a documentação está em falta minha primeira recomendação é simples. Substitua o gerente do projeto. Pare todas as atividades que não sejam relacionadas à documentação. Coloque a documentação no padrão adequado. A gestão do *software* é simplesmente impossível sem um grande volume de documentação” [9].

Para as fábricas de programas ou componentes, a documentação é de caráter técnico e mudanças do modelo tradicional para o modelo ágil não deveriam representar maiores impactos. Enquanto aos outros tipos de fábrica de *software*, o impacto é diretamente proporcional à complexidade e diversidade das atividades nelas desenvolvidas.

E. *Matriz de Comparação*

A tabela 1 exibe um resumo dos impactos decorrentes da aplicação das premissas acima discutidas, nos distintos tipos de fábrica de *software*.

Tabela 1: Impacto comparativo da adoção das premissas dos modelos tradicional e *Agile* nos distintos tipos de fábrica de *software* (matriz elaborada pelo autor com base em [7] e [14]).

		Classificação das Fábricas de *Software*			
		Programas / Componentes	Projetos Físicos	Projetos de *Software* / Outsourcing	Projetos (ampliada)
Gestão de Requisitos	Tradicional	(+) Requisitos definidos antecipadamente.	(+) Requisitos definidos antecipadamente.	(+) Requisitos gerenciados internamente	(+) Requisitos gerenciados internamente
	Ágil	(-) Dificuldade para aceitar modificações no meio do processo	(=) Requisitos modificados dentro e fora do processo.	(+) Requisitos modificados dentro do processo de fábrica	(+) Requisitos modificados dentro do processo de fábrica
Processos e Controle	Tradicional	(=) Gerente de projeto responsável	(=) Gerente de projeto responsável	(=) Gerente de projeto responsável	(=) Gerente de projeto responsável
	Ágil	(+) Sem necessidade de gerente de projeto formal	(+) Sem necessidade de gerente de projeto formal	(+) Sem necessidade de gerente de projeto formal	(+) Sem necessidade de gerente de projeto formal
Envolvimento do Cliente	Tradicional	(+) Sem participação do cliente no processo da fábrica	(=) Pouca participação do cliente no processo da fábrica	(-) Participação ativa do cliente na fase inicial e na homolocação	(-) Participação ativa do cliente na fase inicial e na homolocação
	Ágil	(-) Cliente ativamente participando do sprint	(+) Cliente ativamente participando em todo o processo	(+) Cliente ativamente participando em todo o processo	(+) Cliente ativamente participando em todo o processo
Gestão da Documentação	Tradicional	(=) Pouca documentação, só caráter técnico	(=) Vários documentos técnicos mandatórios	(-) Numerosos documentos técnicos e de negócio	(-) Numerosos documentos técnicos e de negócio
	Ágil	(=) Gestão do sprint backlog	(+) Gestão do Product Backlog, Sprint Backlog e Burndown chart	(+) Gestão do Product Backlog, Sprint Backlog e Burndown chart	(+) Gestão do Product Backlog, Sprint Backlog e Burndown chart

Legenda: (+) impacto positivo, (-) impacto negativo, (=) sem impacto.

VI. CONCLUSÃO

Nos últimos anos observou-se uma tendência importante de adoção das metodologias ágeis no mercado das fábricas de *software*. Numerosos trabalhos acadêmicos tem abordado esta temática do ponto de vista operacional, porém nem sempre a comparação entre os modelos tradicional e *Agile* foi contextualizada no conceito de fábrica de software. As fábricas podem ser classificadas segundo o escopo de atuação em fábricas de programas e/ou componentes, fábricas de projetos físicos, fábricas de projetos de *software* e/ou *outsourcing*, e fábricas de projetos ampliadas [7].

O modelo tradicional desenvolvido a partir do trabalho de Royce, mais conhecido como *Waterfall* ou cascata, foi durante muitos anos o padrão para a gestão do desenvolvimento [9]. Sua estrutura linear, semelhante a uma linha de produção industrial, e fortemente separada em etapas isoladas, facilitou sua rápida assimilação. À medida que a engenharia de *software* evoluiu, falhas importantes foram encontradas neste modelo, originárias em suas premissas relativas à imutabilidade dos requisitos decorrente do modelo linear.

Em resposta a essa situação, um grupo de metodologista de *software* redigiram o Manifesto *Agile* [12], definindo as bases conceituais do modelo ágil. Este

modelo prioriza a interação e as pessoas, mais que as ferramentas e os processos, o *software* em funcionamento mais que documentação abrangente, colaboração com o cliente mais que negociação de contratos e resposta às mudanças mais que seguir o plano. O modelo ágil reconhece que os requisitos vão mudar constantemente ao longo do processo de desenvolvimento, em parte devido à própria natureza dos seres humanos que trabalham nele, e faz questão de aproveitar a evolução dos requisitos para entregar um produto de maior valor para o cliente.

Na comparação entre os dois modelos surgiram quatro (4) tópicos de destaque: gestão de requisitos, processos e controle, envolvimento do cliente e gestão da documentação. Para cada um destes, foi feita uma comparação contextualizada em cada tipo de fábrica de *software* (representada na tabela 1) e ponderada segundo o impacto que teria para ser implantada em cada instância. Foi constatado que as fábricas de programas/componentes não apresentam vantagens importantes na adoção das premissas do modelo ágil; pelo contrário, em razão do escopo reduzido de atuação, fatores como a estabilidade dos requisitos resultam muito favoráveis. Contrariamente, as mesmas premissas aplicadas às fábricas de projetos físicos sugerem um panorama mais favorável para o modelo ágil por conta da flexibilidade dos processos e da redução da documentação. Esta tendência se intensifica no caso das fábricas de projetos de *software* e das fábricas de *software* ampliada, nas quais o envolvimento do cliente no processo de desenvolvimento pode representar uma importante vantagem competitiva.

Em razão das limitações deste artigo, grandes generalizações foram assumidas na criação da matriz comparativa, motivo pelo qual as recomendações nela contidas não devem se considerar como regra geral senão como elemento de contexto na tomada de decisões estratégica. Elementos como o modelo comercial da organização, regulamentações locais e contexto tecnológico, não foram avaliados explicitamente, porém são fatores decisórios importantes na escolha do modelo de desenvolvimento da fábrica de *software.*

De acordo ao exposto como hipótese na introdução deste artigo, confirma-se que a escolha de uma metodologia de desenvolvimento depende de fatores intrínsecos do projeto e da fábrica de *software*, e sua aplicabilidade não pode ser generalizada sem considerar o contexto interno e externo da organização. A matriz de comparação exposta neste artigo deve servir como marco de referência para dita escolha, sendo um elemento passível de ser detalhado em futuros trabalhos acadêmicos.

A. Implicações e Recomendações para a Prática

A análise apresentada neste trabalho revela a importância de considerar o escopo da fábrica de *software* em cada um dos fatores comparados, na escolha do modelo de gestão do desenvolvimento. Desta forma, surge uma primeira recomendação implícita para evitar o reducionismo de analisar somente o modelo per se no processo decisório, sem contextualizar adequadamente na estrutura da fábrica. Recomenda-se também a utilização (sujeita a adaptações) da matriz de comparação desenvolvida neste trabalho, como ferramenta de abstração e *benchmarking* para a tomada de decisões respeito ao modelo.

B. Limitações do Estudo

Como mencionado anteriormente, existem diversos fatores comerciais, tecnológicos e regulatórios no contexto da fábrica de *software*, que não foram adequadamente considerados neste trabalho por conta das limitações de espaço predefinidas.

Outra limitação conhecida deste trabalho refere à possibilidade de modelos híbridos entre os modelos tradicional e ágil durante a fase de migração, como sugerido em por Cohn [16]. Esta opção foi propositalmente excluída do escopo da análise apresentada neste artigo, no intuito de manter o foco no objetivo da comparação.

Cabe destacar também a ausência neste artigo de casos de estudo diretos apoiando a pesquisa bibliográfica, relativizando os resultados ao marco teórico.

C. Sugestões de Pesquisas Futuras

Conforme destacado nas limitações do estudo, a ausência de casos práticos impõe restrições às conclusões deste trabalho. Uma primeira sugestão é a ampliação do escopo deste artigo com a inclusão de casos de estudo significativos, a fim de estender e validar a análise comparativa dos modelos.

Uma segunda linha de pesquisa recomendada refere-se ao impacto comercial e financeiro da migração do modelo tradicional para ágil nas fábricas de *software*, na expectativa de aportar conclusões valiosas num segmento muito pouco explorado pela bibliografia disponível. Um trabalho dessa natureza visa responder questões centrais para os responsáveis das decisões estratégicas nas fábricas de *software*, que transcendem os aspectos intrínsecos da engenharia.

REFERÊNCIAS

[1] Suganya G.; Mary, S. A. Sahaya Arul. *Progression towards Agility: A comprehensive survey. In:* INTERNATIONAL CONFERENCE ON COMPUTING, COMMUNICATION AND NETWORKING TECHNOLOGIES, 2., 2010, Chettinand. Artigo. Chettinand: Ieee, 2010. p. 1 - 5.

[2] Nerur, Sridhar; Mahapatra, Radhakanta; Mangalaraj, George. *Challenges of Migrating to Agile Methodologies: Organizations must carefully assess their readiness before treading the path of agility.* Communications of the ACM, [s.l.], p. 1-7. May 2005.

[3] Jesson, Jill K.; Matheson, Lydia; Lacey, Fiona M. Doing your Literature Review: Traditional and Systematic Techniques. London: Sage, 2011.

[4] Bemer, R. W.. Machine-controlled production environment. *In:* NATO CONFERENCE ON SOFTWARE ENGINEERING, 1., 1968, Garmisch. Report NATO Conference**.** Garmisch: Nato Conference On *Software* Engineering, 1968. p. 94 - 95.

[5] Cusumano, Michael A.. The Software Factory: A Historical Interpretation. IEEE Software, Los Alamitos, p. 23-30. 2 mar. 1989.

[6] Dias, Leonardo Dominguez. Método de Instanciação de uma Arquitetura de Processos Aplicado em Fábrica de *Software*. 2009. 114 f. Dissertação (Mestrado) - Curso de Engenharia, Universidade de São Paulo, São Paulo, 2009.

[7] Fernandes, Aguinaldo Aragon; Teixeira, Descartes De Souza. Fábrica de *Software*. Brasil: Atlas, 2004. 308p

[8] Rocha, Thayssa Aguila da; Oliveira, Sandro Bezerra; Vasconcelos, Alexandre Lins de. Adequação de Processos para Fábricas de *Software*. In: SIMPÓSIO INTERNACIONAL DE MELHORIA DE PROCESSOS DE *SOFTWARE*, 2004, São Paulo. Simpósio. São Paulo: VI Simpósio Internacional de Melhoria de Processos de *Software*, 2004. p. 131 - 142.

[9] Royce, Winston. Managing the Development of Large Software Systems. IEEE Wescon Proceedings, S. L., n. , p.328-338, ago. 1970.

[10] Boehm, Barry; Turner, Richard. Balancing Agility and Discipline: A guide for the perplexed. Boston: Addison-wesley, 2003. 266 p.

[11] Deemer, Pete; Benefield, Gabrielle. THE SCRUM PRIMER: An Introduction to Agile Project Management with Scrum. Version 1.04. Disponível em http://www.rallydev.com/documents/scrumprimer.pdf, 2007. Acesso em 11/02/2014.

[12] Agile Manifesto, 2001 Disponível em http://www.agilemanifesto.org Acesso em 04/11/2013.

[13] Humphrey; Watts S.. A Discipline for Software Engineering. SEI Series in Software Engineering. Addison-Wesley, 1995.

[14] Sutherland, Jeff. Schwaber, Ken. The Scrum Papers: Nut, Bolts, and Origins of an Agile Framework. 1.1 Cambridge: Scrum Inc, 2007.

[15] Sutherland, Jeff; Schwaber, Ken. The Scrum Guide. 07/2011. Disponível em http://www.scrum.org. Acesso em 09/10/2011.

[16] Cohn, Mike. Desenvolvimento de *Software* com Scrum: Aplicando Métodos Ágeis com Sucesso. Nova York: Bookman, 2011. 496 p.

[17] Cho, Juyun; Kim, Yongseog; Olsen, David. A Case Study on the Applicability and Effectiveness of Scrum Software Development in Mission-Critical and Large-Scale Projects. Amcis 2006 Proceedings, Acapulco, n. , p.3704-3711, 04 ago. 2006.

[18] Batra, Dinesh; Sin, Thant; Yin, Sheng. Modifed Agile Practices for Outsourced Software Projects. Amcis 2006 Proceedings, Acapulco, n. , p.3871-3880, 04 ago. 2006